به نام خدا

جزوه درس:
پردازش موازی (Parallel Processing)

مقطع و رشته:
کارشناسی ارشد | مهندسی کامپیوتر – نرم‌افزار

مدرس:
جناب دکتر علی غلامی رودی
(gholamirudi@nit.ac.ir)

تهیه و تنظیم:
علی رضائی
(ry_bat@stu.nit.ac.ir)

نیم‌سال دوم سال تحصیلی ۱۴۰۲-۱۴۰۱

توجه ۲: با توجه به ترم تحصیلی تدوین جزوه، ممکن است سرفصل‌های تدریس‌شده ترم‌های دیگر توسط مدرس با مطالب داخل جزوه اندکی متفاوت باشد.

توجه ۳: با توجه به سرعت نوشتار در حین تدریس مباحث، ممکن است برخی اشکالات جزئی در جزوه وجود داشته باشد. قبل از استفاده به عنوان مرجع امتحان، حتماً مطالب یکبار مطالعه و بازبینی شود.

پردازش موازی (Parallel Processing)

کتاب های Jájá, Miller

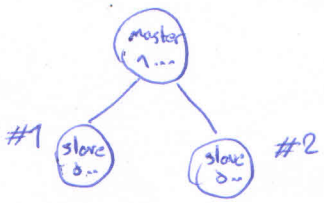
۲۴ آبان ۱۴۰۱ :

مثال : فرض کنیم بخواهیم ۱۰۰۰ عدد را جمع کنیم

```
def sum(A):
    Sum = 0
    for i in Range(0,1000)
        Sum += A[i]
    return sum
```

* فرض کنیم جمع کردن هر دو عدد ۴ واحد زمان برای فرد ببرد. ۸۰۰۰ نانیه

* در نتیجه اگر فردی بخواهد ۱۰۰۰ عدد را جمع کند، حدوداً ۸۰۰۰ نانیه باید وقت بگذارد. یک ایده برای کم کردن محاسبات :



* زمان انتقال هر عدد ۴ واحد از Master به slave ها : ۲ نانیه

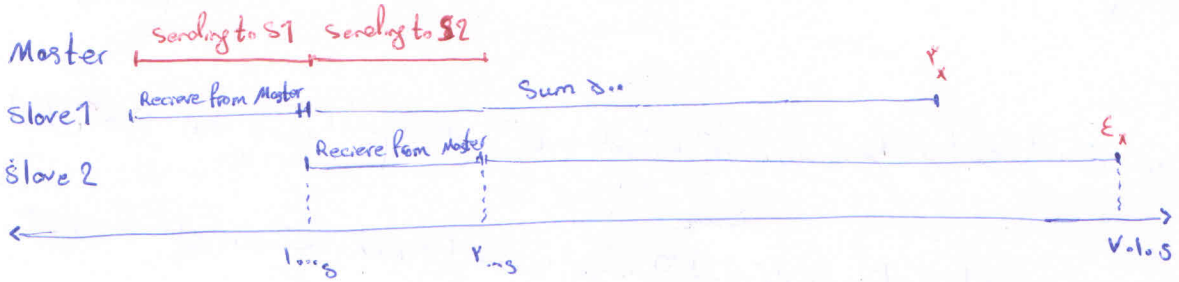
Parallel Algorithm for ...

for slave :

Recieve 500 numbers from Master
res ← SUM(B)
Send res to Master

for Master :

1. Send A[1:500] to slave #1
2. Send A[501:1000] to " #2
3. a ← reciere from slave #1
4. b ← " " " " 2
5. return a+b



$t_{seq} = 8000s$

$t_{par} = 7010s$

$Speedup = \frac{8000}{7010} \simeq 1,14$

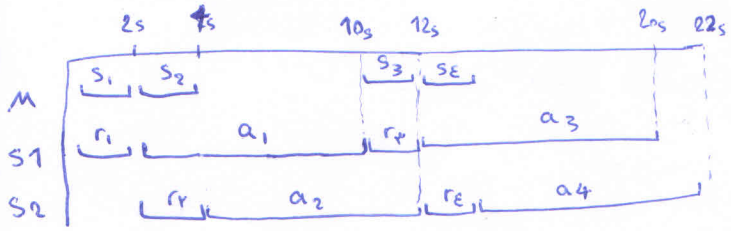
می‌خواهیم الگوریتم قبلی را تغییر دهیم تا به وضعیت بهتر speedup برسیم.

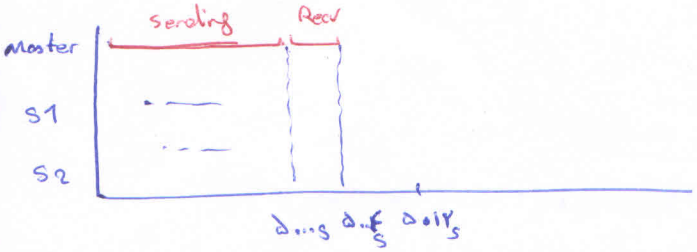
Master:

- for i = 1 to n:
  - if i is odd,
    - send A[i] to slave 1
  - else:
    - " " " " 2
- a ← recieve ---
- b ← recieve ---
- return a+b

slave:

- sum = 0
- for i = 1 to 500:
  - x ← recieve from master
  - sum = sum + x
- send sum to master





$$\Rightarrow t_{Par'} = 5012$$

$$\rightarrow Speedup' = \frac{8000}{5012} \simeq 1,6$$

* چون سخت افزارها توان پردازش موازی دارند، تمایل داریم از الگوریتم‌های موازی استفاده کنیم تا از توان پردازشی استفاده بهینه‌تری داشته باشیم.

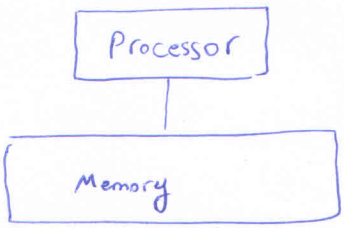
* از طرفی هرجا Bus داشته باشیم، نگاه دسترسی به حافظه با ایجاد محدودیت برای پردازنده مواجه می‌شود.

* همچنین به دلیل محدودیت Disk و یا محدودیت سرعت I/O می‌توان از پردازش موازی بهره برد.

* استفاده از حافظه مشترک نیز می‌تواند نسبت به حافظه غیرمشترک، سرعت را افزایش دهد.

: ۱۴۰۱/۰۱/۱۴

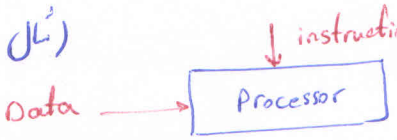
Von Neumann



Flynn's Taxonomy of Parallel Computers

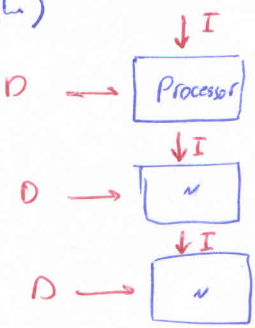
- Data stream
- Instruction stream

(مثال)



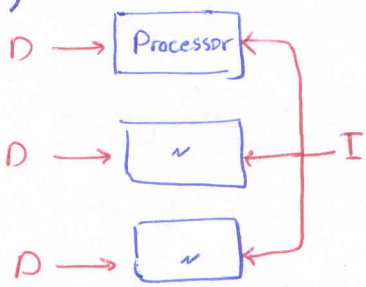
: SISD → Single Instruction Single Data

(مثال)



MIMD: Multiple Instructions Multiple Data

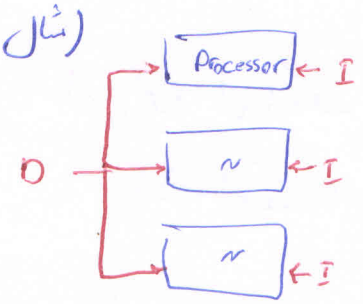
(مثال)



SIMD : Single Instruction Multiple Data

نکته: SSE ، AVX در CPU های امروزی

(مثال)



MISD: عملا کاربرد ندارد

Model of computation

برای طراحی یک الگوریتم جدید (مثلاً در سمت پردازش موازی) ابتدا یک مدل محاسباتی را در نظر می‌گیریم. الگوریتم ترجیحاً باید بدون در نظر گرفتن یک سخت افزار خاص باشد. مثلاً الگوریتم محاسبه فاکتوریل به صورت زیر است:

```
res ← 1
for i = 1 to n
    res ← res * i
return res
```

RAM (Random access Machine)

فرض کنیم الگوریتمی طراحی کرده‌ایم و می‌خواهیم الگوریتم را از نظر بهینه بودن بسنجیم. مثلاً برای الگوریتم فاکتوریل:

$$T(n) = (2n+1)t_{assignment} + nt_{inc} + nt_{mul} + nt_{read} + nt_{write}$$

- با فرض استفاده از مدل RAM:
  - به هرهای حافظه دسترسی در زمان ثابت انجام می‌شود
  - همه عملیات حسابی روی اعداد صحیح در زمان ثابت انجام می‌شود.

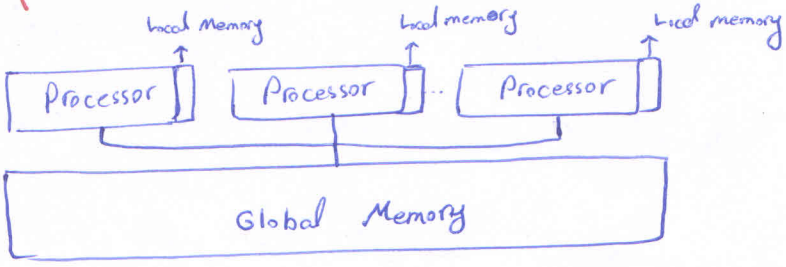
$$\Rightarrow T(n) = O(n)$$

★ مزیت مدل RAM در چنین مسأله‌ای:

- تحلیل و طراحی وابسته به سخت افزار نیست
- قابل پیاده‌سازی روی سخت افزارهای مختلف
- تحلیل و مقایسه الگوریتم‌های مختلف

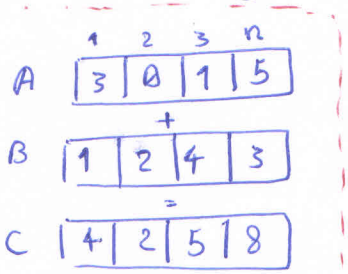
مشکل: این مدل برای یک CPU عملکرد خوبی دارد اما زمانی که چند CPU داشته باشیم جوابگو نیست و باید از مدل‌های دیگری استفاده کنیم.

PRAM (Parallel Random Access Machine)



- همه پردازنده ها به هرجای حافظه سراسری می توانند در زمان ثابت دسترسی داشته باشند.
- عملیات حسابی روی اعداد صحیح توسط هر پردازنده در زمان ثابت انجام می شود.
- همه پردازنده ها هنگام اجرای کشند. در یک Lock step
- همه پردازنده ها در هر گام دستور یکسانی اجرا می کشند.

مثال) فرض کنیم ۲ بردار در حافظه سراسری داریم می خواهیم عناصر متناظر دو بردار جمع شده و بردار حاصل در حافظه سراسری ذخیره شود.



- n بردارهای ورودی
- P پردازنده

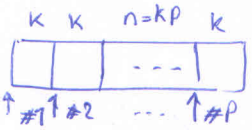
• $P=n$ ، PRAM

- پردازنده i-ام انجام می دهد $(1 \leq i \leq n)$
- $C[i] = A[i]+B[i]$

برای سل RAM :

```
for i=1 to n
    C[i] = A[i]+B[i]
```

• $P<n$ ، $n=kP$ ، PRAM



- پردازنده i-ام انجام می دهد $(1 \leq i \leq n)$
- $start = (i-1)*k + 1$
- for $j = 1$ to k ← حافظه محلی پردازنده
- $C[j] = A[j]+B[j]$

مثالی در وصف بالا: P پردازنده، پردازنده i-ام انجام می دهد:

$j = \frac{i}{\varepsilon} + 1$ ← حافظه محلی

$A[j] = A[j+1] + 1$

فرض اولیه برای A : ۲ ۳ ۱ ۱ ۴ ۲ ۳ ۵

P_1	P_2	P_3
$j \leftarrow 1$	$j \leftarrow 1$	$j \leftarrow 1$
A[2]	A[2]	A[2]
4	4	4
A[1]	A[1]	A[1]

- در این حالت شرایطی پیش می‌آید که هنگام خواندن و نوشتن روی خانه‌های مشترک حافظه به صورت همزمان عملیات انجام شود.

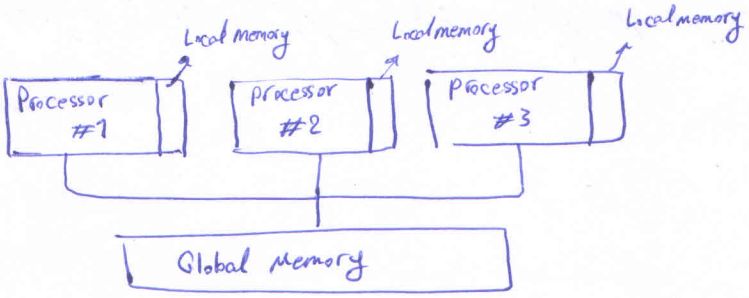
- برای حل این مسأله، PRAM از این جهت به دسته‌هایی تقسیم می‌شوند:

PRAM
- Exclusive
- Concurrent

→
- EREW : Exclusive Read - Exclusive write
- CREW : Concurrent Read - Exclusive write (به پردازنده‌های امروزی شبیه است)
- ERCW : Exclusive " - concurrent " (کاربردی ندارد)
- CRCW : Concurrent " - " "

PRAM (cont.)

۱۴۰۱/۱۲/۰۶ :



نکته مربوط به جلسه قبل: اگر طبق CRCW، خواندن و نوشتن همزمان در یک خانه حافظه صورت گیرد چه اتفاقی رخ می‌دهد؟ به مثال توجه کنیم:

پردازنده i-ام $1 \le i \le P$

اگر
- $x \leftarrow i \bmod 2$
- $y \leftarrow 1 - x$
- $A[x] = A[y]$

P_1: $A[1] \leftarrow A[0]$

P_2: $A[0] \leftarrow A[1]$

پاسخ: صورت سؤال گمراه کننده است. در واقع در مدل PRAM حالتی نداریم که همزمان خواندن و نوشتن روی یک خانه حافظه داشته باشیم. بلکه همزمان یا خواندن داریم یا همزمان نوشتن. مثلاً در دستور مقابل ابتدا A[0] و A[1] هر دو خوانده می‌شوند و سپس هر دو نوشته می‌شوند.

Concurrent write

اگر در رابطه با نوشتن همزمان، چند حالت خواهیم داشت :

- Common : همه پردازنده ها به طور همزمان فقط مقدار یکسانی را توانند بنویسند
- Arbitrary : به صورت تصادفی مقداری از پردازنده ها در حافظه قرار می گیرد
- Priority : پردازنده با اولویت بالاتر موفق می شود
- Merging : حالتی از قبیل +، or، and، xor، *

سوال : فرض کنیم یک الگوریتم A برای CRCW-Poriority داریم. آیا این الگوریتم در حالت های زیر کار می کند؟

- روی CREW ؟ اگر نوشتن همزمان باشد احتمالاً کار نمی کند
- روی CRCW-common ؟ احتمالاً کار نمی کند
- روی CRCW-Arbitrary ؟ اجرای شود اما احتمالاً خروجی نادرست خواهیم داشت

معیارهای مقایسه اجرای الگوریتم های موازی با غیر موازی

- Speed up :

$$Sp(n) = \frac{T^*(n)}{T_p(n)}$$

$T^*(n)$ ← زمان اجرای بهترین الگوریتم ترتیبی

$T_p(n)$ ← پیچیدگی زمانی الگوریتم موازی

* ایده آل زمانی است به ازای P پردازنده، سرعت ما P برابر شود. مثلاً اگر یک نقاش خانه ای را ۱۰ روزه رنگ کند، توقع داریم ۲ نقاش خانه را ۵ روزه رنگ کنند.

- Efficiency :

$$Ep(n) = \frac{Sp(n)}{P}$$

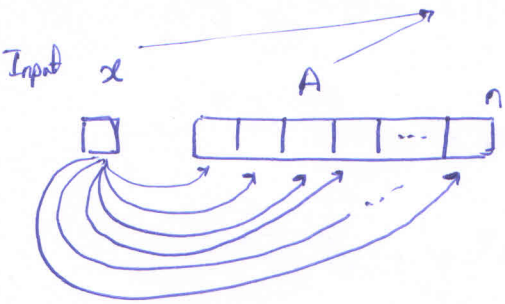
* مقدار ایده آل ۱ است.

