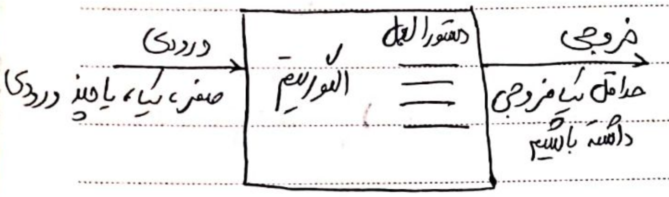
فصل اول: مقدمه‌ای بر تحلیل الگوریتم‌ها

الگوریتم: الخوارزمی / روش حل مسئله

تعریف دقیق الگوریتم در علم کامپیوتر: هر دستورالعملی که مراحل مختلف انجام کاری را به زبان دقیق و با جزئیات کامل بیان کند به گونه‌ای که ترتیب مراحل و شروط خاتمه عملیات در آن کاملاً مشخص باشد.



* الگوریتم مجموعه‌ای است متناهی از مراحل که هر مرحله ممکن است مستلزم یک یا چندین عملیات باشد.

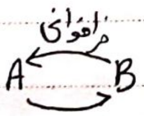
ویژگی‌های دستورالعمل‌های الگوریتم: الف) باید غیرمبهم باشد ب) باید قابل اجرا باشد پ) شروط خاتمه داشته باشد

مثال: الف) (۶×۵) or (۳)+۲) ✗ ب) جمع کل اعداد روی مجموعه اعداد حقیقی IR + IR ✗ پ) محاسبه تمام اعداد اول ✗

* الگوریتم حتماً خاتمه‌پذیر است ولی فرایند می‌تواند خاتمه‌پذیر نباشد.

تقسیم‌بندی الگوریتم‌ها: بازگشتی ـ غیربازگشتی . الف) بازگشتی = ۱ـ مستقیم ۲ـ غیرمستقیم

بازگشتی مستقیم = این الگوریتم‌ها درداخل خود خودشان را فراخوانی می‌کنند (معمولاً با آرگومان کوچک‌تر) ۵! ← ۴! ← ۳!



بازگشتی غیرمستقیم = هرگاه دو الگوریتم درداخل خودشان دیگری را فراخوانی کنند بازگشتی غیرمستقیم هستند.

زمینه‌های مطالعه الگوریتم‌ها

الف) طراحی الگوریتم: روش‌های مختلفی برای طراحی الگوریتم‌ها وجود دارد: استقرای ریاضی - تقسیم و غلبه - حریصانه - برنامه نویسی پویا - پس‌رد - انشعاب و تحدید

ب) معتبرسازی: اثبات درستی الگوریتم. یک الگوریتم زمانی درست است که به ازای هر ورودی مناسب خروجی درست تولید کند.

پ) تحلیل الگوریتم (تحلیل مقدم یا ارزیابی کارایی): تحلیل یک الگوریتم به مرآنیزی اطلاق می‌شود که تعیین می‌کند الگوریتم به چه مدت زمان از CPU برای انجام محاسبات و عملیات (پیچیدگی زمانی) و به چه مقدار حافظه (پیچیدگی مکانی) برای ذخیره سازی داده‌ها و برنامه نیاز دارد.

ت) پیاده سازی: بعد از انجام ۳ مرحله‌ی قبل می‌توان الگوریتم را با یک زبان برنامه‌نویسی پیاده سازی کرد مثل C, C++, C#, Java

ث) تست برنامه: (تحلیل مؤخر)

۱- اشکال زدایی (Debug): اجرای برنامه روی مجموعه داده‌های ورودی برای بررسی و یافتن خطاها و برطرف کردن آنها

۲- پروفیلینگ (Profiling): اندازه‌گیری کارایی، تحلیل مؤخر = مرآنیز اجرای برنامه‌ی درست بر روی مجموعه داده‌های ورودی جهت تعیین زمان و حافظه‌ی لازم برای محاسبه‌ی نتیجه است

تحلیل الگوریتم ها (پیچیدگی زمانی)

فرض کنید یک کامپیوتر با یک RAM داریم که در آن الف) دستورالعمل ها یکی بعد از دیگری اجرا می‌شوند (معادله پردازش موازی نیستیم) ب) ذخیره و بازیابی هر عضو در مدت زمان ثابت انجام می‌شود

ب) پردازش ترتیبی ← CPU یک هسته — پردازش موازی ← CPU چهارهسته‌ای - هشت هسته‌ای

آن چه برای تحلیل یک الگوریتم لازم است ۱- تعیین عملیات مورد استفاده مثل عمل اصلی - دستورات مقایسه‌ای - خواندن و نوشتن - فراخوانی پردازه‌ها (Process) - دستورات انتساب $a=2$ و محاسبه کردن زمان اجرای آن‌ها ۲- تعیین تعداد کافی از مجموعه‌ی داده‌ها برای تعیین تمامی الگوهای رفتاری ممکن (بدترین حالت، بهترین حالت، حالت متوسط) * (بسته به ورودی رفتار متفاوت است) *

مرتبه زمانی الگوریتم

زمان اجرای یک الگوریتم بستگی دارد به:

- سخت افزار — مربوط به پروفیلینگ — بعد از پیاده سازی
  - سرعت پردازنده
  - زبان برنامه نویسی
  - کامپایلر
- نوع دستورها و تعداد دستورها — تحلیل مفهوم — قبل از پیاده سازی
  - تعداد ورودی‌ها و خروجی‌ها

زبان های برنامه نویسی

- زبان های سطح بالا — به زبان کاربر — Python, C#, java
- زبان های سطح میانی — C, Pascal
- زبان های سطح پایین — زبان ماشین — assembly

تحلیل مقدم = $tP(n) = \sum$ (زمان اجرا) * (تعداد دفعات اجرا)
مربوط به سخت افزار
زمان اجرای یک الگوریتم با n ورودی

$tP(n) = \sum^{\text{دستورالعمل}}$ تعداد دفعات اجرا

فرض: فرض می‌کنیم که زمان اجرای همه‌ی دستورالعمل‌ها مساوی و برابر واحد زمان CPU است

۱۱/۴

مثال)

الف) $x = x + y$ → ۱ بار

ب)
```
for i=1 to n do        n+1
    x=x+y              n  ⇒ n بار
```
$2n+1 \rightarrow t(n) \rightarrow$ پیچیدگی زمانی

ج)
```
for i=1 to n do
{
  for j=1 to n do  ⇒ n² بار
  {
    x=x+y
  }
}
```

نکته ۸/۱۰ معمولاً زمان اجرای یک الگوریتم یک چند جمله‌ای برحسب n است.

$t(n)$: c مقدار ثابت
$an+b$ خطی
an^2+bn+c مرتبه ۲

نکته ۸/۱۰ در تحلیل مقدم آن چیزی که مهم است مشخص کردن نوع منحنی زمانی است.