- cost :

$$cp(n) = p \cdot T_p(n)$$

* معمولا ایده آل برای $cp(n)$ ، $T^*(n)$ است .

Broadcast (پخش داده)



در حافظه اصلی

- الگوریتم RAM برای Broadcast

```
for i=1 to n
    A[i] = x
```

$T^*(n) = O(n)$

الگوریتم PRAM برای Broadcast به صورت CREW ، $P=n$:

```
پردازنده i-ام به ازای هر i را
A[i] ← n
```

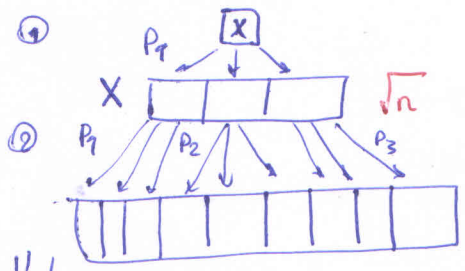
$T_p(n) = O(1)$

$S_p(n) = O(\frac{n}{1}) = O(n)$

$E_p(n) = O(1)$

$C_p(n) = O(n)$

① الگوریتم PRAM در حالت EREW Broadcast



با طول $\sqrt{n}$

① $O(\sqrt{n})$ به صورت ترتیبی توسط P_1 ، x را در آرایه X کپی کن

② $O(\sqrt{n})$ پردازنده i-ام به ازای هر $\sqrt{n}$ تا i را :

- X(i) را در $\sqrt{n}$ خانه به سمت راست با پردازنده i-ام کپی کن .

$T_p(n) = O(\sqrt{n})$

$S_p(n) = O(\frac{n}{\sqrt{n}}) = O(\sqrt{n})$

$C_p(n) = O(n)$

- هدف از طراحی الگوریتم های موازی چیست؟

۱- مقیاس پذیری (Scalability)

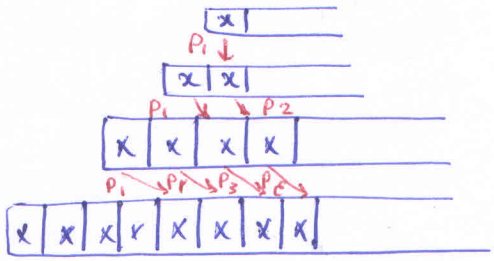
- مدل PRAM چقدر عملی است؟

- تعداد پردازنده های پیاده سازی ها خیلی محدودتر است.
- انتقال داده ها بین پردازنده و حافظه سراسری
- و ...

استفاده از $\log^n$ بجای $\sqrt{n}$ در الگوریتم ①

$$T_p(n) = O(\log^n + \frac{n}{\log^n}) = O(\frac{n}{\log^n})$$
$$S_p(n) = O(\frac{n}{\frac{n}{\log^n}}) = \log^n$$

الگوریتمی دیگر از PRAM برای EREW Broadcast



$O(1)$ - $A[1] \leftarrow x$

$O(\log^n)$ - for $i = 1$ to $\log^n$

$O(1)$ [- • Processor j executes $1 \leq j \leq 2^{i-1}$
- • $A[2^{i-1} + j] = A[j]$

$P(n) = O(n)$

$T_p(n) = O(\log^n)$

$S_p(n) = O(\frac{n}{\log^n})$

$C_p(n) = O(n \log^n)$

۱۴۰۱/۱۲/۰۸

Work (میزان کار انجام شده در یک الگوریتم)

$$W(n) = \sum_{i=0}^{T(n)} w_i$$

کار انجام شده در گام i-ام الگوریتم

=

تعداد پردازنده‌های مشغول در گام i-ام

الگوریتم بتل :

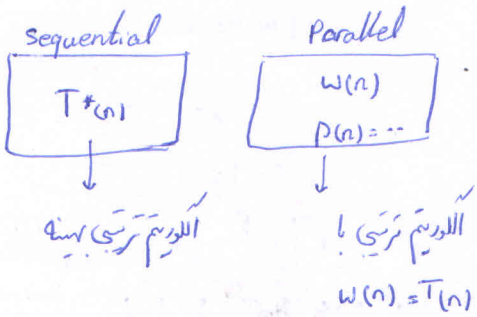
$$T(n) = O(\log^n)$$

$$W(n) = 1 + 2 + 4 + \cdots + 2^{\log^{n-1}} = 2^{\log^n} - 1 = O(n)$$

$$C(n) = T(n) \cdot P(n) = O(n \log^n)$$

رابطه $W(n)$ ، $T^*(n)$ ، $C(n)$ ، $T(n)$:

- $C(n) \geq W(n)$
- $T^*(n) \leq W(n)$
- $T(n) \leq W(n)$



نکته : الگوریتم work optimal : $W(n) = T^*(n)$

الگوریتم cost optimal : $C(n) = T^*(n)$

نکته : الگوریتم بتلی work optimal هست اما cost optimal نیست.

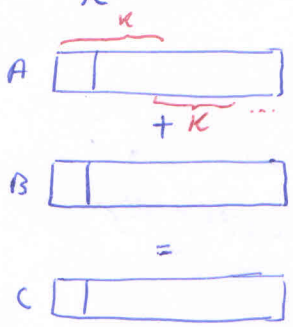
Work-Time presentation Fromework

پیرو مثال بردار :

- Processor i executes $1 \le i \le P$
  - for $j = (i-1)K + 1$ to iK
    - $C[j] = A[j] + B[j]$

$$T(n) = O\left(\frac{n}{P}\right)$$

$$P = \frac{n}{K}$$



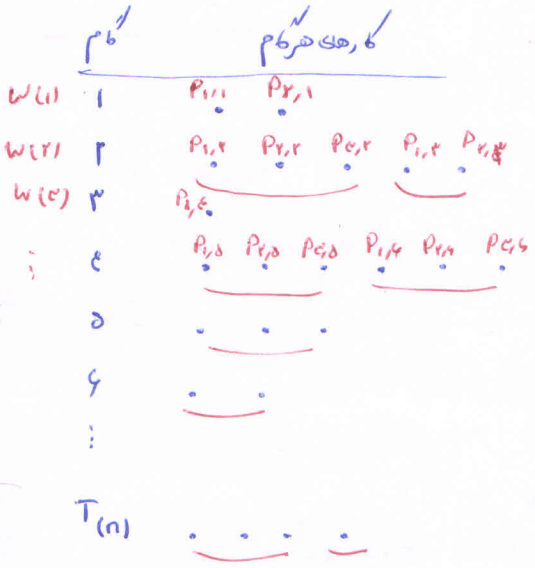
نمایشی دیگر از نمایش بالا

- for $i=1$ to n Par
  - $C(i) = A(i) + B(i)$

$$P(n) = O(n)$$
$$T(n) = O(1)$$
$$W(n) = O(n)$$

مزایای این شیوه نمایش
- شبیه الگوریتم ترتیبی
- پردازنده های نامحدود
- محاسبه کار
- تحلیل ساده تر

نکته در مثالی که تنها فقط P پردازنده داشته باشیم.



$$T_P(n) = \sum_{i=1}^{T(n)} \left\lceil \frac{w_i}{P} \right\rceil < \sum_{i=1}^{T(n)} \left(\frac{w_i}{P} + 1\right)$$

$$\le \frac{\sum_{i=1}^{T(n)} w_i}{P} + \sum_{i=1}^{T(n)} 1$$

$$\le \frac{W(n)}{P} + T(n)$$

فقط P پردازنده

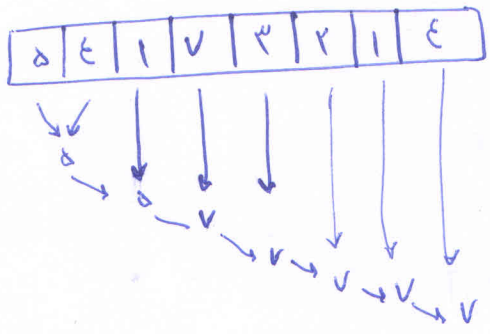
$P = O(P(n))$

$$\Longrightarrow T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$$

work-Time scheduling principle

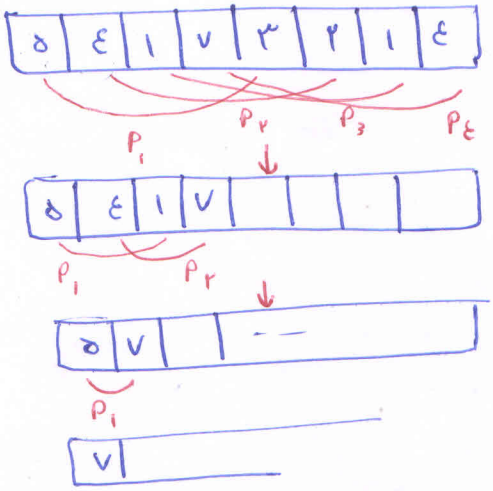
Reduction EREW

اگر sequential کنیم :



در sequential اعداد ۱ تا ۸ تا مقایسه می‌شوند. (به ترتیب از چپ به راست آرایه)

اگر به صورت موازی و در هر مرحله ۲ تا ۲ تا مقایسه کنیم :



```
. for i = 1 to Log n        O(Log n)
    O(1) { . for j = 1 to n/2^i     Par
           . A[j] = max(A[j], A[j + n/2^i])
  . Return A[1]
```

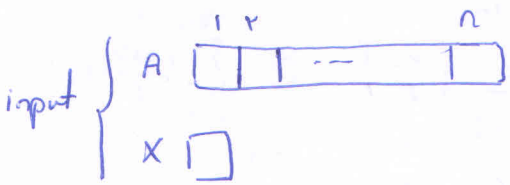
$T(n) = O(\log n)$

$P(n) = O(n)$

$W(n) = O(n)$ ← در واقع مثل مسأله قبلی است اما برعکس محاسبه می‌شود. یعنی برعکس $1+2+4+\dots+2^{\log n - 1}$ که نهایتاً برابر $(1-2^{\log n})$ و برابر $O(n)$ خواهد بود.

Searching in PRAM

۱۴.۱ ، ۱۲، ۱۳



$T^*(n) = O(n)$

به ازای مقایسه تک تک اعضای آرایه با X در الگوریتم RAM

Output [مقدار X یا A یا خیر]

```
res ← false
for i = 1 to n   Par?
    if A[i] = X :
        return True   # res ← True
```

این الگوریتم در EREW PRAM قابل اجرا است چون X را باید همزمان بخواند.

Searching in EREW PRAM

(i)

1. $X \leftarrow$ EREW-Broadcast(x)

   $Y \leftarrow$ یک آرایه‌ی بولی به طول n

2. for $i=1$ to n par

   $Y[i] =$ false

(ii)

3. for $i=1$ to n par

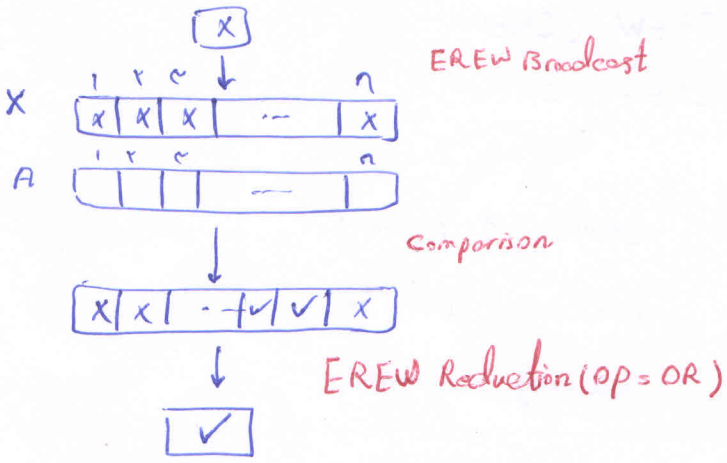
   if $A[i] = X[i]$:

   $Y[i] =$ True

(iii)

4. res $\leftarrow$ EREW Reduction OR (Y)

5. return res



$P(n) = O(n + n + n + n)$ (i) (iv) (iii)

$T(n) = O(\text{Log}^n + 1 + 1 + \text{Log}^n)$ (i) (i) (iii)

$S(n) = O\left(\frac{n}{\text{Log}^n}\right)$

$C(n) = O(n\,\text{Log}^n) \longrightarrow$ not costoptimal

$W(n) = O(n) \longrightarrow$ work optimal

نکته: اگر از CREW استفاده می‌کردیم، گام (i) لازم نبود، اما پیچیدگی زمانی همچنان تغییری نمی‌کرد.

Searching in CRCW PRAM (فرض می‌کنیم از نوع Common است)

$y \leftarrow$ false

for $i=1$ to n par

  if $A[i] = X$:

    $y \leftarrow$ True

return y

$T(n) = O(1)$

$P(n) = O(n)$

$C(n) = O(n)$ cost optimal

$W(n) = O(n)$ work optimal

چندنکته : قدرت مدل X از ۲ بیشتراست $X > 2$ (هرالگوریتمی برای ۲ در X هم کار می‌کند)

EREW < CREW

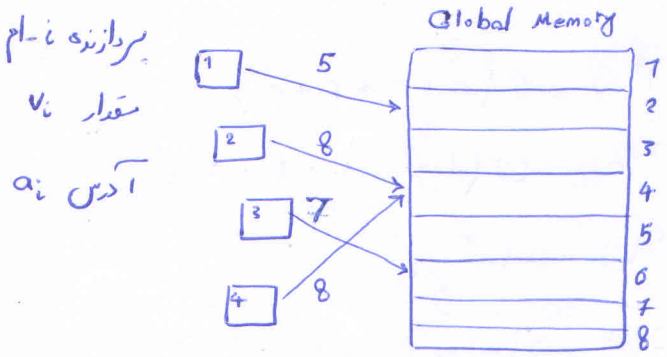
CREW < CRCW

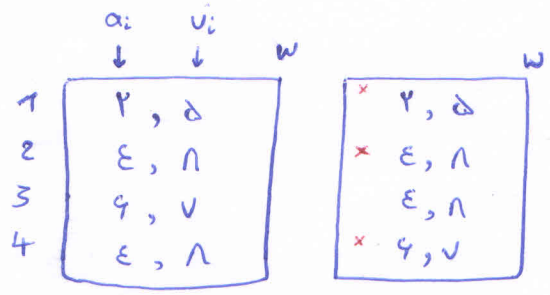
Common < pariority

Arbitrary < pariority

Arbitrary > Common

شبیه سازی نوشتن همزمان

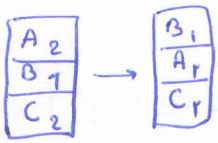




گام ۱ : مقداردهی آرایه

گام ۲ : ترتیب سازی براساس a_i (stable)

اگر متغیر یکان را داشتیم، ترتیب آن‌ها باید حفظ شود. مثلاً :



گام ۳ : عناصر اول هر آدرس به حافظه نوشته شوند.

هر گام نوشتن به صورت زیر انجام می‌شود :

فرض : P_i مقدار v_i را به a_i ی خواهد نوشت.

- for $i = 1$ to p : par
  $w[i] = (a_i, v_i)$
- EREW Sort (w) ← a_i باتوجه به
- for $i = 1$ to p : par آدرس
  if $i = 1$ or $w[i].a \neq w[i-1].a$:
  write $w[i].v$ to $w[i].a$
  ↳ مقدار

$P(n) = O(p)$

$T(n) = O(\text{Log } p)$

الگوریتم A برای CRCW Common ← شبیه سازی برای CREW

هر گام ← $O(Log^p)$

کل الگوریتم ← $O(f(n).Log^p)$

نکته: ممکن است حالتی داشته باشیم که به الگوریتمی برسیم که از حالت بالا بهتر عمل کند. درواقع لزوماً پایین بینه توسط مرزهای بالا مطرح نیست اما تضمین می کند حالتی نیست که بدتر از آن باشد.