O= زمانی order (مرتبه زمانی) : مقدار ثابت - خطی - درجه ۲ ⇐ منحنی

مثال) نوع منحنی زمانی کد الگوریتم زیر را مشخص کنید

```
for i=1 to n do
{ j=i                ⇒ n
  while j>0 do                      ⇒ n.log n = O
    j=[j/2]          ⇒ log n
}
```

$j=[j/2]$: دستور اصلی → داخلی ترین

زمانی که دستور تقسیم بر ۲ یا ضرب در ۲ شود جواب log می شود

نکته 8 ٪: هرگاه در تحلیل زمانی دستور حلقه داشتیم اگر گام حلقه در عددی ثابت مثلاً ۲ ضرب یا بر آن تقسیم شد مرتبه زمانی آن حلقه $\log^n$ در مبنای آن عدد ثابت خواهد بود

نمادهای مجانبی: O (بزرگ) ← بدترین حالت، Ω (امگا بزرگ) ← بهترین، Θ ← حالت متوسط، o (o کوچک)، ω (امگا کوچک) 8

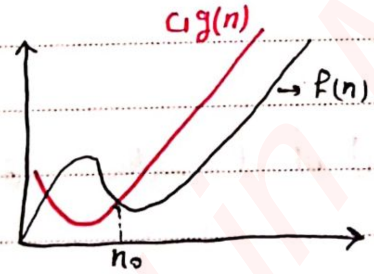
تعریف O (بدترین حالت) =

$f(n)=O(g(n))$

$\Leftrightarrow \exists c_1>0 \ \forall \geq n_0 \ f(n) \leq c_1 \, g(n)$

c_1: عدد حقیقی، n_0: عدد طبیعی، $f(n)$, $g(n)$: معرف زمان اجرا

تعبیر هندسی =



مثال)

$f(n)=7n^2+4n+9$ → می خواهیم بدترین حالت را پیدا کنیم → جواب $= O(n^2)$ بزرگترین توان همان big o است یعنی بدترین حالت

$c_1 g(n)$: یعنی تابعی پیدا کنیم که از $f(n)$ بزرگتر باشد مانند $8n^2$ ، n^3

مثال)

$f(n) = 8n^2 + 3n^3 \log n + 4n - 1$ $O(n^2 \log n)$

$f(n) = 3n^3 + 4n^2 + 3 \log n + 1$ $O(n^3)$

قضیه ۱: اگر $A(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0$, $a_m > 0$ آنگاه $A(n) = O(n^m)$

نتیجه ۱: اگر برنامه‌ای دارای k دستور الف به درجات بزرگی $O(n^{m_1}), O(n^{m_2}), \dots, O(n^{m_k})$ باشد آنگاه درجه بزرگی برنامه برابر: $O(n^m)$ است که $m = \text{maximum}\{m_1, m_2, \dots, m_k\}$

نتیجه ۲ (قانون جمع): اگر برنامه P از دو قطعه برنامه‌ای P_1 و P_2 به ترتیب با زمان‌های اجرای $T_1(n)$ و $T_2(n)$ تشکیل شده باشد که در آن $T_1(n) = O(f(n))$ و $T_2(n) = O(g(n))$ در این صورت زمان اجرای برنامه‌ای P یعنی $T(n)$ برابر با حداکثر بودن $O(\text{maximum}\{f(n), g(n)\})$

مثال)

نتیجه ۳ (قانون حاصل‌ضرب): اگر $T_1(n) = O(f(n))$, $T_2(n) = O(g(n))$ آنگاه $T_1(n)\, T_2(n) = O(f(n)\, g(n))$

مثال)

$T_1 = 3n^3 + 4$ $\downarrow O(n^3)$ $T_2(n) = 7n \log n + 4n + 1$ $O(n \log n)$ $T_1(n)\, T_2(n) = O(n^4 \log n)$

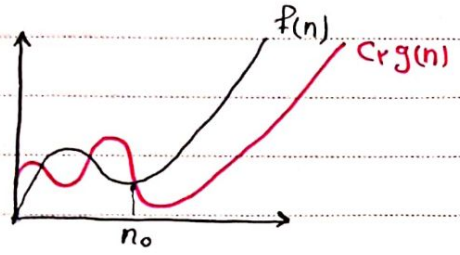
تعریف Ω (بررسی بهترین حالت اجرای الگوریتم)

$f(n) = \Omega(g(n))$

$\Leftrightarrow \exists\, c_2 > 0 \quad \forall n \geq n_0 \quad f(n) \geq c_2 g(n)$

عدد حقیقی — عدد طبیعی

تعبیر هندسی



مثال)

$f(n) = 7n^2 + 4n + 9 \quad \Omega(n^2)$

$4n^2$

Ωn

تعریف Θ (بررسی حالت میانگین اجرای الگوریتم)

$f(n) = \Theta(g(n))$

$\Leftrightarrow \exists\, c_1, c_2 > 0 \quad \forall n \geq n_0 \quad c_2 g(n) \leq f(n) \leq c_1 g(n)$

حقیقی — طبیعی

* ارتباط $\Theta = \Omega = O$ (فشردگی)

$f(n) = 7n^2 + 4n + 9$

$O(n^2)$
$\Omega(n^2)$ $\Rightarrow \Theta = (n^2)$

۱۲/۱۸

تعریف O (صورت):

$f(n) = O(g(n)) \longleftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} \to 0$

مثال) $7n^2+8n+3$

$$\lim_{n\to\infty} \frac{7n^2+8n+3}{n^3} \to 0 \qquad O(n^3)$$

$n^3 \to$ بزرگتر برای اینکه به صفر میل کند

مثال) $\log n+7$

$$\lim_{n\to\infty} \frac{\log n+7}{n} \to 0 \qquad O(n)$$

تعریف ω (امگا کوچک)

$$f(n) = \omega(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} \to \infty$$

مثال)

$$\lim \frac{7n^2+8n+3}{n} \to \infty \qquad \omega(n)$$

$n \to$ کوچکتر برای اینکه به بینهایت میل کند

مثال)

$$\lim \frac{3n+2}{\log n} \to \infty \quad \omega(\log n) \qquad \lim \frac{\log n+7}{\text{عدد ثابت}} \to \infty \quad \omega(1)$$

تشابه بین نمادهای O، Ω، Θ، o، ω در مقایسه بین زمان‌های اجرای الگوریتم‌ها و یا ($\leq$، $\geq$، $<$، $>$، $=$) در مقایسه بین اعداد طبیعی

$a \leq b \approx \to f(n) = O(g(n))$

$a \geq b \approx \to f(n) = \Omega(g(n))$ $\qquad$ $a < b \approx \to f(n) = o(g(n))$

$a = b \approx \to f(n) = \Theta(g(n))$ $\qquad$ $a > b \to f(n) = \omega(g(n))$

تحلیل الگوریتم مرتب سازی حبابی (Bubble sort)

Agorithm Bubble sort (A, n) ← ورودی ، تعداد ورودی

آرایه A=[1...n] ورودی

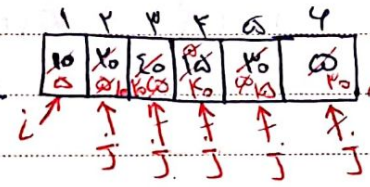
هدف: مرتب سازی آرایه مزبور

(a) for i=1 to n-1
  (b) for j=n down to i+1 do
    (c) if A[j] < A[j-1] then
    {
      (d) temp = A[j-1]
      (e) A[j-1] = A[j]
      (f) A[j] = temp
    }

معنی جای A[j] و A[j-1] عوض می شود

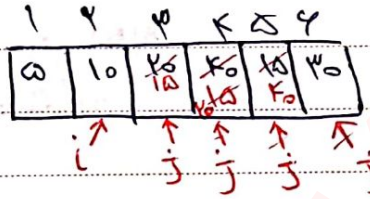
توضیح با مثال 8

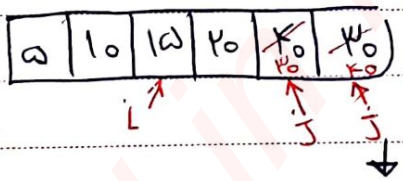
interation 1



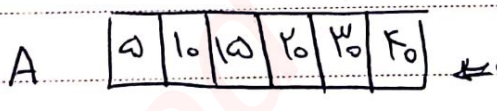
در هر مرحله خانه ای که زیر دست را با خانه قبلی مقایسه می کنیم و اگر کوچکتر بود اعداد با هم جابجا می شوند

interation 2





دوباره همان کارها را می کنیم تا برسیم به



A

خروجی مرتب شده

بدترین حالت 8 آرایه نزولی باشد یعنی همواره شرط دستور c برقرار باشد

i	j	تعداد دفعات اجرا
i=1	n ... ۲	1 + a(n-1) → (اجرا - اجرا) +1
i=۲	n ... ۳	1 + a(n-۲)
i=۳	n ... ۴	1 + a(n-۳)
⋮	⋮	⋮
i=n-1	1	1 + a(1)

a → b, c, d, e, f → یعنی تمام این دستورها یکسان اجرا شوند

جمع اعداد n-1 تا n

$\Rightarrow n + a\frac{(n(n-1))}{2}$ $O(n^2)$

بهترین حالت 8 آرایه مرتب شده صعودی باشد یعنی شرط دستور c هیچ‌گاه برقرار نباشد

i	j	تعداد دفعات اجرا
i=1	n ... ۲	1+۲(n-1)
i=۲	n ... ۳	1+۲(n-۲)
⋮	⋮	⋮
n-1	1	1+۲(1)

a, b, c

a, b, c → چون شرط c برقرار نیست پس فقط b, c برقرار نیستند یعنی دیگر d, e, f نیستند

$\Rightarrow n + 2\frac{(n(n-1))}{2}$ $\Omega(n^2)$ $\theta(n^2)$

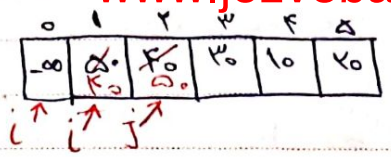
تحلیل الگوریتم مرتب سازی درجی (Insertion sort)

ورودی آرایه A[1...n] ... A[i...n]

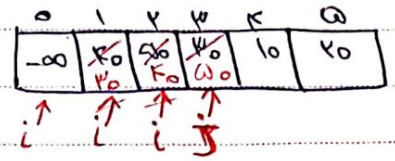
خروجی A[0] = $-\infty$

```
Algorithm Insertion sort (A,n)
A[0] = -∞
for j = ۲ to n do
{
  item = A[j] ,i=j-1
  while item < A[i] do *
  {
    A[i+1] = A[i]
    i = i-1
  }
  A[i+1] = item
}
```

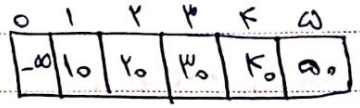
توضیح با مثال



① $j=2$ item $=40$



② $j=3$ item $=30$



مرحله آخر

بدترین حالت: دستور while دستور تعیین کننده است دستور * همواره برقرار باشد یعنی آرایه برعکس مرتب شده باشد

j	دستور * چند بار اجرا میشود؟
۲	۲
۳	۳
⋮	⋮
n	n بار

$$\Rightarrow \underbrace{-1+1+2+3+\dots+n}_{\times} \Rightarrow \frac{n(n+1)}{2}-1 \quad O(n^2)$$

بهترین حالت: آرایه صعودی باشد

j	دستور * چند بار اجرا میشود
۲	۱ ← یعنی فقط * را انجام شد و برقرار نباشد از حلقه بیرون می‌رود
۳	۱
⋮	⋮
n	n بار

$n-2+1 = n-1$

$$\Rightarrow n-1\,(1) = n-1 \qquad \Omega(n) \qquad \Theta=?$$

درجات بزرگی متداول برای الگوریتم‌ها:

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^2\log n) <$$

سرعت ←

$$O(n^3) < O(2^n) < O(n!)$$

مرتبه اجرای الگوریتم‌های بازگشتی 8 : در توابع بازگشتی مرتبه زمانی متناسب با تعداد فراخوانی‌های تابع است

برای حل یک رابطه‌ی بازگشتی ۳ راه حل وجود دارد ۱- جایگذاری با شمار ۲- معادله مشخصه (فصل بعد) ۳- تابع مولد (سری)

۴- ترسیم درخت بازگشتی

مثال) مرتبه اجرای الگوریتم بازگشتی محاسبه فاکتوریل را بدست آورید

روش ۱- ترسیم درخت بازگشت 8 برای مثال n=۴ می‌کشیم.

```
int fact (int n)
{
 if(n==1) return 1;
 return (n*fact(n-1));
}
```

fact(۴)
↓
۴*fact(۳)
↓
۳*fact(۲)
↓
۲*fact(۱)

تعداد فراخوانی‌ها = n بار ⇒ $T(n)=O(n)$ (مرتبه اجرای)

$$\begin{cases} T(n)=1 & n=1 \\ T(n)=T(n-1)+C & \end{cases}$$

fact(n-1) فراخوانی‌های ↓ ... ،* ،if ↓ یعنی زمان‌هایی که عملیات‌ها را انجام می‌دهیم تا به n-۱ برسیم

روش ۲- جایگذاری با شمار 8

$T(1)=1$
$T(2)=T(1)+C=1+C$
$T(3)=T(2)+C=1+2C$
$T(4)=T(3)+C=1+3C$
⋮
$T(n)=1+(n-1)C$
⇓
$O(n)$

مثال) مرتبه اجرای تابع زیر را بدست آورید

```
int T (int n)
{
 if (n<=1) return 1;
 else return T(n/۲)+T(n/۲)
```

$n=8 \Rightarrow T(8)$

$T(6)$ $T(6)$

$T(4)$ $T(4)$ $T(4)$ $T(4)$

$T(2)$ $T(2)$ $T(2)$ $T(2)$ $T(2)$ $T(2)$ $T(2)$ $T(2)$

$T(1)$ $T(0)$ $T(1)$ $T(0)$ $T(1)$ $T(0)$ $T(1)$ $T(1)$

$T(1)=1$

$T(2)=3$ $\quad T(n)=2n-1$

$T(4)=7$ $\quad O(n)$

$T(8)=15$

مثال) مرتبه اجرای تابع فیبوناچی را بدست آورید

n:	0	1	2	3	4	5	6	...
fib:	0	1	1	2	3	5	8	...

فرمول بازگشتی

$$\begin{cases} fib(n) = fib(n-1) + fib(n-2) & \text{جمع - مرتبه ۲} \\ fib(0)=0 \ , \ fib(1)=1 & \text{شرط خاتمه یا شرط مرزی} \end{cases}$$

معادله مشخصه

```
Algorithm fib(n)
if (n<=1) then return n
else return (fib(n-1) + fib(n-2))
```

$fib(n) - fib(n-1) - fib(n-2) = 0$

معادله مشخصه

$C_n r^2 + C_{n-1} r + C_{n-2} = 0$

ضریب جمله n ام — ضریب جمله n-1 ام — ضریب جمله n-2 ام

$$r^2 - r - 1 = 0 \longrightarrow r_1, r_2 = \frac{-b \pm \sqrt{b^2-4ac}}{2a} \Rightarrow \frac{1 \pm \sqrt{1^2+4}}{2}$$