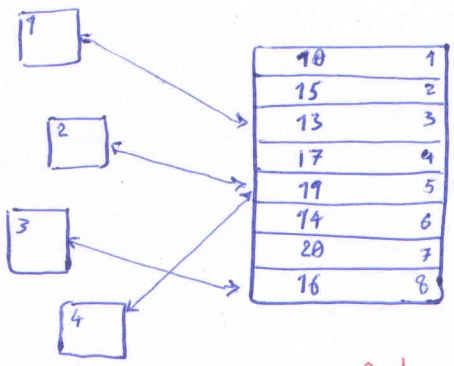
Search CREW $T(n) = O(1)$, $P(n) = O(n)$

↓

CREW $T(n) = O(Log^n)$, $P(n) = O(n)$

۱۴.۱ ، ۱۲ ، ۱۵ :

Simulating Concurrent Read



آدرس خانه شده توسط P_i

$R(a_i , i)$

① (۳,۱) (۵,۲) (۸,۳) (۵,۴)

② Sort

(۳,۱) (۵,۲) (۵,۴) (۸,۳)

در مورد شبیه سازی ترایم مورد اول را بخوانیم و بای را skip کنیم

③ درخواست های اول از هر آدرس در R خوانده می شوند و به آرایه V انتقال می یابند

V: 13, 11, 11, 16

④ خانه های خالی را با مقدار قبلی پر می کنیم

V: 13, 11, 11, 16

⑤ ترتیب خواندن را به صورت ترتیب پردازنده ها برمی گردانیم

X: 13, 19, 16, 17

(با استفاده از Interval Broadcast از EREW PRAM)

$T(n) = O(Log^n)$

$W(n) = O(n)$

$O(1)$. for $i=1$ to n: Par — آدرس حافظه خوانده شده توسط P_i — $O(n)$

$R[i] = (a_i, i)$

$O(\log^2 n)$. EREWSORT (R) → a_i با توجه به — $O(n \log^2 n)$

$T(n) = O(\log^2 n)$

$W(n) = O(p \log^2 p)$

$O(1)$. for $i=1$ to n: Par — $O(n)$

if $i=1$ or $R[i].a \neq R[i-1].a$:

$V[i] \leftarrow$ محتویات آدرس $R[i].a$

$O(\log^2 n)$. Interval Broadcast (V) — $O(n)$

$O(1)$. for $i=1$ to n: Par — $O(n)$

$X[R[i].index] = V[i]$

فرض: الگوریتم A با خواندن همزمان داریم، به طوری که:

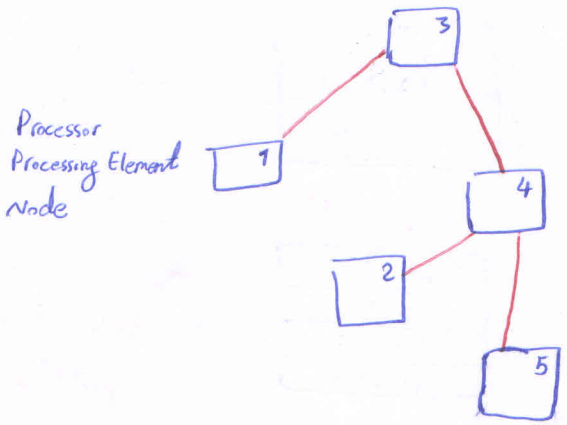
$T(n) = O(f(n))$ ، $W(n) = O(h(n))$

پس از شبیه سازی خواندن همزمان:

$T'(n) = O(f(n) \cdot \log^p)$ ، $W'(n) = O(h(n) + f(n) \cdot p \log^p)$

← کار انجام شده برای شبیه سازی

Network Model



- حافظه مشترک ندارد.
- پردازنده‌ها شبیه هستند.
- بین پردازنده‌ها لینک‌های دوطرفه هستند.
- در هر گام 1 پیغام می‌تواند از یک طرف لینک به طرف دیگر فرستاده شود.
- دو عمل برای تبادل پیغام: { Send / recv

انتقال یک مقدار از P_1 به P_5

P_1 : Send(3, X) ①

P_3 : X ← Recv(1)

P_3 : Send(4, X) ②

P_4 : X ← Recv(3)

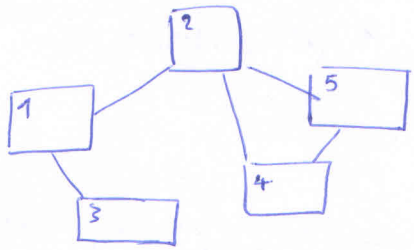
P_4 : Send(5, X) ③

P_5 : X ← Recv(4)

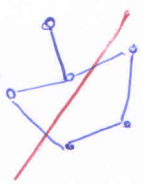
۱۴۰۲/۰۱/۱۹ :

بیان ترم برای ۱۴۰۲/۰۳/۰۱ تنظیم شد.

معیارهای مقایسه Topology های شبکه :



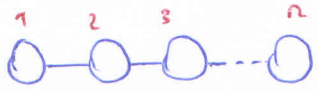
- قطر (Diameter) : بزرگترین کوتاه‌ترین مسیر موجود در Topology شبکه را قطر می‌گوییم. مثلاً در شکل بالا قطر برابر ۳ است.
- درجه (Degree) : حداکثر لینک‌هایی که به یک نود وصل شده. در شکل بالا ۳ است.
- همبندی رأسی (VertexConnectivity) : چه تعداد رأس اگر حذف شود، شبکه از همبندی خارج می‌شود؟ (حداکثر تعداد رأس‌های که اگر حذف شوند، شبکه ناهمبندی شود) در شکل بالا اگر نود ۱ یا ۲ حذف شود، شبکه ناهمبندی شود. پس VC=1
- همبندی یالی (EdgeConnectivity) : ماکسیمم تعداد یال‌های که اگر حذف شوند، شبکه ناهمبندی شوند. در شکل بالا EC=1
- عرض تنصیف (Bisection width) : حداقل یال‌هایی که بین دو بخش گراف با رأس‌های برابر قرار می‌گیرند. (رأس‌های گراف به دو بخش تقسیم شوند) در شکل زیر ۲ یال است.



- تعداد گام های لازم برای عملیاتی مثل : Min ، Sum ، Sort ، Broadcast ، ...
- تعداد لینک های شبکه

چند مثال از توپولوژی های شبکه :

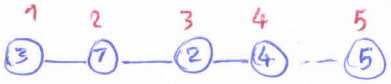
- Linear Array (آرایه خطی)



قطر : $n-1$ حداکثر درجه : 2

هندسی (رأسی دیالی) : 1 عرض Bisection : 1

مثال : فرض کنیم می خواهیم کمینه مقادیر درون آرایه ها در پردازنده 1 قرار گیرد .



عدد i ام ، ذخیره : $A[i]$ شده در حافظه محلی P_i

ورودی : کمینه باید در P_1 قرار گیرد .

رفتار هر پردازنده i ام :

```
if i < n :
    - X ← recv(i+1)
    - A[i] = min(A[i], X)
if i > 1 :
    - send(A[i], i-1)
```

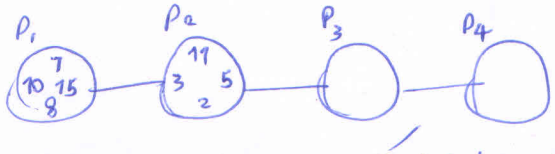
$P(n) = n$

$T(n) = O(n)$

$C(n) = O(n^2)$ کارایی ات

$W(n) = O(n)$ هزینه بهینه یت

اگر فرض کنیم به تعداد اعداد مورد نظر پردازنده نداریم ، $P(n) = \sqrt{n}$. به عبارتی هر پردازنده بیش از یک عدد خواهد داشت .



ورودی : هر پردازنده $\sqrt{n}$ عدد نامه ای دارد .

خروجی : کمینه اعداد به P_1 منتقل شود .

(ابتدا کمینه هر پردازنده محاسبه می شود ، سپس الگوریتم قبل)

رفتار پردازنده i ام اجرای کند :

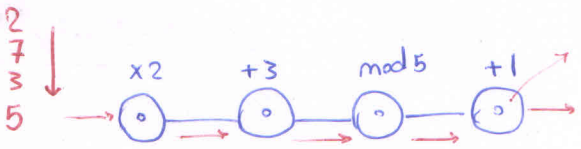
$O(\sqrt{n})$. کمینه $\sqrt{n}$ عدد حافظه محلی P_i را محاسبه کن ← $A[i]$

$O(\sqrt{n})$. شبیه الگوریتم قبلی کمینه را به P_1 انتقال بده

$P(n) = \sqrt{n}$ $C(n) = O(n)$ کارایی ات

$T(n) = O(\sqrt{n})$ $W(n) = O(n)$ هزینه بهینه ات .

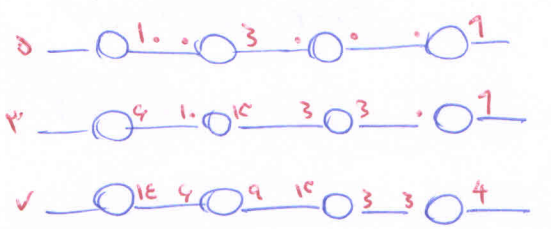
آرایه تپنده (Systolic Array)



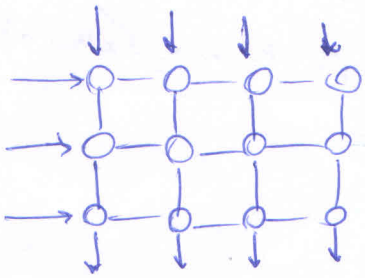
ورودی‌ها به صورت آرایه‌ای وارد شده (از یک طرف) و از طرف دیگر خارج می‌شوند.

P_1:
```
x ← recv(P0)
send(P2, x*2)
```

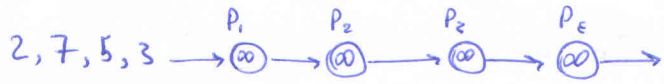
P_2:
```
x ← recv(P1)
send(P3, x+3)
```



آرایه تپنده مسئله‌ات فقط خطی نباشد و ورودی و خروجی آن به صورت زیر باشد:



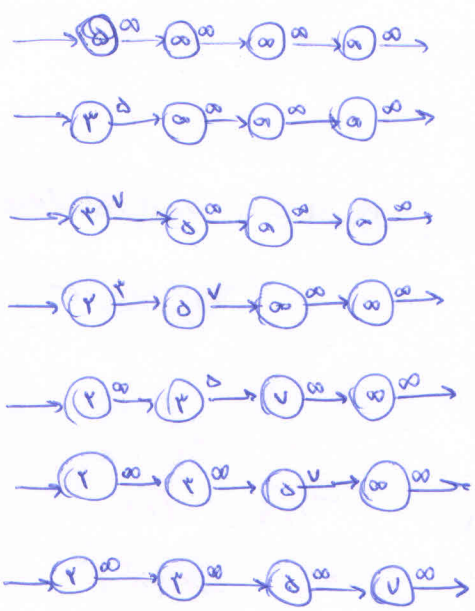
مرتب‌سازی اعداد با آرایه تپنده



ورودی: n عددی که از چپ وارد می‌شوند

خروجی: $A[i]$ عدد i-ام در ترتیب مرتب شده باشد.

حافظه پردازنده P_i



پردازنده i-ام

```
for j=1 to 2n:
  . x ← recv(P_{i-1})
  . y ← min(A[i], x)
  . z ← max(A[i], x)
  . A[i] ← y
  . send(P_{i+1}, z)
```

$T(n) = O(n)$

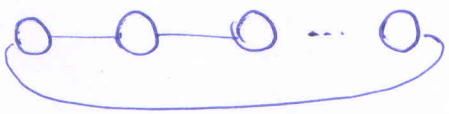
$P(n) = O(n)$

$W(n) = O(n^2)$

$C(n) = O(n^2)$

۱۴۰۲/۰۱/۲۱

- Ring :



Diameter : $\frac{n}{2}$

(Edge/node) Connectivity = 2

Bisection width : 2

محاسبه Min :

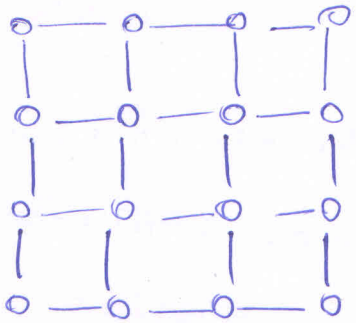
هر پردازنده یک عدد $A[i]$

خروجی : کمینه در P_1 قرار گیرد

(در $\frac{n}{2}$ گام قابل انجام است)

نکته : فرض کنیم شبکه x با نظر k داریم . (کمینه به یک رأس مشخص داده شود) با معروفات ، می‌توان گفت حداقل دور دارد که مسأله کمینه در k انجام شود.

- Mesh $_{n \times n}$ (2D Grid)



Diameter : $2n-2$

Connectivity : 2

Bisection width : n

یکراه برای محاسبه Min در این توپولوژی این است که ابتدا کمینه هر سطر محاسبه شود و سپس در یک ستون ، کمینه کمینه ها محاسبه شود.

$P_{i,j}$: پردازنده سطر i و ستون j

$A[i,j]$: عدد پردازنده $P_{i,j}$

محاسبه کمینه به صورت سطری

محاسبه کمینه به صورت ستونی

```
if j < n:
    x ← recv(P_{i,j+1})
    A[i,j] ← Min(x, A[i,j])
if j > 1:
    send(P_{i,j-1}, A[i,j])
if i < n:
    x ← recv(P_{i+1,j})
    A[i,j] ← Min(x, A[i,j])
if i > 1:
    send(P_{i-1,j}, A[i,j])
```

$P(n) = O(n^2)$

$T(n) = O(n)$

$w(n) = O(n^2)$

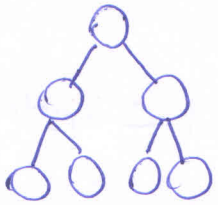
$C(n) = O(n^3)$

$T^*(n) = O(n^2)$

work optimal

not cost optimal

- Tree



Diameter : $2\log^n$

Edge/Node Connectivity : 1

Bisection width : 1

الگوریتم Broadcast در توپولوژی درخت : (یک عدد در ریشه ← به همه رأس ها انتقال یابد)

- اگر رأس ریشه باشد :

x ← عدد ذخیره شده در رأس

- در غیر این صورت :

x ← از پدر رأس عدد را دریافت کن (Recv(x))

- به ازای هر فرزند رأس مثل c :

x را به c بفرست Send(c,x)

$P(n) = n$

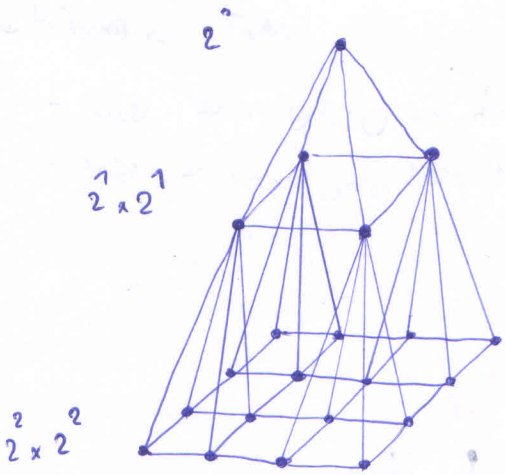
$T(n) = O(\log^n)$

work optimal $W(n) = O(n)$

Not cost optimal $C(n) = O(n\log^n)$

نکته : بواسطه Bisection w=1 ، چون دوطرفه در هر مرحله فقط یک متغیر می تواند جابه جا شود، الگوریتمی مثل مرتب سازی را نمی توانیم در $\log^n$ انجام دهیم.

- Pyramid (مخروط چندضلعی)



سطح i ← یک مش $2^i \times 2^i$

تعداد رأس ها : $2^0 \times 2^0 + 2^1 \times 2^1 + 2^2 \times 2^2 + \dots + 2^n \times 2^n$

$= 2^0 + 2^2 + 2^4 + \dots + 2^{2n} < 2^{2n+1}$

Degree : $3 \leq$ درجه رأس ها ≤ 9

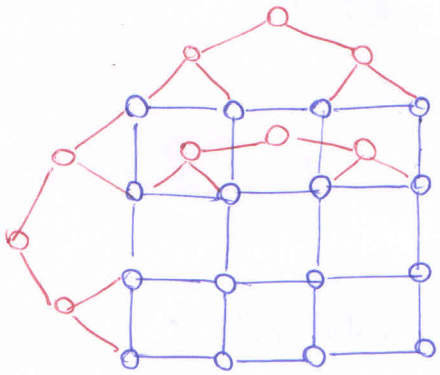
Diameter : $2n$

connectivity : 3

Bisection width : $2 + 2^1 + 2^2 + 2^3 + 2 + \dots + 2^n = 2^{n+1}$

- Mesh of Trees

$2^n \times 2^n$

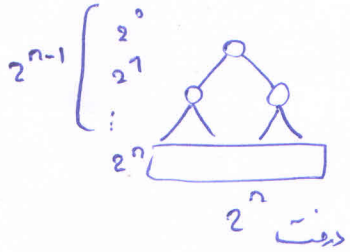


برای هر سطر و هر ستون یک درخت خواهیم داشت!

تعداد رأس‌ها : $2^{2n} + 2 \times 2^n \times (2^n - 1)$

(رأس‌های مشترک: 2^{2n}، تعداد درخت‌ها: 2×2^n، تعداد رأس‌های جدید هر درخت: $2^n - 1$)

$= 3 \times 2^{2n} - 2^{n+1}$



Max Degree = 6

۱۴۰۲/۰۱/۲۶ :

ارائه مبحث قبل :

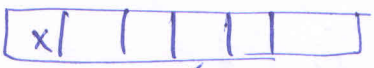
مثل : مرتب سازی n عدد در Mesh of Trees $n \times n$

Note : $n = 2^k$

→ تعداد رأس‌ها : $n^2 + 2n(n-1) = 3n^2 - 2n$

- درودی : n عدد در سطر اول MoT
- خروجی : اعداد درودی به ترتیب در سطر اول ظاهر شوند.

Note : Broadcast in Linear Array :



پردازنده i-ام اجرای کند :

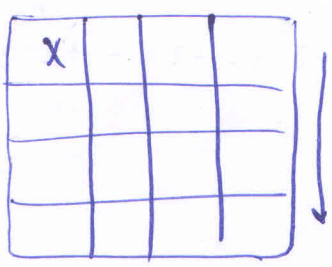
```
if i > 1 :
    A(i) ← Recv(i-1)
if i < n :
    Send(i+1, A(i))
```

★ Broadcast در MoT

- درودی : عددی در یکی از رأس‌های Mesh
- خروجی : عدد به همه رأس‌های Mesh انتقال یابد.

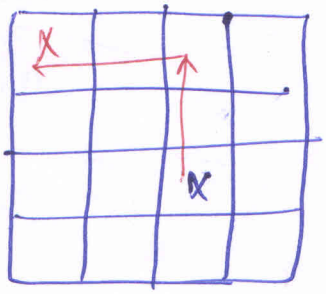
یک سطر اول Broadcast
سپس تمام ستون‌ها Broadcast

نکته ① : ① به صورت سطری



② به صورت ستونی

نکته ② :



ابتدا X را به سطر اول می‌بریم سپس الگوریتم شکل چپ را اجرا می‌کنیم.

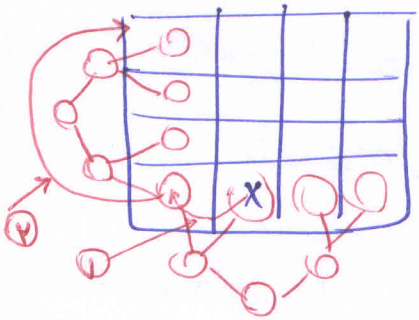
با توجه به نکته ۱ و ۲ ، اگر بخواهیم از درخت برای Broadcast در MoT کمک بگیریم :

۱- استفاده از درخت سطر اول برای انتقال عدد به همه رأس های سطر اول $O(\log n)$

۲- به ازای همه ستون ها : $O(\log n)$

$T(n) = O(\log n)$ ⇐

عدد سطر اول را به رأس های ستون انتقال بده



* اگر x در خانه ای غیر از خانه اول باشد ، ابتدا با دو درخت به خانه اول انتقال می یابد ، سپس الگوریتم بالا برای Broadcast اجرای شود .

آرایه A_i باشد :

① با استفاده از درخت ها به $A_{i,1}$ انتقال می دهیم

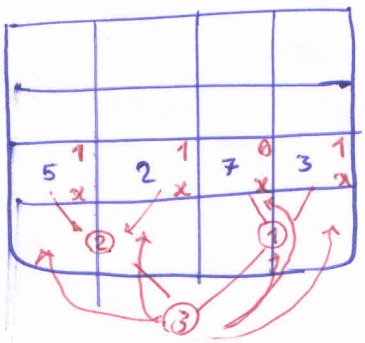
② الگوریتم بالا اجرای شود .

عملیات کاهش : یافتن تعداد اعداد کوچکتر از دوری در یک سطر (الگوریتم ①)

ورودی : یک عدد

خروجی : تعداد اعداد کوچکتر از آن عدد ← تعداد دهه خانه ها تراز کند .

$T(n) = O(\log n)$



مثال : $x = 6$

W

$O(n)$ ۱- x را به صورت سطری Broadcast می کنیم

$O(n)$ ۲- هر رأس عددش را با x مقایسه می کند :

اگر برقرار باشد ۱ و در غیر این صورت ۰ خروجی است .

$O(n)$ ۳- محاسبه حاصل جمع مقایسه ها ← حل کاهش در درخت سطری

$O(n)$ ۴- انجام Broadcast از ریشه درخت سطری ← حل Broadcast درخت سطری

انجام عملیات کاهش در درخت (مثل جمع) :

رأس انجام دهنده :

۱- اگر برگ نباشد :

x ← دریافت از فرزند چپ

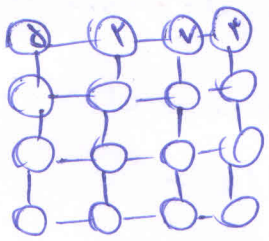
y ← " " راست

$x+y$ ⟸ مقدار ذخیره شده در رأس

۲- اگر ریشه نباشد

مقدار را برای پدر بفرست

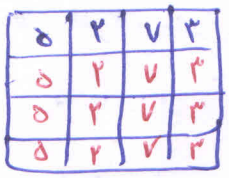
اکنون به مسأله اول یعنی مرتب سازی n عدد در $MoT_{n\times n}$ می‌پردازیم:



۱- اعداد سطر اول را به صورت ستونی Broadcast می‌کنیم. ← $O(\log n)$

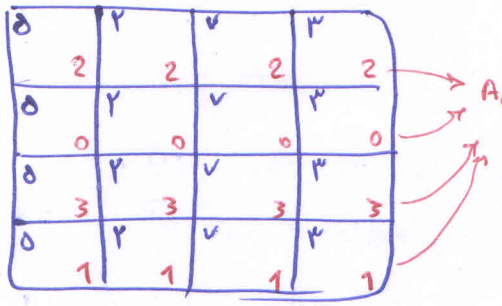
۲- به ازای هر سطر i به صورت موازی: ← $O(\log n)$

تعداد اعداد کوچکتر از $A_{1,i}$ را برای سه می‌کنیم ← با الگوریتم ① صفحه قبل



۳- به ازای هر سطر i به صورت موازی: ← $O(\log n)$

مقدار $A_{1,i}$ را Broadcast کن

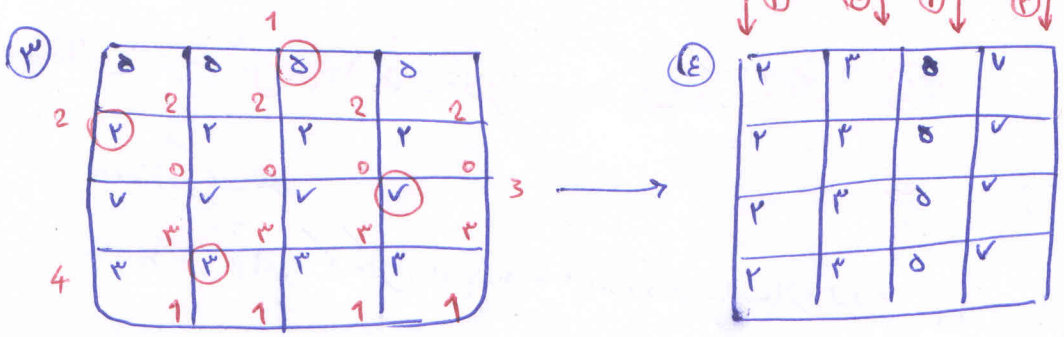


۴- به ازای هر سطر i به صورت موازی: ← $O(\log n)$

$S \leftarrow$ تعداد اعداد کوچکتر $A_{1,i}$

خانه‌ای را که شماره‌اش برابر $S+1$ است به صورت ستونی Broadcast انجام می‌دهیم.

نوبت: ③ ④ ① ②



۵- به ازای هر سطر دستور انجام می‌شود پس ۱، ۲، ۳، ۴ ← $W(O(n^2))$

$T(n) = O(\log^2 n)$

Hyper Cube (ابَرمکعب)

۲۸ ، ۰۱ ، ۱۴۰۱ :

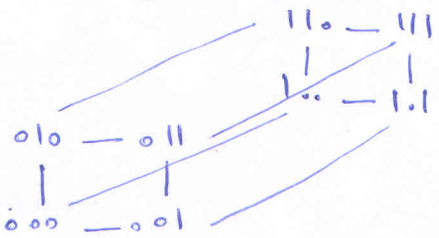
H_K ابرمکعب K بعدی ⟵ 2^K رأس دارد

هر رأس ⟵ یک عدد

$K = 3$

0
1
⋮
$2^K - 1$

رأس هایی که در یک بیت باهم تفاوت دارند توسط یک یال به یکدیگر متصل می شوند.



← بین دو رأس که عددشان در یک بیت اختلاف دارد یک یال اضافه می کنیم.

Degree = K

Diameter = K

Connectivity (Edge/Node) = 3

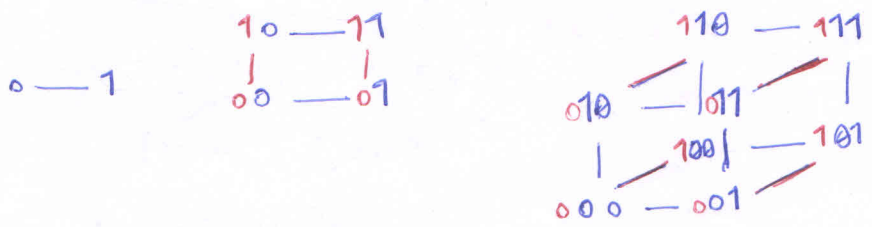
Bisection width = 2^{K-1}

# of Links = $\frac{K \times 2^K}{2} = K \times 2^{K-1}$

درجه هر رأس ← تعداد رأس ها ← تعداد تکرار در شمارش

ساختار بازگشتی تولید ابرمکعب :

H_1 H_2 H_3



عملیات Broadcast در Hypercube

ورودی: عددی در رأس شماره 0 و $A(i)$: عدد ذخیره شده در رأس i-ام

خروجی: انتقال عدد به همه رأس های آن مکعب

نکته: از ایده ساختار بازگشتی برای Broadcast استفاده می کنیم. الگوریتم در k گام انجام می شود.

- به ازای i از k تا 1 تکرار کن:
  - پردازنده‌ی j به ازای $0 \le j < 2^{i-1}$:
    - $Send(j+2^{i-1}, A(j))$
  - پردازنده j به ازای $2^{i-1} \le j < 2^i$:
    - $A(j) \leftarrow Recv(j-2^{i-1})$.

این پردازنده ها به صورت سوازی انجام می دهند

عملیات کاهش در HyperCube

می توانیم با ساختار بازگشتی و عکس الگوریتم Broadcast استفاده کنیم. یعنی H_3 مقادیر خود را به H_2 بفرستد و به همین صورت

ورودی: عدد $A(i)$ در پردازنده i-ام

خروجی: حاصل کاهش (مثلاً جمع) در پردازنده 0

به ازای i از k تا 1 تکرار کن:
- پردازنده j به ازای $2^{i-1} \le j < 2^i$
  - $Send(j-2^{i-1}, A(j))$.
- پردازنده j به ازای $0 \le j < 2^{i-1}$
  - $X \leftarrow Recv(j+2^{i-1})$
  - $A(j) \leftarrow A(j)+X$

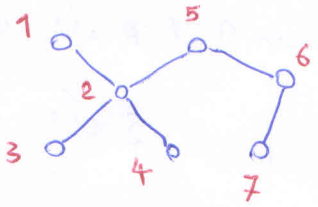
$T(n) = O(\log n)$

$W(n) = O(n)$

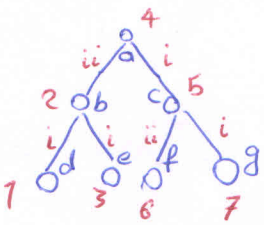
	HC	Mesh
قطر	$O(\log n)$	$O(\sqrt{n})$
عرض	$O(n)$	$O(\sqrt{n})$

Network Embedding / Remapping

Source



Target



ابتدا ساختار سمت چپ را به سمت راست نگاشت می‌کنیم.

1-2 : ⟶ b-d
2-3 : ⟶ b-e
2-4 : ⟶ b-a
2-5 : ⟶ b-a-c
5-6 : ⟶ c-f
6-7 : ⟶ f-c-g

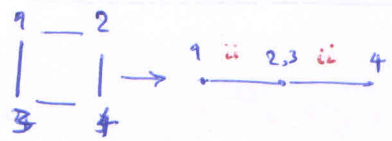
→ Dialation = 2

Dialation (کشیدگی) : بیشترین طول یال پس از نگاشت

congestion (ازدحام) : بیشترین تعداد یال شبکه میزبان که به یک یال شبکه مقصد نگاشت شده‌اند.
(از روی یال‌های تکراری در کشیدگی هم می‌توان محاسبه کرد.)

مثال:

Dialation = 1
congestion = 2



Prefix Sum

A 2 5 3 4 7

Prefixes:
- 2
- 2+5 = 7
- 2+5+3 = 10
- 2+5+3+4 = 14
- 2+5+3+4+7 = 21

A	5	2	4	1

n

S	5	7	11	12

n

الگوریتم ترتیبی برای محاسبه جمع پیشوندی

ورودی: آرایه A با n عنصر

خروجی: آرایه S

$T(n) = O(n)$

$A(1) \longrightarrow S(1)$

به ازای i از ۲ تا n:

$S(i-1) + A(i) \longrightarrow S(i)$

	1	2	3	4	5	6	7	8
A	3	5	7	1	4	2	8	6

	A_1+A_2	A_3+A_4	A_5+A_6	A_7+A_8
B	8	8	6	14

$A_1+A_2+A_3+A_4$

	A_1+A_2			
S_B	8	16	22	36

S_A		8		16		22		36

```
EREW PrefixSum(A, n):        (A: Array, n: Length of A)
S: output  B, T: Arrays of size n/2
if n==1 → return A
for i=1 to n/2 :  Parallel                          O(1)
    . B(i) ← A(2i-1) + A(2i)
. T ← EREW PrefixSum(B, n/2)
. for i=1 to n :                                    O(1)
    . if i is even :
        . S(i) = T(i/2)
    . else :
        . S(i) = T((i+1)/2) - A(i+1)
. return S
```

$T(n) \leq c + T(\frac{n}{2}) \longrightarrow T(n) = O(\log n)$

$n \quad c$
$\frac{n}{2} \quad c$
$\frac{n}{4} \quad c$

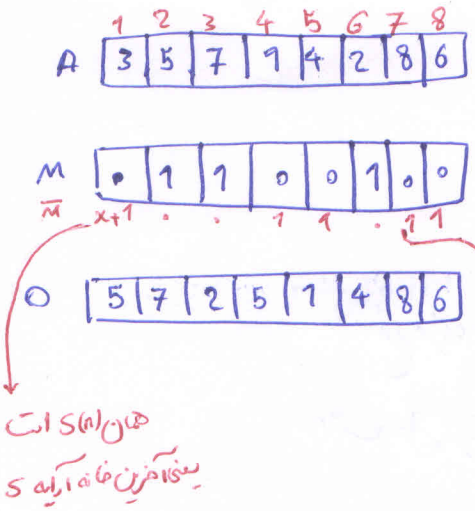
$W(n) \leq Cn + W(\frac{n}{2}) \longrightarrow W(n) = O(n)$ work optimal

$n \quad nc$
$\frac{n}{2} \quad \frac{n}{2}c$

$P(n) = O(n)$

$Cost(n) = O(n \log n)$ not cost optimal

: Array Packing



آیا به صورت خطی حل کنیم طی دو پیمایش در $O(n)$ این کار می‌شود یا نه.

این کار از PrefixSum برای تخمین Mark های قبلی استفاده می‌کنیم.