ریشه‌ها

$r_1 = \frac{1+\sqrt{5}}{2}$

$r_2 = \frac{1-\sqrt{5}}{2}$

دو ریشه حقیقی متمایز

فرمول عمومی $\boxed{Fib(n) = C_1 r_1^n + C_2 r_2^n}$

$$\Rightarrow fib(n) = C_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + C_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$fib(0): \ C_1 \left(\frac{1+\sqrt{5}}{2}\right)^0 + C_2 \left(\frac{1-\sqrt{5}}{2}\right)^0 = 0 \Rightarrow C_1 + C_2 = 0 \Rightarrow C_1 = -C_2$$

$$fib(1): \ C_1 \left(\frac{1+\sqrt{5}}{2}\right)^1 + C_2 \left(\frac{1-\sqrt{5}}{2}\right)^1 = 1 \Rightarrow -C_2 \left(\frac{1+\sqrt{5}}{2}\right) + C_2 \left(\frac{1-\sqrt{5}}{2}\right) = 1$$

$$\Rightarrow \frac{-c_2 - \sqrt{5}c_2 + c_2 - \sqrt{5}c_2}{2} = 1 \Rightarrow \frac{-\sqrt{5}c_2}{2} = 1 \Rightarrow -\sqrt{5}c_2 = 1 \Rightarrow c_2 = -\frac{1}{\sqrt{5}}, \quad c_1 = \frac{1}{\sqrt{5}}$$

$$Fib(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n \Rightarrow \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

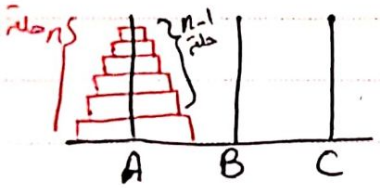
$\frac{1+\sqrt{5}}{2} \to 1,4$ ، $\frac{1-\sqrt{5}}{2} \to 0,4$

$$O(1,4)^n$$

برج هانوی 8

مسئله برج هانوی را با ۳ میله‌ی A و B و C در نظر بگیرید در این جا هیچ حلقه‌ای را نمی‌توان مستقیماً از A به B یا از B به A منتقل کرد و چنین انتقال‌هایی تنها به کمک میله‌ی C انجام می‌شود اگر در شروع کار n حلقه در میله‌ی A داشته باشیم و $t(n)$ حداقل تعداد جابه‌جایی‌ها برای انتقال n حلقه از A به B باشد الف) یک رابطه‌ی بازگشتی برای $t(n)$ ارائه دهید ب) این رابطه‌ی بازگشتی را حل کنید؟

قوانین: در برج هانوی هیچ حلقه‌ای نمی‌تواند روی حلقه‌ی کوچک‌تر از خودش قرار گیرد و در هر مرحله فقط یک حلقه جابجا می‌شود



الف) $t(n)$ ← حداقل تعداد جابه‌جایی‌ها

$T(n-1)$ ← $n-1$ حلقه را با یک ستون C از A به B منتقل می‌کنیم

$2T(n-1)+1$ ← جابه‌جایی $n-1$ حلقه از C به A

↓ برای جابه‌جایی بزرگ‌ترین حلقه به میله‌ی C

$2T(n-1)+2$

↓ جابه‌جایی حلقه بزرگ‌تر از C به B

$3T(n-1)+2$ ← جابه‌جایی دوباره $n-1$ حلقه از A به B

$$T(n) = \begin{cases} T(1)=2 & n=1 \quad \text{1 حلقه} \\ 3T(n-1)+2 & n>1 \end{cases}$$

↓

$T(n) =$

ب) روش جایگذاری با تکرار:

$$T(n) = 3T(n-1) + 2 \Rightarrow 2 + 3T(n-1)$$

$3T(n-2)+2$

$$T(n) = 2 + 3\left[3T(n-2)+2\right] \Rightarrow 2 \times 3 + 3^2\left[T(n-2)\right]$$

↓

$2 + 3^2 T(n-3)$

$= ۲ + ۲\times۳ + ۳^۲ [۲ + ۳T(n-۳)] \Rightarrow ۲ + ۲\times۳ + ۲\times۳^۲ + ۳^۳ \underbrace{T(n-۳)}_{۲+۳T(n-۴)}$

$= ۲ + ۲\times۳ + ۲\times۳^۲ + ۲\times۳^۳ + ۳^۴ T(n-۴)$

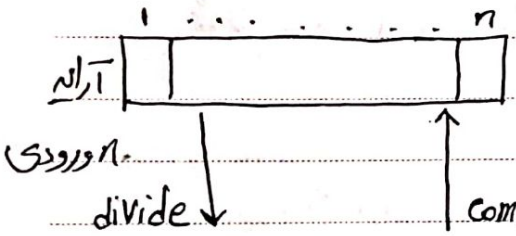
$\Rightarrow ۲ + ۲\times۳ + ۲\times۳^۲ + ۲\times۳^۳ + \dots + ۲\times۳^{n-۲} + ۳^{n-۱} T(\underbrace{۱}_{۲})$ ($\frac{n-(n-۱)}{۱}$ برای پیدا کردن تعداد)

$t(n) = \frac{۲(۳^n - ۱)}{۳-۱} = ۳^n - ۱ \quad \Leftarrow \frac{t_۱(q^n-۱)}{q-۱}$

$O(۳^n)$

۹۸/۱/۲۴

روش‌های تقسیم و غلبه (Divide and conquer) (D&C)



ایده روش‌های تقسیم و غلبه، تقسیم مسئله به زیر مسئله‌های کوچک‌تر (divide) و حل این زیر مسئله‌های و در نهایت ادغام آن‌ها برای رسیدن به جواب نهایی است

مسئله‌ای با n ورودی که در آرایه‌ی A ذخیره شده است را می‌خواهیم حل کنیم.

صورت عمومی یک الگوریتم تقسیم و غلبه:

```
Algorithm D&C (low, high)
if small (low, high) return G (low, high)   → یک تابع بولی است که می‌گوید آیا مسئله به اندازه کافی کوچک است یا خیر
else
{
mid = Divide (low, high)
return combine (D&C (low, mid), D&C (mid+1, high))
```

(تقسیم و غلبه بازگشتی هم هست) * $D\&C(1,n) \leftrightarrow A$

زمان اجرای الگوریتم‌های تقسیم و غلبه:

$$t(n) = \begin{cases} g(n) \\ t(\lceil n/2 \rceil) + t(\lfloor n/2 \rfloor) + f(n) \end{cases}$$

$g(n)$: برای n های کوچک ← دو حالتی که آرایه کامل باشد

$t(\lceil n/2 \rceil) + t(\lfloor n/2 \rfloor) + f(n)$: برای n های بزرگ ← آرایه‌ها زیادند پس باید تقسیم کنیم

فقط برای divide یعنی زمان شکستن مسئله به زیر مسئله‌های کوچک‌تر

برای conquer زمان ترکیب جواب‌های زیر مسئله‌های کوچک

$\lceil\ \rceil$ $\lfloor\ \rfloor$ ← یعنی ممکن است دو طرف متقارن نباشد

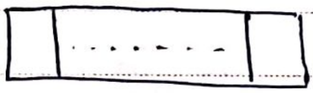
$$t(n) = \begin{cases} g(n) & \text{برای n های کوچک} \\ t(n/2) + t(n/2) + f(n) \end{cases}$$

$n=2$ برای راحتی کار

$$2t[n/2] + f(n)$$

با فرض: $mid = \left\lfloor \frac{low+high}{2} \right\rfloor$

مثال) الگوریتم تقسیم و غلبه برای پیدا کردن عنصر ماکزیمم یک آرایه:

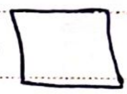


تعداد مقایسه‌ها

- شرط خاتمه: اگر آرایه یک عضوی باشد: n مقایسه

max: خودش

آرایه تک عضوی $high = low$

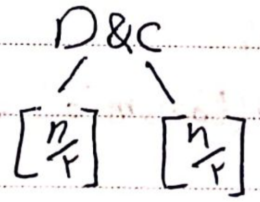


- اگر آرایه دو عضوی باشد: ۱ مقایسه

max: عدد بزرگتر

$high = low + 1$ ← یعنی آرایه دو عددی

- $n > 2$ مسئله بزرگ است ⇐



این روش های تقسیم و غلبه برای حل مسائل مشابه روش بازگشتی است برای یافتن عنصر max یک آرایه n عنصری ۵ اگر n=1 باشد ← max = خودش و ۰ مقایسه