الگوریتم EREW برای مسئله Array Packing

ورودی: A, M خروجی: O

- $S \leftarrow$ EREW PrefixSum (M) $O(\log n)$
- for i=1 to n: Parallel $O(1)$
  - if M(i) == 1:
    - O(S(i)) ← A(i)
- for i=1 to n: Parallel $O(1)$
  - $\bar{M}(i) = 1 - M(i)$
- $\bar{M}(1) \leftarrow \bar{M}(1) + S(n)$ $O(1)$
- $\bar{S} \leftarrow$ EREW PrefixSum ($\bar{M}$) $O(\log n)$
- for i=1 to n: Parallel $O(1)$
  - if M(i) == 0:
    - $O(\bar{S}(i)) \leftarrow A(i)$
- return O $O(1)$

$T(n) = O(\log n)$

$w(n) = O(n)$ work optimal

$C(n) = O(n \log n)$ not cost optimal

Maximum Sum Subsequence:

	1	2	3	4	5	6	7	8
	-2	4	1	-3	5	-7	2	4
جمع	-2	4	5	2	7	0	2	6
شروع دنباله	1	2	2	2	2	2	7	7

ایده مطرح برای بدست آوردن پاسخ

در حالت حل ترتیبی مسأله

۱۴۰۲/۰۲/۱۶ :

الگوریتم:

	1	2	3	4	5	6	7	8
A	2	-3	4	1	-3	5	-2	1

Postfix Sum →

	5	3	6	2	1	4	-1	1

		i				j		
S	2	-1	3	4	1	6	4	5

$S[j] - S[i] + A[i]$

M	5	5	5	5	5	5	5	5

PostMax Fix

از بین زیر دنباله‌هایی که از i شروع می‌شوند ← بیشترین جمع محاسبه شود.

$$A[i-i] = \underbrace{S[i] - S[i]}_{\text{fix}} + A[i]$$

$$A[i-i+1] = S[i+1] - S[i] + A[i]$$

$$A[i-i+2] = S[i+2] - S[i] + A[i]$$

$$\vdots \qquad \vdots$$

$$A[i-j] = S[j] - S[i] + A[i]$$

باید Max باشد. از آرایه M برای این کار کمک می‌گیریم.

الگوریتم محاسبه جمع زیر دنباله با بیشترین مجموع:

ورودی: دنباله A خروجی: جمع زیر دنباله با بیشترین جمع

- $S \leftarrow$ EREW Prefix Sum (A) $O(\log n)$ n
- $M \leftarrow$ EREW Postfix Max (S) $O(\log n)$ n
- For $i = 1$ to n: Par $O(1)$ n

  $X[i] = M[i] - S[i] + A[i]$

- return EREW Max (X) $O(\log n)$ n

$T(n) = O(\log n)$

$W(n) = O(n)$ work optimal

$C(n) = O(n \log n)$ not cost optimal

جمع زیر دنباله‌ای با بیشترین مجموع که از i شروع شده

با استفاده از Reduction و اعمال Max روی جمع

Interval Broadcast (انتشار بازه‌ای)

الگوریتم سه انتشار بازه‌ای:

ورودی: A و M خروجی: O

- به‌ازای i از 1 تا n: (موازی) $O(1)$
  - اگر $M[i]$ برابر 1:
    - $I[i] \leftarrow i$
  - در غیر این صورت:
    - $I[i] \leftarrow 0$
- $\text{EREW PostfixMax}(I) \rightarrow K$
- به‌ازای i از 1 تا n:
  - $O[i] = A[K[i]]$ ← این قسمت از الگوریتم هرزمان دارد. به‌همین دلیل نمی‌تواند EREW باشد. باید الگوریتم را تغییر دهیم.

	1	2	3	4	5	6	7	8
A	2	3	4	1	8	5	2	7
M	1	0	0	1	1	0	0	1
I	1	0	0	4	5	0	0	8
K	1	1	1	4	5	5	5	8
Interval Broadcast (O)	2	2	2	1	8	8	8	7

I جدید: | 1,2 | 0,3 | 0,4 | 4,1 | 5,8 | 0,5 | 0,2 | 8,7 |

عملوندهای جدید: $(i, A[i])$

تعریف یک عمل جدید:

$(i,x) \otimes (j,y)$ → از i → (i,x) ; از j → (j,y)

اگر $M[i]$ برابر یک باشد، مقدار اندیس را می‌گیرد، در غیر این صورت صفر است.

مثال: $k[4] = \big((1,2) \otimes (0,3)\big) \otimes \big((0,4) \otimes (4,1)\big)$

$(1,2) \otimes (0,3) = (1,2)$

$(0,4) \otimes (4,1) = (4,1)$

$(1,2) \otimes (4,1) = (4,1)$

K جدید: | 1,2 | 1,2 | 1,2 | 4,1 | 5,8 | 5,8 | 5,8 | 8,7 |

الگوریتم اصلاح شده:

- به‌ازای i از 1 تا n: (موازی) $O(1)$
  - اگر $M[i]$ برابر 1 باشد:
    - $(i, A[i]) \rightarrow I[i]$
  - در غیر این صورت:
    - $(0, A[i]) \rightarrow I[i]$
- $\text{EREW PrefixNewMax}(I) \rightarrow K$ $O(\log n)$

عمل تعریف شده روی نوع مرتب‌های I

- به ازای i از ۱ تا n : سوازی $O(1)$

- $O[i]$ ⟵ عنصر دوم $K[i]$ ⟵ { عنصر اول : اندیس بیشه / عنصر دوم : مقدار A با اندیس بیشه

$T(n) = O(\log^n)$

$w(n) = O(n)$

$C(n) = O(n\log^n)$

$S(n) = O(\frac{n}{\log^n})$

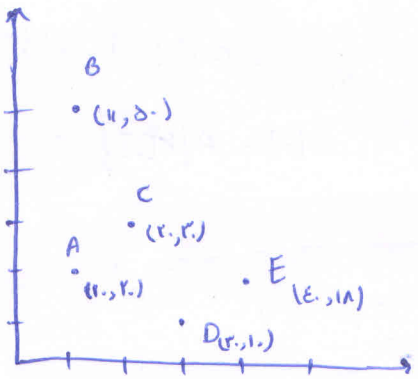
۱۴۰۲/۰۲/۱۸ :

Point Domination Query

دوروی : تعدادی نقطه در صفحه

خروجی : نقاط غلبه‌پذیر

P dominates Q if $\begin{cases} P_x > Q_x \\ P_y > Q_y \end{cases}$



۱. $O(\log^n)$ نقاط دوروی را با توجه به مؤلفه x مرتب کن* ⟵ A

۲. $O(1.\log^n)$ EREWPostfixMax(A) ⟵ S (با در نظر گرفتن مؤلفه y عناصر آرایه A)

۳. $O(1)$ به ازای i از ۱ تا n : Par

- اگر S[i] > A[i]y : مؤلفه y نقطه A[i]

$0 \longrightarrow O[i]$

- در غیر این صورت :

$1 \longrightarrow O[i]$

مرتب سازی با توجه به مؤلفه X — A → | A | B | C | D | E |

بیشه پسوندی مؤلفه y نقاط آرایه A — S | 50 | 50 | 30 | 18 | 18 |

خروجی : معکوس نردبان‌ها — | 0 | 1 | 1 | 0 | 1 |

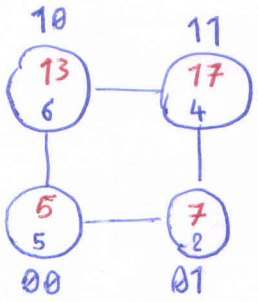
$T(n) = O(\log^n)$

$w(n) = O(n\log^n)$

$C(n) = O(n\log^n)$

* الگوریتمی برای مرتب سازی از $O(n\log^n)$ داریم که $w(n) = n\log^n$

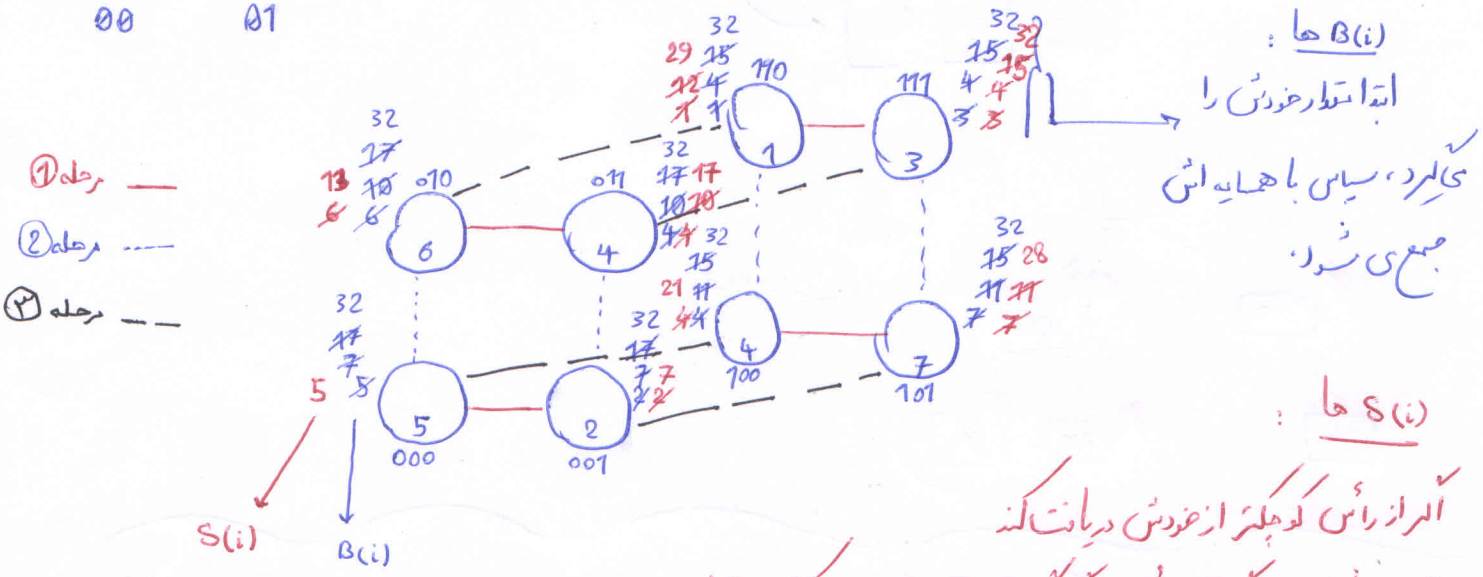
Prefix Sum in Hyper cube



$A(i) \longrightarrow$ عدد رأس i-ام

$B(i) \longrightarrow$

$S(i) \longrightarrow$ جمع پیشوندی i-ام



B(i) ها :

ابتدا مقدار خودش را

می‌گیرد، سپس با همسایه‌اش

جمع می‌شود.

S(i) ها :

اگر از رأس کوچکتر از خودش دریافت کند

جمع می‌شود، اما اگر از رأس بزرگتر بگیرد، ignore می‌کند و مقدار

قبلی را نگه می‌دارد. مقدار اولیه S(i) برابر A(i) است.

نکته : در هر مرحله برای محاسبه S(i)، چنانچه جمع انجام شود باید با آخرین مقدار B(i) جمع انجام شود، نه مقدار S(i) قبلی

محاسبه Prefix Sum در Hypercube با بُعدی k :

توسط پردازنده i-ام اجرا می‌شود :

- $S(i) \leftarrow A(i)$
- $B(i) \leftarrow A(i)$
- for $j=0$ to $k-1$
  - friend $\leftarrow i$ xor 2^j
  - Send (friend, B(i)) $\longrightarrow$ Non-blocking
  - his $\leftarrow$ recv (friend)
  - if friend $< i$:

    $S(i) = his + S(i)$

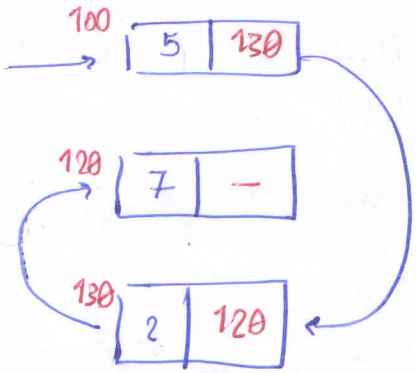
    $B(i) = his + B(i)$
  - else :

    $B(i) = B(i) + his$

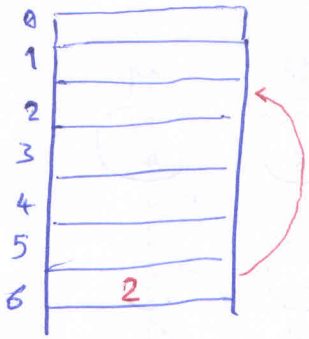
$T(n) = O(\log n)$

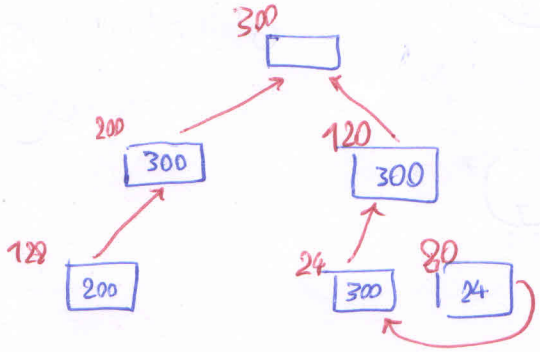
نکته : عملیات عملی شرکت پذیر باشد اما قابل جابجایی نباشد.

Pointer Jumping (پرش اشاره‌گر)



Linked List نمونه





نمونه دوم:

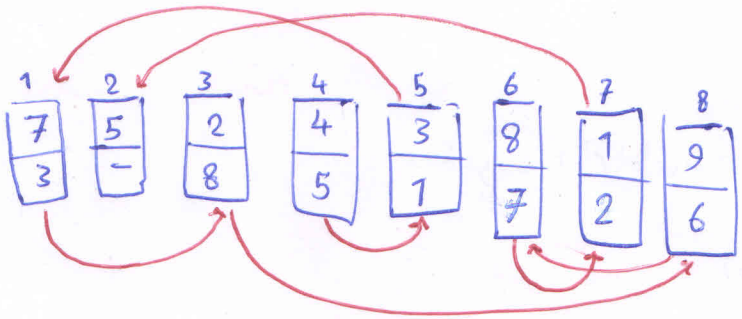
بحث اصلی:

	1	2	3	4	5	6	7	8
A	7	5	2	4	3	8	1	9

N	3	-	8	5	1	7	2	6

شروع از اندیس 4

پایان در اندیس 2



* یک مسأله: فلان عنصر در لیست چقدر با عنصر آخر فاصله دارد؟

5	0	4	7	6	2	1	3

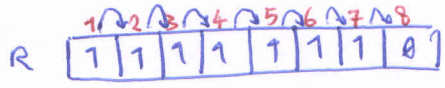
مثلاً برای مسأله بالا، مراحل داشت:

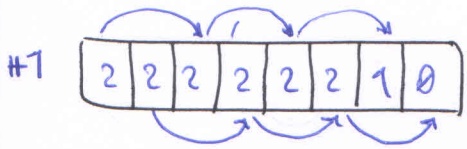
$T^*(n) = O(n)$

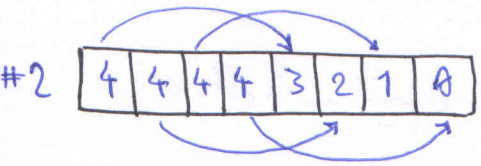
List Ranking

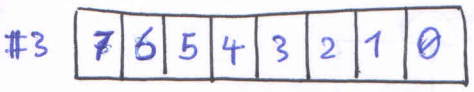
ورودی: لیست پیوندی (Linked list)

خروجی: فاصله هر عنصر لیست با عنصر آخر لیست









$O(1)$
- for $i=1$ to n Par
  - if $N(i) \neq -$:
    - $R(i) = 1$
  - else :
    - $R(i) = 0$

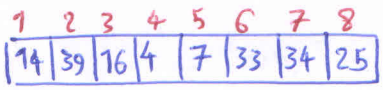
$O(\log n)$
- for $i=1$ to n: Par
  - while $N(i) \neq -$:
    - $R(i) = R(i) + R(N(i))$
    - $N(i) = N(N(i))$

$T(n) = O(\log n)$

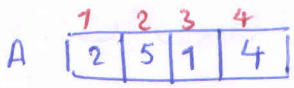
$W(n) = O(n \log n)$

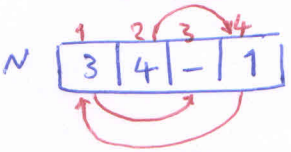
$C(n) = O(n \log n)$

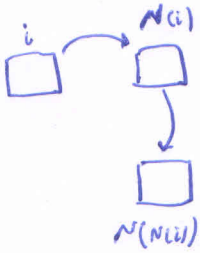
Prefix Sum (در حالت استفاده از لیست پیوندی)