اگر n=2 باشد ← با یک مقایسه می‌توان max را بدست آورد

برای n های بزرگ‌تر از ۲ ← آرایه به دو زیرآرایه با تعداد عناصر $\lceil \frac{n}{2} \rceil$ و $\lfloor \frac{n}{2} \rfloor$ تقسیم می‌کنیم و به طور بازگشتی (max1 , max2)

عنصر max این دو زیرآرایه را بدست می‌آوریم. برای پیدا کردن max کل آرایه به یک مقایسه بین max1 و max2 نیاز داریم

مثال)

Algorithm Findmax (A, Low, high, fmax)

(آرایه اعداد، ابتدا، انتها: ورودی‌ها — ماکزیمم: خروجی)

۱ و ۲ شرط‌های خاتمه

۱ـ یعنی یک عضو دارد آرایه پس max خودش است

۲ـ آرایه ۲ عضو دارد خانه اول low و خانه دوم high و با مقایسه max بدست می‌آید

۳ـ n بزرگ شود باید شکسته شود mid را که وسط است بدست می‌آوریم و باید max بالا و max پایین را بدست آوریم و با مقایسه ۲ max هر کدام بزرگ‌تر باشد max کل می‌شود

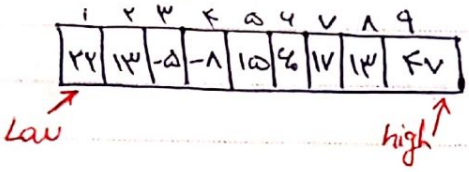
```
1 if (Low = high) then fmax = A[Low]        (if small)
2 else if (high = low+1) then
      if A[Low] > A[high] then
         fmax = A[Low]
      else fmax = A[high]
3 else {                                     (divide)
      mid = [(Low+high)/2]
      Find max (A, Low, high, max1)
      Find max (A, Low, high, max2)
   if max1 > max2 then fmax = max1           (conquer (combine))
   else fmax = max2
```

findmax (A, 1, 9, fmax)

ترسیم روی مثال ۶

ورودی:

$A[1\ldots9] = [۲۲, ۱۳, -۵, -۸, ۱۵, ۶۰, ۱۷, ۱۴, ۴۷]$



تحلیل زمانی ۶ تعداد مقایسه ها

$$t(n) = \begin{cases} t\lceil n/2 \rceil + t\lfloor n/2 \rfloor + f(n) & \text{مقایسه آخر بین Max و Max}_2 \\ t(1) = 0 \ , \ t(2) = 1 & \text{حالت خاص} \end{cases}$$

$n = 2^m$

$$\begin{cases} t(n) = 2t(n/2) + 1 \\ t(1) = 0 \ , \ t(2) = 1 \end{cases}$$

روش جایگذاری با شماره

$$t(n) = 1 + 2t(n/2) \xrightarrow{\text{برای } n \leftarrow n/2} 2t(n/2^2)$$

$$t(n) = 1 + 2 + 2^2 t(n/2^2) \xrightarrow{\text{برای } n \leftarrow n/2^2} 1 + 2t(n/2^3)$$

$$t(n) = 1 + 2 + 2^2 + 2^3 t(n/2^3)$$

$$t(n) = 1 + 2 + 2^2 + 2^3 + 2^4 t(n/2^4)$$

$$t(n) = 1 + 2 + 2^2 + 2^3 + 2^4 + \ldots + 2^m t(1)$$

n باید تقسیم بر چند شود تا 1 شود $\leftarrow \frac{n}{n} = \frac{n}{2^m}$

$\leftarrow$ یعنی برای $\frac{n}{2^m}$

تعداد جمله‌ها = عددی که ضرب شده و داریم $= \frac{t_1(q^n - 1)}{q - 1} = \frac{2^m - 1}{1}$

$$O(2^m) \xrightarrow{n = 2^m} O(n)$$

تمرین ۸: الگوریتم fmax را طوری تغییر دهید که علاوه بر یافتن max و عنصر min نیز پیدا کند و مرتبه زمانی آن را مشخص کنید.

قضیه اصلی برای حل روابط تقسیم و غلبه 8: این قضیه به ما این امکان را می‌دهد که خیلی سریع جواب یک رابطه تقسیم و غلبه را برحسب Θ بدست آوریم

$$\begin{cases} t(n) = a\,t(n/b) + cn^k \\ t(1) = c \end{cases}$$

که در آن a و b و c و k اعداد مثبت و $a \geq 1$ و $b > 1$

الف) اگر $a > b^k$ $\quad t(n) = \Theta(n^{\log_b^a})$

ب) اگر $a = b^k$ $\quad t(n) = \Theta(n^k \log n)$

پ) اگر $a < b^k$ $\quad t(n) = \Theta(n^k)$

۱/۱/۳۱

جواب رابطه‌های تقسیم و غلبه‌های زیر را بدست آورید

الف) $t(n) = 2t(n/2) + 1$ ← همان فرمول find max $\quad$ حالت الف $t(n) = \Theta(n^{\log_b^a})$

$a=2 \quad b=2 \quad k=0 \quad a > b^k \Rightarrow 2 > 2^0$ $\quad t(n) = \Theta(n^{\log_2^2}) \Rightarrow \Theta(n)$

ب) $t(n) = 2t(n/2) + n$

$a=2 \quad b=2 \quad k=1 \quad 2 = 2^1$ حالت b $\quad t(n) = \Theta(n^k \log n)$

$t(n) = \Theta(n \log n)$

ب) $t(n)=t(\frac{2n}{3})+1$

$a=1 \quad b=\frac{3}{2} \quad k=0$ حالت دوم $1=(\frac{3}{2})^0$

$t(n)=\theta(n^k \log n)$

$t(n)=\theta(\log n)$

پ) $t(n)=3t(n/4)+n^2$

$a=3 \quad b=4 \quad k=2$ حالت سوم $3<4^2 \rightarrow 3<b^k$

$t(n)=\theta(n^k)$

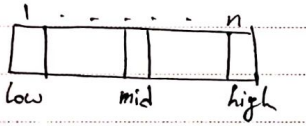
$\theta(n^2)$

الگوریتم جست‌وجوی دودویی binary search:

آرایه را به صورت صعودی مرتب می‌شود

* کلید x ← اگر جست‌وجو موفق باشد اندیس را برمی‌گرداند
اگر جست‌وجو موفق نباشد صفر را برمی‌گرداند

$mid=\left[\frac{low+high}{2}\right]$

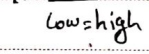


x را با mid مقایسه می‌کنیم و ۳ حالت دارد:

1. $x=A[mid]$ ← جست‌وجو موفق ①
2. $x<A[mid]$ ← از low تا mid-1 ②
3. $x>A[mid]$ ← از mid+1 تا high ③

*

اگر آرایه A یک عضو داشته باشد نیاز مقایسه لازم است



آرایه‌ی مرتب شده $A[1..n]$ به صورت صعودی و کلید x مفروض است

هدف: آیا کلید x در آرایه A هست یا خیر؟ * پس داریم

① $low=high$ و شرط خاتمه‌ی می‌شود

② $high>low$ داریم * یعنی حالتی که بیشتر از یک عضو داشته باشد آرایه

Year. Month. Date. ( )

Algorithm Binary search (low, high, x, j) خروجی که اندیس است یا صفر

(ورودی — ابتدا، انتهای آرایه — کلید)

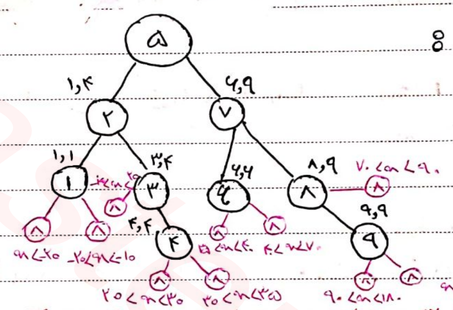
mid = $\left[\frac{low+high}{2}\right]$

① if x = A[mid] then j = mid return A
else
② if x < A[mid] then Binary search (low, mid - 1, x, j)
else
③ Binary search (mid + 1, high, x, j)

توضیح با مثال:

i	۱	۲	۳	۴	۵	۶	۷	۸	۹
A[i] =	-۲۰	-۱۰	۲۰	۳۰	۳۵	۴۰	۷۰	۹۰	۱۸۰
مقایسه‌های موفق	۳	۲	۳	۴	۱	۳	۲	۳	۴

درخت جستجو:



mid = $\frac{9+1}{2}$ = ۵

mid = $\frac{1+4}{2}$ = ۲/۵

mid = $\frac{6+9}{2}$ = ۷/۵

mid = $\frac{1+1}{2}$ = ۱

mid $\frac{3+4}{2}$ = ۳

mid $\frac{8+9}{2}$ = ۸

مقایسه ناموفق = x<-۲۰ / -۲۰<x<-۱۰ / -۱۰<x<۲۰ / ۲۰<x<۳۰ / ۳۰<x<۳۵ / ۳۵<x<۴۰ / ۴۰<x<۷۰ / ۷۰<x<۹۰ / ۹۰<x<۱۸۰ / x>۱۸۰

x = -۱۰ را جستجو کنیم؟ j = ۲ در حالت موفق است

x = ۱۹۰ را جستجو کنیم؟ جستجو ناموفق است چون مقدار ۱۹۰ وجود ندارد

تحلیل زمانی الگوریتم جستجوی دودویی؟

جستجوی موفق ـ جستجوی ناموفق

حالت‌ها \ مرتبه زمانی	بهترین حالت	بدترین حالت	حالت میانگین
جستجوی موفق	$\theta(1)$	$\theta(\log n)$	$\theta(\log n)$
جستجوی ناموفق	$\theta(\log n)$	$\theta(\log n)$	$\theta(\log n)$

میانگین تعداد مقایسه‌ها برای جستجوی موفق = $\frac{3+2+3+4+1+3+2+3+4}{9}$ = ۲/۷۷

میانگین تعداد مقایسه‌ها برای جستجوی ناموفق = $\frac{3+3+3+4+4+3+3+3+4+4}{10}$ = ۳/۴

PAPCO

Year. Month. Date. ( )

n تعداد عناصر آرایه ⇐ $2^3 < 9 < 2^4$

جستجوی موفق ⇐ $2^{k-1} < n < 2^k$ حداکثر k مقایسه حداقل 1

جستجوی ناموفق ⇐ حداقل k-1 و حداکثر k

اگر $2^{k-1} \le n \le 2^k$ آنگاه الگوریتم جستجوی دودویی مستلزم حداقل یک و حداکثر k مقایسه برای جستجوی موفق و حداکثر k مقایسه و حداقل k-1 مقایسه برای جستجوی ناموفق است

مسئله ضرب دو چند جمله‌ای 8

$A(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$

0	1		n
a_0	a_1	...	a_n

(توان‌های x ← اندیس‌ها؛ ضریب‌ها ← محتوا)

$B(x) = b_n x^n + b_{n-1} x^{n-1} + \dots + b_1 x + b_0$

0	1		n
b_0	b_1	...	b_n

$C(x) = A(x) \cdot B(x) = C_{2n} x^{2n} + C_{2n-1} x^{2n-1} + \dots + C_1 x + C_0$

از درجه 2n

0	1		2n
		...	

$C_0 = a_0 b_0$

$C_1 = a_1 b_0 + a_0 b_1$

$C_2 = a_2 b_0 + a_1 b_1 + a_0 b_2$ $\qquad C_k = \sum_{i=0}^{k} a_i b_{k-i}$

$\vdots$

$k = 0, 1, \dots, 2n$

$$A(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

$$B(x) = b_n x^n + b_{n-1} x^{n-1} + \dots + b_1 x + b_0$$

$$C(x) = A(x)\,B(x) = c_{2n} x^{2n} + c_{2n-1} x^{2n-1} + \dots + c_1 x + c_0$$

A: $a_0 \mid a_1 \mid \dots \mid a_{n-1} \mid a_n$ — $n+1$

B: $b_0 \mid b_1 \mid \dots \mid b_{n-1} \mid b_n$ — $n+1$

C: $c_0 \mid c_1 \mid \dots \mid c_{2n-1} \mid c_{2n}$ — $2n-1$, $2n$

ضعف ضرایب

$c_0 = a_0 b_0$

$c_1 = a_0 b_1 + a_1 b_0$ → یعنی حاصل ضرب a_i, b_j که $i+j=1$ است c_1 می‌شود

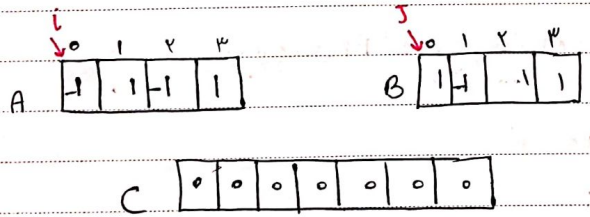
$c_2 = a_0 b_2 + a_1 b_1 + a_2 b_0$ → $a_1 b_1$ ، $1+1=2$

⋮

مثال) $A(x) = x^3 - x^2 + x - 1$ $\quad B(x) = x^3 + x^2 - x + 1$

روش‌ها: روش مجموعی — تقسیم و غلبه مجموعی — تقسیم و غلبه سریع

روش مجموعی




```
Algorithm multPoly (A,B,C)
for i:=0 to 2n do C[i]:=0 Repeat   2n+1   O(n) ?
for i:=0 to n do                   n ?
For j:=0 to n do                   n        } n^2+n  Θ(n^2)
PAPCO C[i+j] := C[i+j]+A[i]*B[j]
```

تا حلقه روی کامل نشود (j) i را نشان می‌دهیم. بعد از این که j به خانه آخر برسد i را یکی اضافه به جلو می‌بریم و دوباره j را تا آخر می‌بریم.