با اضافه آرایه A در شکل قبل:







$O(1)$
- for $i=1$ to n: Par
  - $S(i) \leftarrow A(i)$

$O(\log n)$
- for $i=1$ to n: Par
  - while $N(i) \neq -$
    - $S(N(i)) = S(i) + S(N(i))$
    - $N(i) = N(N(i))$

$T(n) = O(\log n)$

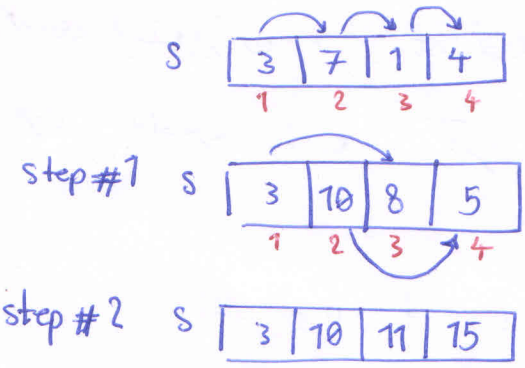
$W(n) = O(n \log n)$

$C(n) = O(n \log n)$

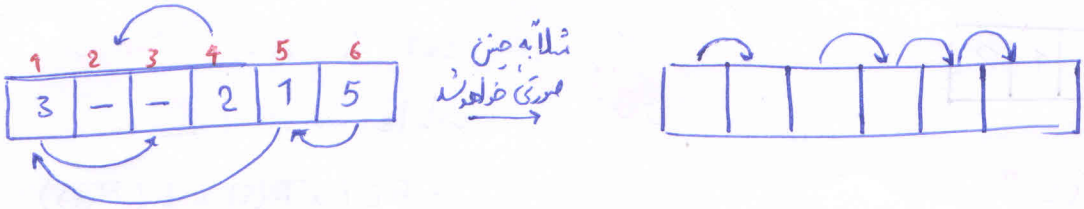
۱۴۰۲/۰۲/۲۵

ادامه مبحث قبل:

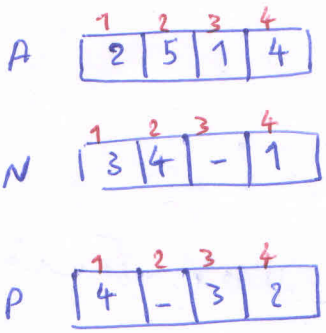
مثال:



نکته: اگر حالتی داشته باشیم که بیش از یک بار هم به عنوان دوری داشته باشیم، الگوریتم درست کار می‌کند.



Reversing a Linked List

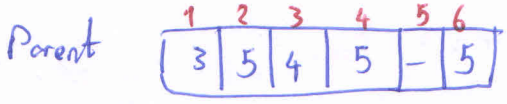


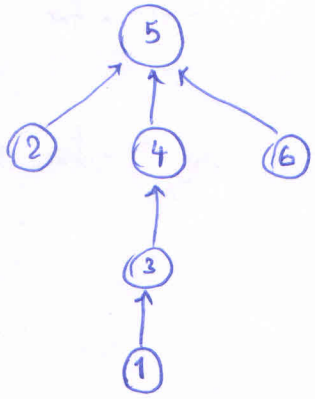

```
O(1)  • for i=1 to n:    Par
      • P(i) ← -
O(1)  • for i=1 to n:    Par
      • P(N(i)) ← i
```

$T(n) = O(1)$

$w(n) = O(n)$

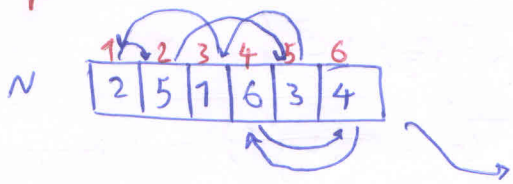
نمایش درخت با استفاده از این تکنیک: (درخت برعکس در نظر گرفته می‌شود. یعنی از فرزندان به ریشه می‌رسیم.)





Loops

حالت دیگر: حلقه (Loop) در این نمونه



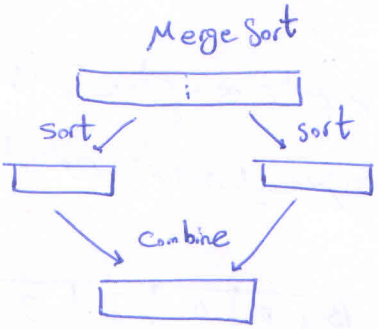
از دو حلقه تشکیل شده است.

نکته: اگر بخواهیم به ازای تعداد لیت دوری، ماکسیمم تعادیر هرلیت را در همان لیت محاسبه کنیم، ابتدا از Prefix Max استفاده می‌کنیم. سپس اگر بخواهیم تعادیر Max در تمام اعضای هرلیت باشد از PostfixMax استفاده می‌کنیم.

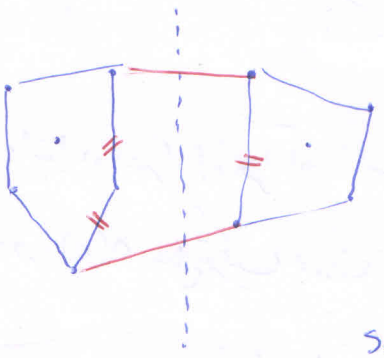
Divide & Conquer

1. divide
2. Solve recursively
3. combine

به عنوان مثال در مسأله MergeSort



- Convex Hull



0. Sort
1. Divide $O(1)$
2. Recursive $2 \times T(\frac{n}{2})$
3. Combine $O(n)$

Sequential $\longrightarrow T(n) = O(n \log n)$

Parallel Divide & conquer Algorithm for convex Hull #1

دوری: $P \leftarrow$ نقاط دوری که برحسب مؤلفه x مرتب شده‌اند.

- $A, B \leftarrow$ تقسیم آرایه P به دو دسته تقریباً مساوی با توجه به مؤلفه x
- $C_A, C_B \leftarrow$ پوش محدب A و B را به صورت موازی حساب کن
- $C \leftarrow$ ترکیب C_A, C_B با الگوریتم ترتیبی $O(n)$

$T(n) \leq T(\frac{n}{2}) + Cn$

$\rightarrow T(n) = O(n)$

$W(n) = O(n \log n)$

$C(n) = O(n^2)$

Parallel Convex Hull Algorithm #2

- مرحله i ام شبه نزل
- ~ ~ i+1 ~
- $C \leftarrow O(\log^2 n)$ الگوریتم C_A و C_B ترکیب

$$T(n) \leq T(\frac{n}{2}) + C\log^2 n$$

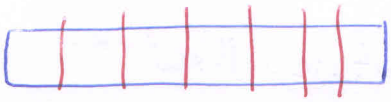
$$\rightarrow T(n) = O(\log^2 n)$$

$$\rightarrow C(n) = \Theta(n \log^2 n)$$

$$W(n) = ?$$

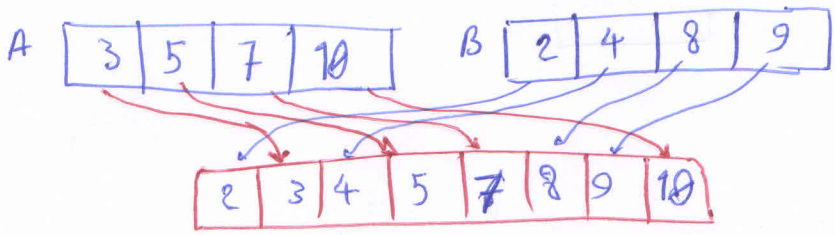
۱۴۰۲/۰۲/۳۰ :

Partitioning



فرض : دو آرایه مرتب داریم و می خواهیم ترکیب آن دو را به صورت مرتب برای انجام دهیم که در نهایت نیز به یک آرایه مرتب برسیم.

input : sorted Array of size N
output: Merged Array of size 2N



ایده : در الگوریتم ترتیبی، ابتدا دو عنصر اول هر آرایه مقایسه می شوند، عنصر کوچکتر به آرایه جدیدی اورد و Pointer آن را اضافه می شود. در نهایت در $O(n)$ عمل ترکیب صورت می گیرد.

Rank $(x, \vec{A})$: تعداد عناصر کوچکتر از x در آرایه A مرتب

در مثال بالا : Rank $(5, A) = 1$
Rank $(7, B) = 2$

* الگوریتم ترتیبی : یکبار پیمایش در آرایه و مقایسه در $O(n)$

الگوریتم سوازی :

1. $X \leftarrow$ EREW Broadcast (x) $O(\log n)$
2. for $i=1$ to n : par $O(1)$
   if $x[i] > A[i]$:
   $C[i] = 1$
   else :
   $C[i] = 0$
3. return EREW Reduction (sum) (C) $O(\log n)$

$T(n) = O(\log n)$
$W(n) = O(n)$ work optimal

مثال :

A | 7 | 1 | 5 | 3 |

$x = 6$

① X | 6 | 6 | 6 | 6 |

② C | 0 | 1 | 1 | 1 |

③ C | 3 |

Rank (x, A)
↳ آرایه مرتب از n عنصر

* الگوریتم ترتیبی : جستجوی دودویی $O(\log n)$ (ی خواهیم سوری که از x کوچکتر / بزرگتر / مساوی است را پیدا کنیم)

* الگوریتم سوازی : EREW : $P(n) = O(1)$ یک پردازنده ، $T(n) = O(\log n)$ ← خوبیت چون الگوریتم ترتیبی هم $O(\log n)$ است .

* الگوریتم CREW :

. $b \leftarrow 0$
. for $i = 0$ to n : par
  . if $A[i] < x$:
    . if $i == n$ or $A[i+1] \geq x$:
      $b \leftarrow i$
. return b

$T(n) = O(1)$, $P(n) = n$
$W(n) = O(n)$

بازگشت به مسأله اصلی : ی خواهیم جایگاه هر یک از عناصر آرایه های A, B را در آرایه نهایی C بدانیم . اگر مقدار $Rank(x, C)$ را بدانیم به صورت سوازی عناصر آرایه های دودوی را در خانه مناسب در آرایه فرودی قرار می دهیم .

$$Rank(A[i], C) = \overbrace{Rank(A[i], A)}^{i-1} + Rank(A[i], B)$$

$$Rank(B[i], C) = Rank(B[i], A) + \overbrace{Rank(B[i], B)}^{i-1}$$

الگوریتم ترکیب دو آرایه مرتب:

ورودی: آرایه‌های A و B با اندازه n

خروجی: آرایه C با اندازه 2n

$O(\log^n)$ $O(1)$. for i=1 to n: Par

$C[i + Rank(A[i], B)] \leftarrow A[i]$

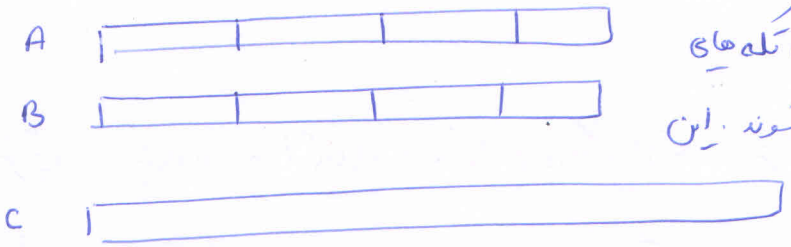
$O(\log^n)$ $O(1)$. for i=1 to n: Par

$C[i + Rank(B[i], A)] \leftarrow B[i]$

if CREW Rank : $T(n) = O(1)$, $P(n) = O(n^2)$, $W(n) = O(n^2)$

if EREW Rank : $T(n) = O(\log^n)$, $P(n) = O(n)$, $W(n) = O(n \log^n)$

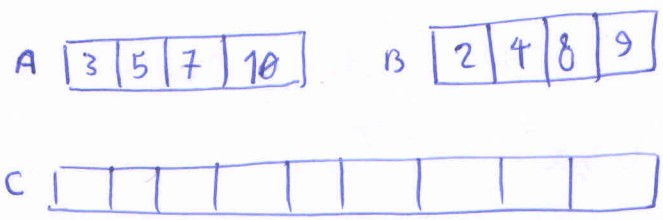
Partitioning یک :



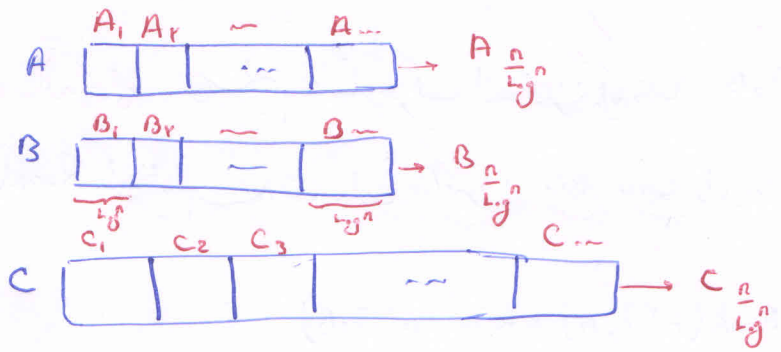
آرایه A و B ابتدا تکه‌تکه می‌شوند، سپس تکه‌های A با تکه‌هایی از B باید Merge شوند. این فرآیند الهام بیای کند.

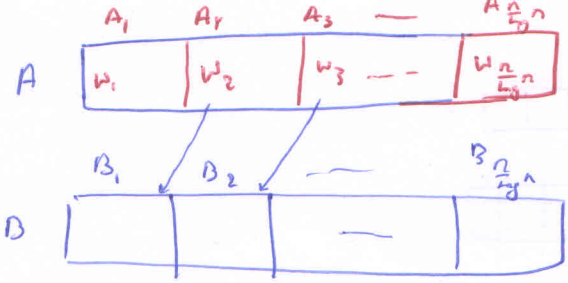
۱۴.۲ ،۰۳ ،۰۱ :

Partitioning (cont.)



باتوجه به آخرین سبت جمله قبل:





مثال :

A_1 w_2

2	3	8	10	15

4	5	7	16	20	22

B_1

الگوریتم ترکیب دو آرایه مرتب :

Work

$O(1)$ - آرایه A را به $\frac{n}{\log n}$ تکه با اندازه $\log n$ تقسیم می‌کنیم. A_i بخش i-ام

w_i عنصر اول A_i باشد.

$O(n)$ - به ازای $i=2$ تا $\frac{n}{\log n}$: Par

- با استفاده از $Rank(w_i, B)$ عنصر آخر B_{i-1} را مشخص کن ← الگوریتم جستجوی دودویی $O(\log n)$

$O(n)$ - به ازای i از 1 تا $\frac{n}{\log n}$: Par

- تکه های A_i و B_i را ترکیب کن و در C_i بریز ← اولین عنصر : عناصر قبل از B_i + عناصر قبل از A_i + 1

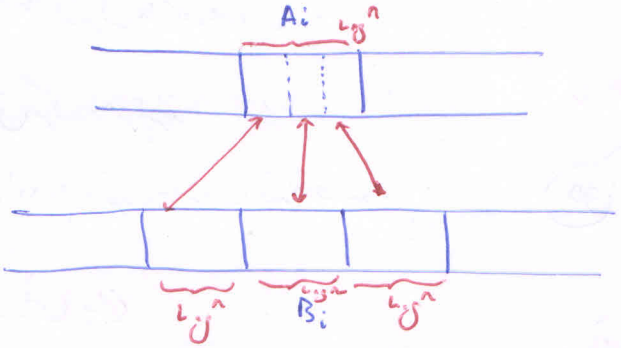
↓

از الگوریتم ترتیبی merge استفاده می‌کنیم

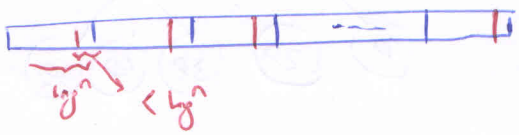
به ازای بدترین حالت چینش عناصر آرایه ورودی

$\Rightarrow T(n) = O(n)$, $W(n) = O(n)$

در این الگوریتم کارهایی که انجام شد $O(n)$ زیاد است. مشکل از قسمت ترکیب تکه‌ها است. از طرفی آرایه B را نمی‌توانیم مثل آرایه A با اندازه ثابت در نظر بگیریم. به همین دلیل برای بدترین حالت ورودی عناصر، آرایه A را به ازای B_i های که اندازه‌شان بیشتر از اندازه مورد نظر می‌شود مجدداً پارتیشن بندی می‌کنیم.