	0	1	2	3	4	5	6
C	0 -1	0 +2	0 +-2	0 -1 0	0 1	0	0

زمانی که $i=3$ ⇒

	0	1	2	3	4	5	6
C	-1	2	-3	2	-1	0	1

$x^6 - x^4 + 2x^3 - 3x^2 + 2x - 1$

روش تقسیم و غلبه (معمولی)

$P(x), Q(x)$ از درجه n

$R(x) = P(x)\,Q(x)$ از درجه $2n$

توضیح روی مثال

$P(x) = x^3 - x^2 + x - 1$ $\qquad$ $Q(x) = x^3 + x^2 - x + 1$

↓ از کوچک به بزرگ

$P(x) = -1 + x - x^2 + x^3$ $\qquad$ $Q(x) = +1 - x + x^2 + x^3$

$\underbrace{-1+x}_{P\,l(x)} + x^2\underbrace{(-1+x)}_{P\,r(x)}$ $\qquad$ $\underbrace{1-x}_{Q\,l(x)} + x^2\underbrace{(1+x)}_{Q\,r(x)}$

↓

$P\,l(x) + x^2 P\,r(x)$ $\qquad$ $Q\,l(x) + x^2 Q\,r(x)$

$R(x) = P(x)Q(x) \Rightarrow P\,l(x)Q\,l(x) + x^2 P\,l(x)Q\,r(x) + x^2 P\,r(x)Q\,l(x) + x^4 P\,r(x)Q\,r(x)$

$\Rightarrow P\,l(x)\,Q\,l(x) + x^2\left[P\,l(x)Q\,r(x) + P\,r(x)Q\,l(x)\right] + x^4 P\,r(x)Q\,r(x)$

$P\,l(x)Q\,l(x) = (-1+x)(1-x) = -1 + 2x - x^2$

$P\,l(x)Q\,r(x) = (-1+x)(1+x) = x^2 - 1$

$P\,r(x)Q\,l(x) = (-1+x)(1-x) = -1 + 2x - x^2$

$P\,r(x)Q\,r(x) = (-1+x)(1+x) = x^2 - 1$

جایگذاری $= -1 + 2x - x^2 + x^2[\underbrace{x^2 - 1 - 1 + 2x - x^2}_{2x-2}] + x^4(x^2-1) = -1 + 2x - 3x^2 + 2x^3 - x^4 + x^6$

درس ۷:

$P(x) = 1 - x + 2x^2 + 3x^3 + 4x^4 - 5x^5 + 6x^6 - 7x^7$

$P(x) = \underbrace{1 - x + 2x^2 + 3x^3}_{Pl(x)} + x^4 \underbrace{(4 - 5x + 6x^2 - 7x^3)}_{Pr(x)}$

فرض: درحالت کلی و برای راحتی کار $n = 2^m$ و $P(x)$، $Q(x)$ چندجمله‌ای‌هایی از درجه $n-1$ خواهند بود $2^m - 1$

نحوه‌ی $R(x) = Pl(x)\,Ql(x) + x^{\frac{n}{2}}\left[Pl(x)\,Qr(x) + Pr(x)\,Ql(x)\right] + x^n\left(Pr(x)\,Qr(x)\right)$

مرتبه زمانی:

یعنی ۴ تا عبارت داریم که درجه $\frac{n}{2}$ دارند

$t(n) = 4t\left(\frac{n}{2}\right) + Cn$ (۵)

زمان ترکیب جمع و ضرب‌های خطی

قضیه اصلی:

$a = 4 \quad b = 2 \quad k = 1$

$a \;\; b^k$

$4 > 2^1$

$t_n = \Theta\left(n^{\log_b^a}\right) = \Theta\left(n^{\log_2^4}\right) = \Theta(n^2)$

۹۸/۲/۱۴

روش ۳ ضرب دوجمله‌ای به روش تقسیم و غلبه سریع:

$R(x) = \underbrace{Pl(x)\,Ql(x)}_{Rl} + x^{\frac{n}{2}}\underbrace{\left[Pl(x)\,Qr(x) + Pr(x)\,Ql(x)\right]}_{\circledast} + x^n \underbrace{Pr(x)\,Qr(x)}_{Rr}$

$Rm = [Pl + Pr][Ql + Qr] = \underbrace{Pl\,Ql}_{Rl} + \underbrace{Pl\,Qr + Pr\,Ql}_{\circledast} + \underbrace{Pr\,Qr}_{Rr}$

$\Rightarrow \circledast = Rm - Rl - Rr$

مثال:

$P(x) = -1 + x - x^2 + x^3 \qquad Q(x) = 1 - x + x^2 + x^3$

$P(x) = \underbrace{-1 + x}_{Pl} + x^2\underbrace{(-1 + x)}_{Pr} \qquad Q(x) = \underbrace{1 - x}_{Ql} + x^2\underbrace{(1 + x)}_{Qr}$

① $R_l = P_lQ_l = -1+2n-n^2$ (14) ⊛ $= -4+4n+1-2n+n^2-n^2+1 = -2+2n$

② $R_r = P_rQ_r = n^2-1$

③ $R_m = [P_l+P_r][Q_l+Q_r] = [-2+2n][2] = 4n-4$

هزینه زمانی : $t(n) = 3t(n/2)+cn$ $a \quad b^k$ $a=3$, $b=2$, $k=1$, $3>2$

$t(n) = \Theta(n^{\log_b^a}) = \Theta(n^{\log_2^3}) \Rightarrow \Theta(n^{1.58})$

روش های حریصانه (greedy) :

۱- روش های هستند تکرار شونده . ۲- برای بهینه سازی (ماکزیمم یا مینیمم) کردن یک تابع به کار می رود

۳- این روش ها معمولاً دارای n ورودی بوده و مسئله از ما می خواهد زیرمجموعه ای از این n ورودی را بدست آوریم به گونه ای که در شرایط معینی صادق باشد

۴- اگر فرض کنید که ورودی یک آرایه n عنصری مثل $A[1 \ldots n]$ باشد : در هر مرحله یکی از عناصر در آرایه بر اساس یک تابع گزینش انتخاب و اگر اضافه کردن آن به مجموعه ی جواب های منتخب امکان پذیر باشد به مجموعه ی جواب اضافه می شود

صورت عمومی یک الگوریتم حریصانه :

```
Algorithm Greedy (A,n)
  solution = Φ
for i=1 to n do
{
 a=select (A)
 if Feasible (a, solution)
 solution = union (a, solution)
```

- solution = Φ : در ابتدا مجموعه جواب خالی است
- for i=1 to n do : به ازای همه ورودی ها
- a=select (A) : یک ورودی از آرایه A انتخاب می کند و آن را از آرایه حذف می کند
- if Feasible (a, solution) : Feasible یک تابع بولی است که مشخص می کند آیا a می تواند به مجموعه جواب اضافه شود یا خیر؟
- solution = union (a, solution) : a را به مجموعه جواب اضافه می کند

مسئله کوله پشتی:

فرض: یک کوله پشتی به ظرفیت M و n جسم داریم شماره گذاری شده از $1, \dots, n$

وزن جسم i: w_i

ارزش جسم i: P_i

هدف: $0 \le x_i \le 1$ ، $\sum_{i=1}^{n} x_i P_i$ را maximize کنیم
(x_i: درصد برداشتن از جسم i)

مشروط به آن که $\sum_{i=1}^{n} x_i w_i \le M$ یعنی چیزهایی که در کوله پشتی قرار می‌دهیم بیشتر از ظرفیت کوله نباشد.