در بدترین حالت $2 \times \frac{n}{\log n}$ پارتیشن در B ایجاد می‌شود.



$\frac{n}{\log n} + \frac{n}{\log n}$

تکه های با طول حداکثر $\log n$ ← → تکه های با طول $\log n$

① A B

② A B

• الگوریتم دوم برای محاسبه مسأله ، ②

۱- ...

۲- ...

۳- به ازای $i=1$ تا $\frac{n}{\log^n}$ Par :

- اندازه B_i بزرگتر از $\log^n$ باشد ،
- B_i را به تکه های حداکثر $\log^n$ تایی تقسیم کن
- از روی تکه های حاصل A_i را به تعدادی تکه تقسیم کن شبیه گام دوم ـ

۴- به ازای $i=1$ تا تعداد تکه ها Par

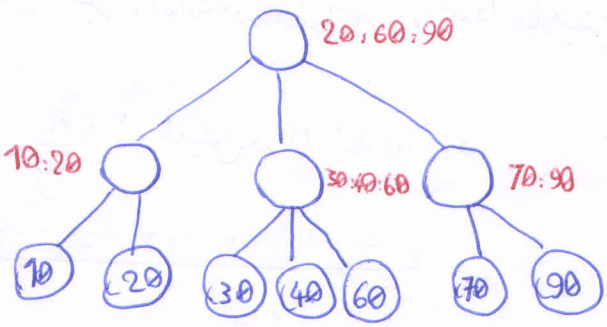
- ترکیب A_i و B_i $\leftarrow$ C_i

work	Time
$O(1)$	$O(1)$
$O(n)$	$O(\log^n)$
$O(n)$	$O(\log \log^n)$
$O(n)$	$O(\log^n)$

$O(n) \leftarrow O\left(\frac{n \cdot \log \log^n}{\log^n}\right)$ $O(1)$ $O(\log \log^n)$

۱۴۰۲/۰۳/۰۶ :

2-3 Tree



• هر رأس غیر برگ ، ۲ یا ۳ فرزند دارد .

• داده ها در برگ ها ذخیره می شوند .

• عمق برگ ها برابرند .

• داده ها در برگ ها مرتب شده هستند .

در هر رأس میانی :

بزرگترین عدد زیر درخت سمت چپ $L(v)$

~ ~ ~ ~ راست $R(v)$

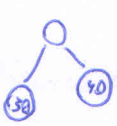
~ ~ ~ ~ میانی $M(v)$

مسأله ۱ : جستجوی عنصر در درخت $T(n) = O(\log^n)$

ارتفاع درخت $\leq \log_2^n$ $<$ $\log_3^n$

مسأله ۲ : درج عناصر در درخت $T(n) = O(\log^2 n)$

① ابتدا می خواهیم عدد ۴۰ را در درخت زیر درج کنیم



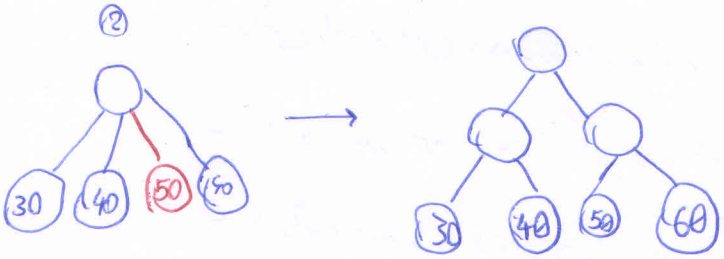
② سپس می خواهیم عدد ۵۰ را اضافه کنیم.

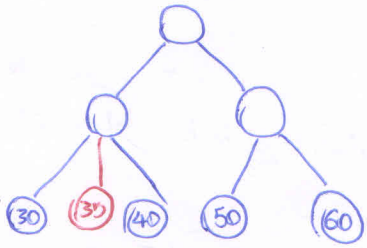
③ سپس می خواهیم عدد ۳۵ را اضافه کنیم.

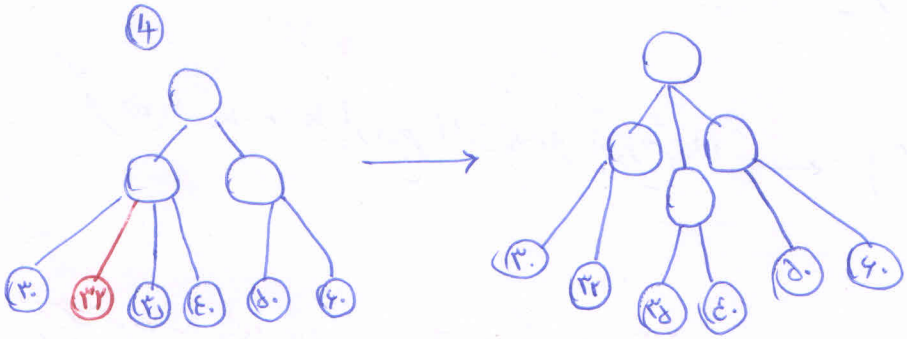
④ " " " ۴۲ را اضافه کنیم

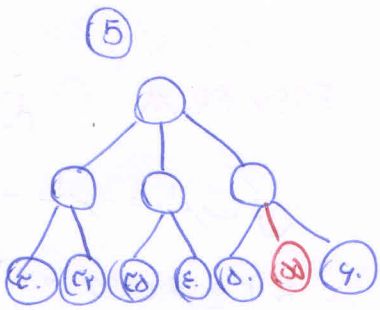
⑤ " " " ۵۵ را اضافه کنیم

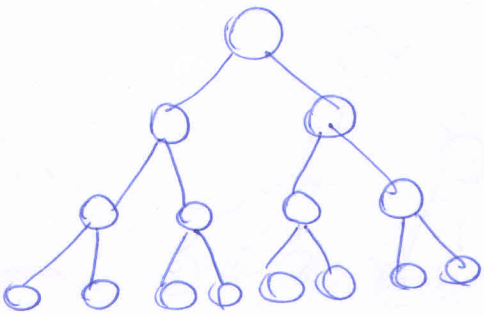
⑥ " " " ۵۶ را اضافه می کنیم.

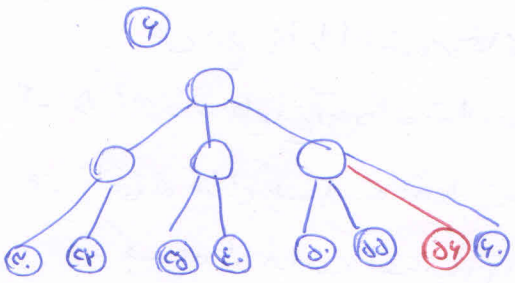


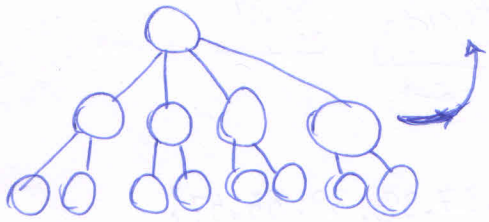












الگوریتم درج در درخت ۲-۳ : مقدار x

۰ ← جستجو برای رأسی که برگ با متدار x باید به آن اضافه شود.

۰ تا وقتی که تعداد فرزندان x بیشتر از ۳ باشد :

- رأس را به دو رأس با دو فرزند تقسیم کن

- P ← پدر x ← اگر x ریشه باشد ، رأس P را به عنوان پسر x اضافه کن

- دو رأس جدید را جایگزین x به عنوان فرزندان P قرار بده

- P ← x

نکته : اگر k به عنوان تعداد اعداد ورودی باشد ، $k = \Omega(n)$ (یعنی رشد بالایی داشته باشد)

- مرتب سازی و ساخت دوباره درخت $O((n+k)\log(n+k))$

اگر $k=O(n)$ ، k بار درج با استفاده از الگوریتم بالا ← $O(k\log n)$

درج مجموعه k تایی به صورت موازی در درخت ۲-۳ :

الگوریتم ① EREW PRAM

۱- عناصر B را مرتب کن $O(\log k)$

۲- به ازای i از ۱ تا k موازی $O(\log k)$

- محل درج b_i را با جستجوی ترتیبی بیاب

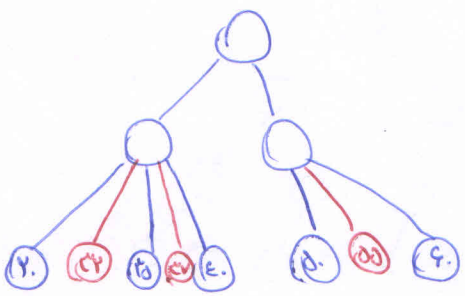
۳- B را با توجه به محل درج آن‌ها به تعداد دسته تقسیم کن $O(1)$

۴- تا وقتی که حداقل یک دسته عدد داشته باشد : موازی

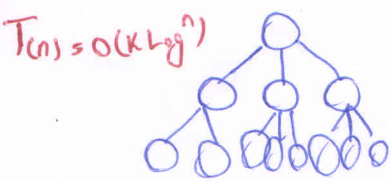
۴-۱- یک عدد از هر دسته انتخاب کن و به رأس پدر آن اضافه کن

۴-۲- به ازای d از عمق درخت تا ریشه تکرار کن : موازی

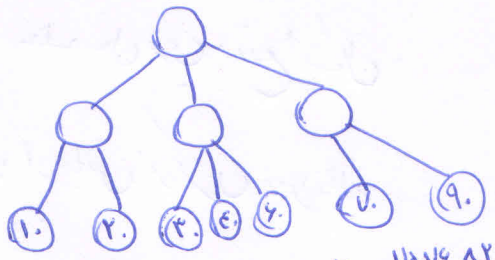
- رأس هایی که بیشتر از ۳ فرزند دارند را به دو رأس تقسیم کن.



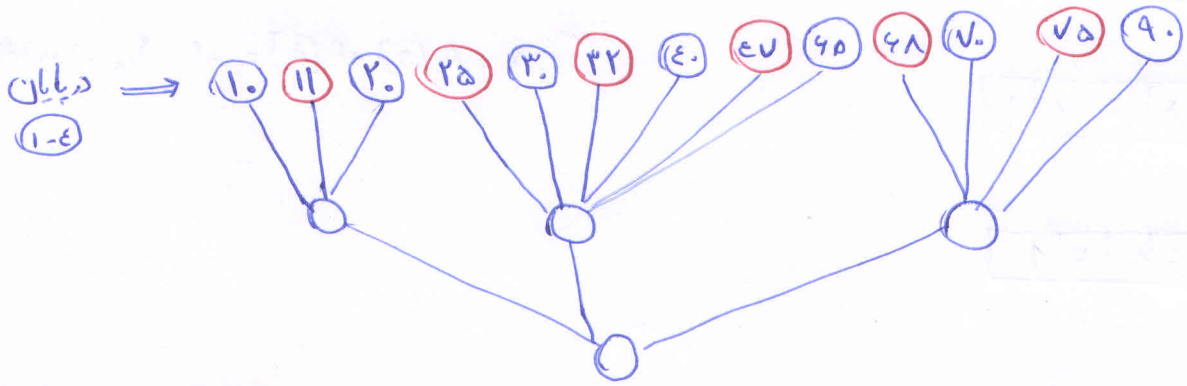
$B = \{32, 37, 38, 39, 55, 57\}$

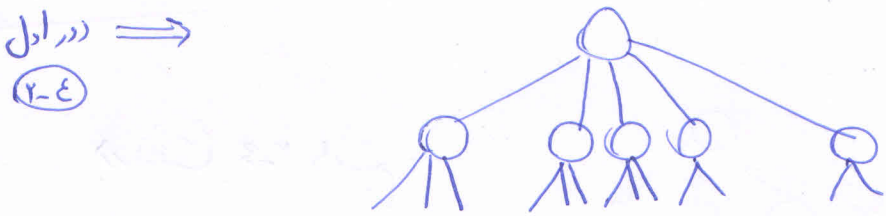


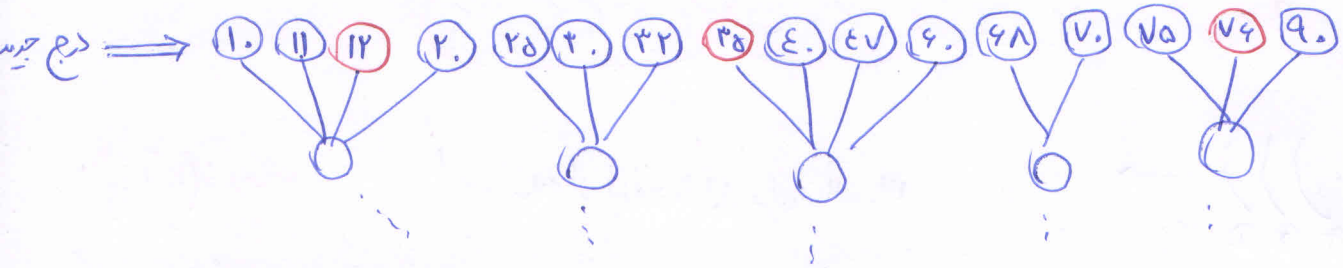
مثالی برای الگوریتم قبل :

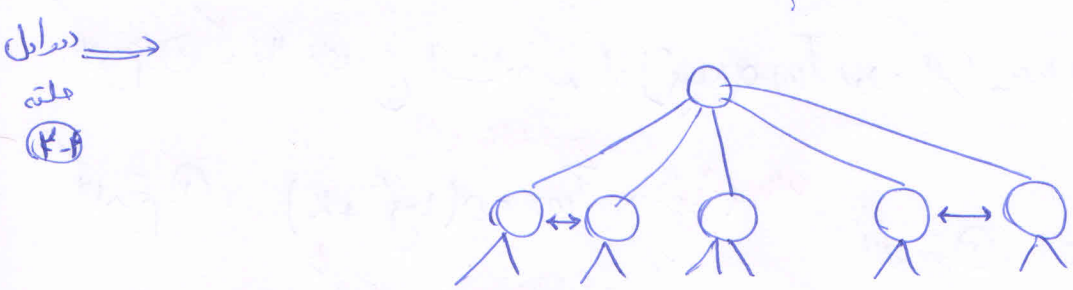


B = ۱۱ ۱۲ ۲۵ ۳۲ ۳۵ ۴۷ ۴۸ ۷۵ ۷۶ ۸۲









۱۴۰۲/۰۳/۰۸ : جلسه حل تمرین و رفع اشکال

نمونه سوال از پایان بزرگترین زیردنباله :

A	۲	-۱	۱	-۳	۲	-۱	۴	۱

S (Prefix Sum)	۲	۱	۲	-۱	۱	۰	۴	5

Postfix Max	۵	۵	۵	۵	۵	۵	۵	۵

۵	۳	4	3	6	4	5	1

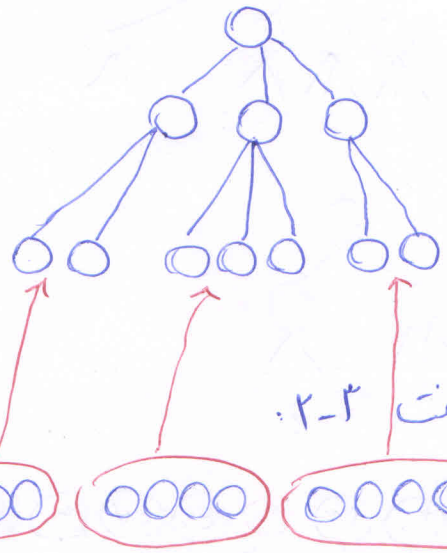
مثلاً : $Sum_{3-3} = A[3] = 1$ $= S_{(3)} - S_{(2)} + A[3]$

$Sum_{3-4} = A[3] + A[4] = -1 = S_{(4)} - S_{(3)} + A[3]$

۱۴۰۲/۰۳/۱۳ :

Pipelining

در درخت 3-۲ داشتیم :



B :

درج k عنصر به صورت موازی در درخت ۳-۲ :

استفاده از ایده خط لوله برای مسأله بالا

$w(n) = (k^2 \log n)$

الگوریتم ① که قبلاً برای درج استفاده شد از $T(n) = O(k \log n)$ بود. اگر از خط لوله استفاده کنیم :

الگوریتم ② : $T(n) = O(\log n + k)$ ← $w(n) = (k^2 \log n)$

پایان درج اول از هر دسته

درج‌های بعدی پس از درج اول

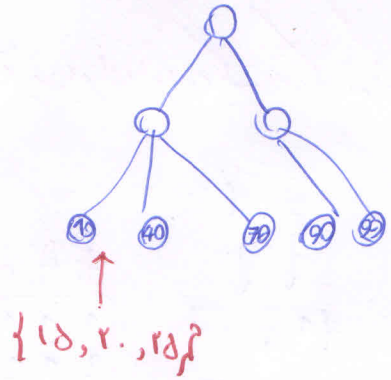
الگوریتم ③ : انتخاب عنصر میانی از هر لحظه

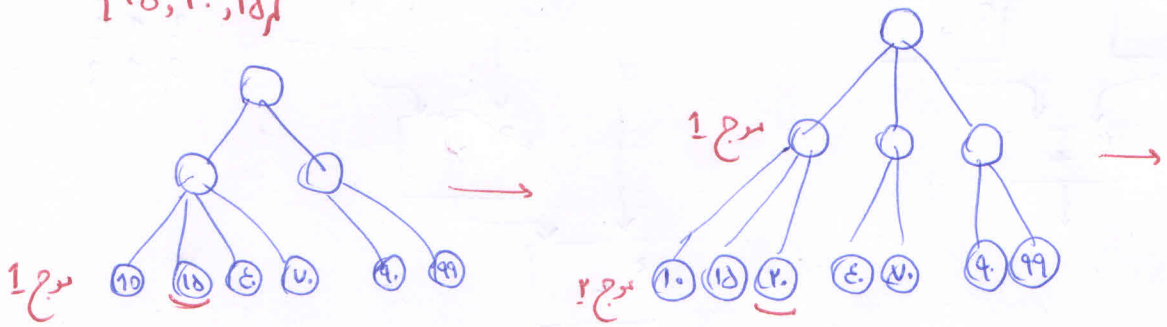
تعداد درج ها $(\log k)$

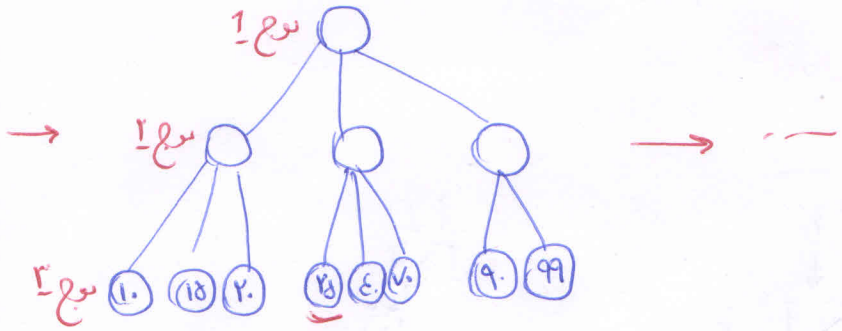
$T(n) = O(\log n + \log k)$

مثال از الگوریتم ④ :

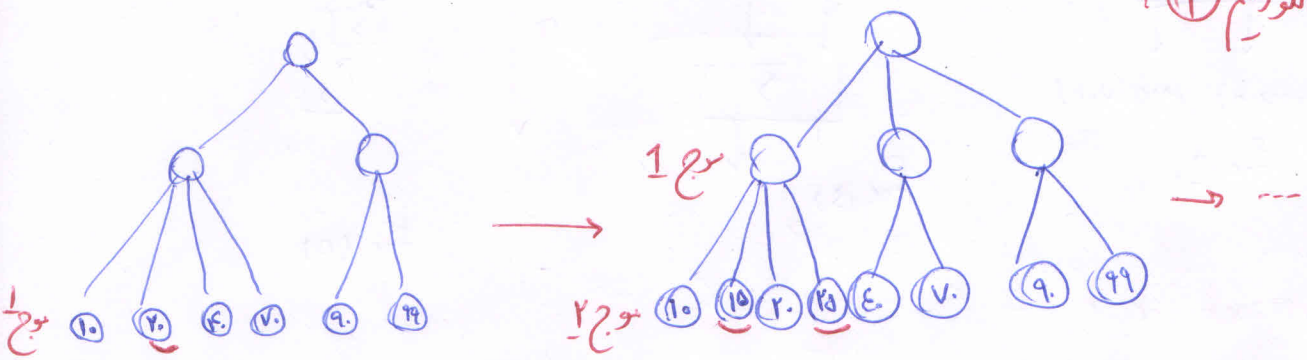
فرض کنیم اعداد {۱۵, ۲۰, ۲۵} را می‌خواهیم درج کنیم.







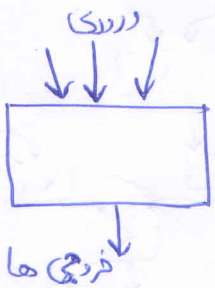
مثال بالابرای الگوریتم ③ :

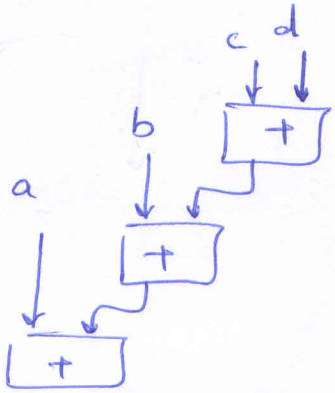


Combinatorial circuits

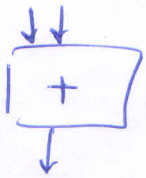
- Node
- Processing element
- Processor

گره‌ی





مثلاً :



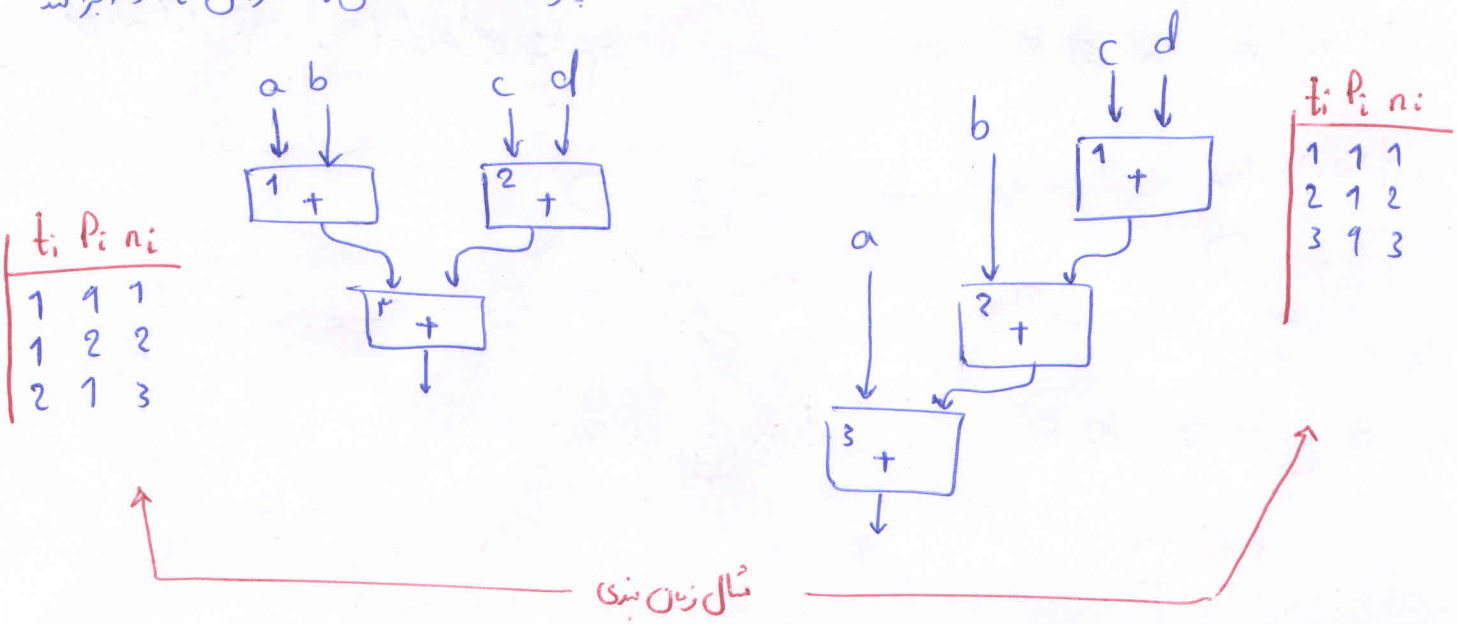
مسیر بحرانی : بلندترین مسیر از ورودی‌ها به خروجی (critical path)

پیچیدگی شبکه : تعداد رأس‌ها (Network complexity)

زمان بندی برای اجرای یک شبکه ترکیبی:

P_i t_i n_i

پردازنده P_i در زمان t_i، رأس n_i را اجرا کند.

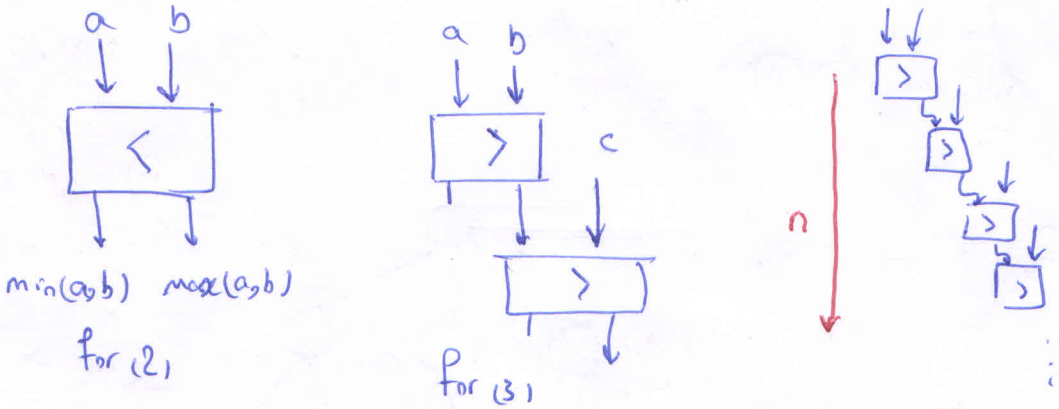


t_i	P_i	n_i
1	1	1
1	2	2
2	1	3

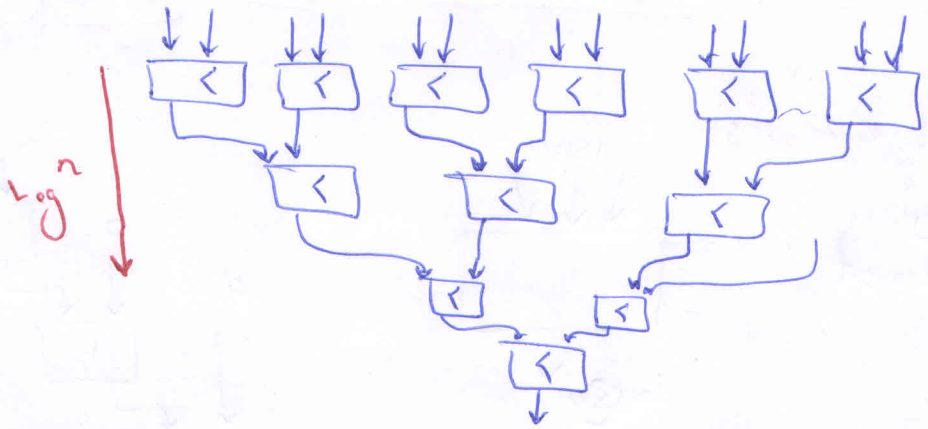
t_i	P_i	n_i
1	1	1
2	1	2
3	1	3

مثال زمان بندی

Sorting Network



better approach for n:



دنباله مونوتونیک (یکنواخت - monotonic)

A: 2,3,3,4,5 B: 7,3,2,1

C: 7,3,2,1,2,3,3,4,5

D: 4,5,7,3,2,1,2,3,3

دنباله bitonic: از ترکیب دو دنباله یکنواخت حاصل می‌شود.

نکته: عکس‌العمل عناصر نیست داده شوند.

Bitonic split عمل

$$S = (a_1, a_2, \dots, a_n, a_{n+1}, \dots, a_{2n})$$

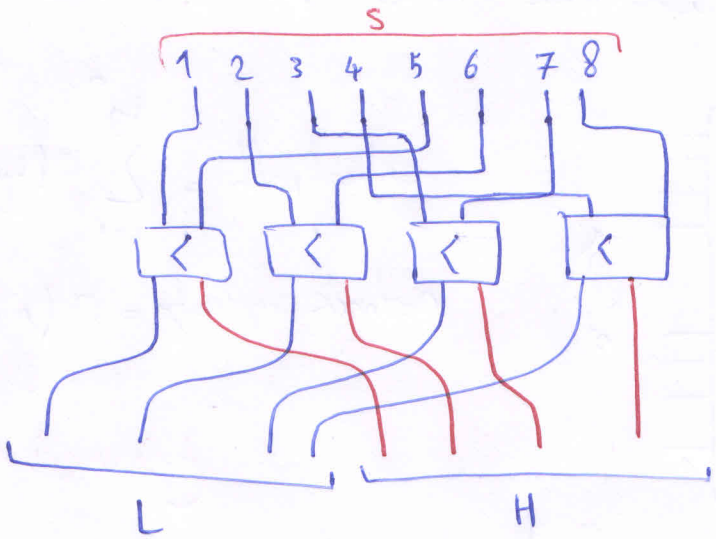
↓ یک دنباله Bitonic به طول $2n$

خروجی‌ها:

$$L = (\min(a_1, a_{n+1}), \min(a_2, a_{n+2}), \dots, \min(a_n, a_{2n}))$$

$$H = (\max(a_1, a_{n+1}), \max(a_2, a_{n+2}), \dots, \max(a_n, a_{2n}))$$

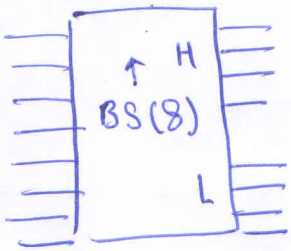
مثال:



ویژگی‌های Bitonic Split:

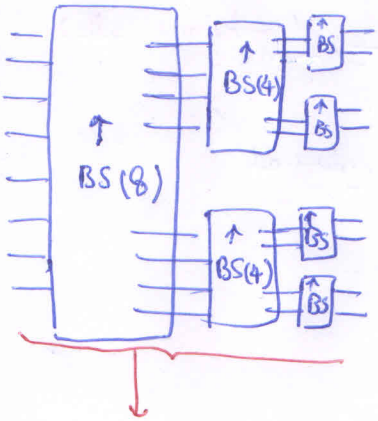
- هر عنصر L از هر عنصر H بزرگتر است.
- اگر S، Bitonic باشد، L و H هم Bitonic هستند.

استفاده از بلوک به جای شکل با جزئیات:



هدف اول: مرتب کردن یک دنباله Bitonic

"باید به صورت بازگشتی از چند BS استفاده کنیم:

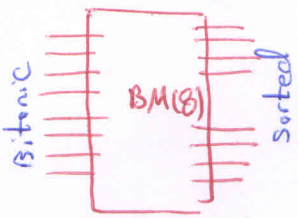


$T_{Prom}(n) = O(\log^2 n)$ = مسیر بحرانی

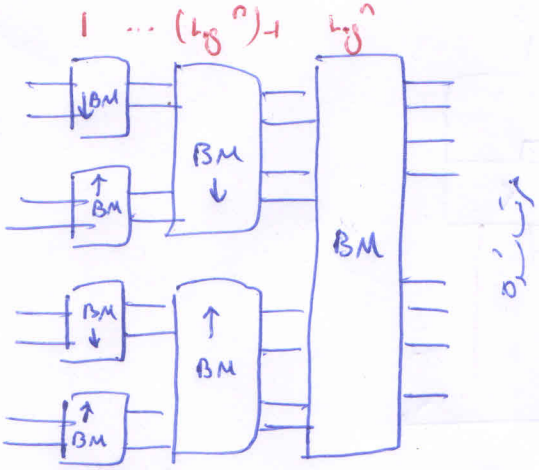
پیچیدگی: $O(\frac{n}{2}\log^2 n) = O(n\log^2 n)$

تعداد node در هر لایه ↓ تعداد لایه‌ها

Bitonic Merge



هدف دوم: مرتب کردن یک دنباله عام از عناصر (ممکن است Bitonic نباشد.)



$$\text{critical path} = \sum_{i=1}^{\log n} O(\log 2^i) = O(\log^2 n)$$

$$\text{network complexity} = O(n \log^2 n)$$

تعداد راس‌های هر سطح

تعداد سطح‌های Bitonic Split