توضیح الگوریتم همراه با مثال: فرض $n=3$ ، $M=20$

$(P_1, P_2, P_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$ $\quad (x_1, x_2, x_3) = ?$

برای مثال $(1/2, 1/3, 1/4)$ $\quad w = 16.5$ $\quad P = 12.5 + 8 + \frac{15}{4}$

$(0, 2/3, 1)$ $\quad w \le 20$ $\quad P = 31$

$(1, 2/15, 0)$ $\quad w \le 20$ $\quad P = 28.2$

برای حل مسئله کوله پشتی اگر اجسام را به ترتیب نزولی ارزش بر واحد وزن انتخاب کنیم جواب بهینه حاصل می‌شود.

$$\frac{P_1}{w_1} \ge \frac{P_2}{w_2} \ge \frac{P_3}{w_3}$$

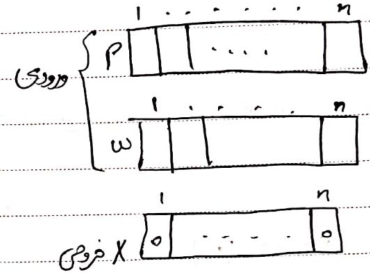
مرحله ۱: مسئله مرتب سازی $\frac{P_1}{w_1}$ تا $\frac{P_n}{w_n}$

مرحله ۲: حل مسئله کوله پشتی

```
Algorithm Greedy_knapsack (n, w, P, M, X, Profit)
```
(n, w, P, M: ورودی — X, Profit: خروجی — Profit: ارزش نهایی)

```
X = 0, Profit = 0, CU = M
```
(CU: ظرفیت باقی‌مانده از کوله‌پشتی)

```
for i = 1 to n do
{
   if w[i] > CU then exit
   x[i] = 1 , Profit + P[i]
     CU = CU - w[i]
}
   x[i] = CU / w[i] ;
   Profit = Profit + X[i] * P[i]
```



مثال)

$n = 6 \quad M = 14$

$P = (P_1, P_2, P_3, P_4, P_5, P_6) = (10, 7, 12, 13, 6, 20)$

$w = (w_1, w_2, w_3, w_4, w_5, w_6) = (2, 1, 3, 2, 12, 8)$

$\frac{P}{w} = (5, 7, 4, 6/5, 0/5, 2/5)$ ترتیب اولویت انتخاب جسم‌ها: ③ ① ④ ② ⑥ ⑤

(اندیس‌ها: ۱ ۲ ۳ ۴ ۵ ۶)

مرتبه زمانی = $\Theta(n \log n)$ مرتب‌سازی

خود مسئله کوله‌پشتی = $\Theta(n)$

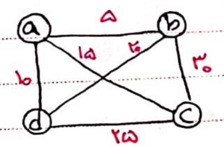
شماره جسم i	w_i	P_i	$\frac{P_i}{w_i}$	مقدار برداشت x_i	ظرفیت باقی‌مانده CU	ارزش نهایی Profit
۲	۱	۷	۷	۱	۱۴-۱=۱۳	۰+۷=۷
۴	۲	۱۳	۶/۵	۱	۱۳-۲=۱۱	۷+۱۳=۲۰
۱	۲	۱۰	۵	۱	۱۱-۲=۹	۲۰+۱۰=۳۰
۳	۳	۱۲	۴	۱	۹-۳=۶	۳۰+۱۲=۴۲
۶	۸	۲۰	۲/۵	$\frac{6}{8}$	۰	۴۲+۲۰×۶/۸=۵۷
۵	۱۲	۶	۰/۵	۰	—	—

Profit = ۵۸ $\quad$ X = [1, 1, 1, 1, 0, 6/8] (۱ ۲ ۳ ۴ ۵ ۶) بردار جواب

www.jozveban.ir

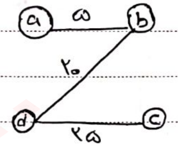
درخت پوشا: فرض کنید $G=(V,E)$ (رئوس، یال) یک گراف همبند و بدون جهت باشد. زیرگراف $T=(V',E')$ را یک درخت پوشا برای G گوییم هرگاه ۱. T = یک درخت باشد ۲. $V'=V$

برای مثال)



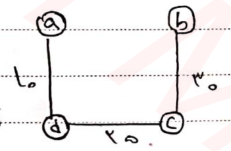
گراف کامل مرتبه ۴ k_4 = تعداد درخت‌های پوشا = $4^{4-2}=16$

درخت‌های پوشا

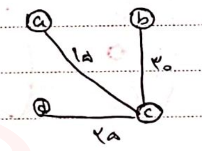


Cost = ۵۰

قضیه = تعداد درخت‌های پوشای گراف کامل k_n = n^{n-2}



Cost = ۶۵



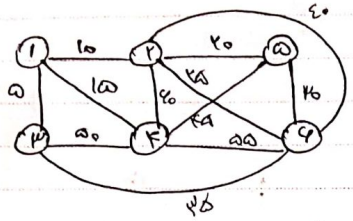
Cost = ۷۰

درخت پوشای مینیموم: ۱. پریم ۲. کراسکال

برای بدست آوردن یک درخت پوشا برای یک گراف وزن‌دار دو الگوریتم پریم و کراسکال استفاده می‌شود. در هر یک از این الگوریتم‌ها فرض بر این است که ورودی $G=(V,E,Cost)$ که G یک گراف وزن‌دار همبند است که در آن $V=\{1,2,\dots,n\}$ مجموعه رئوس و $Cost[i,j]$ که $i,j=1,2,\dots,n$ یک ماتریس $n\times n$ که معرف هزینه یال‌ها است. (ماتریس هزینه‌ها)

خروجی هر کدام از این الگوریتم‌ها T آرایه‌ای شکل T با $n-1$ یال برای درخت پوشای مینیموم است.

Scanned by CamScanner



$G = (V, E, Cost)$

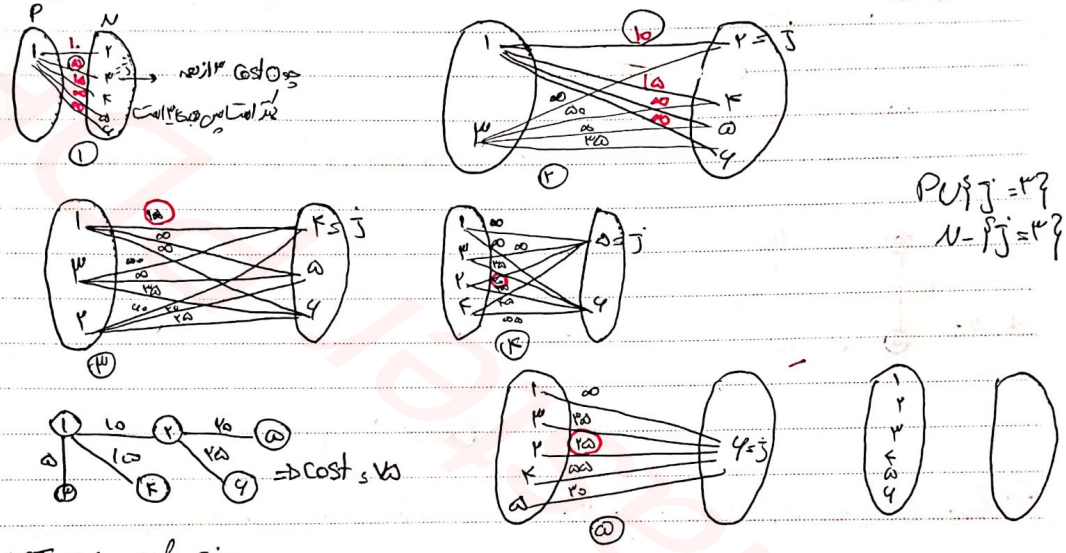
$V = \{1, 2, \dots n\}$

	1	2	3	4	5	6
1	0	10	5	15	∞	∞
2	10	0	∞	40	20	25
3	5	∞	0	50	∞	35
4	15	40	50	0	45	55
5	∞	20	∞	45	0	30
6	∞	25	35	55	30	0

= Cost

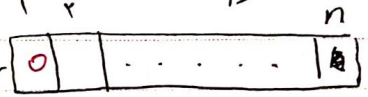
6×6

الگوریتم پریم (Prim) O



MST درخت پوشای مینیمم

ایده الگوریتم O: از یک راس دلخواه شروع می‌کنیم که مجموعه راس شروع را راس ۱ در نظر می‌گیرند این راس را پردازش شده و سایر رئوس را پردازش نشده فرض می‌کنیم در هر مرحله رأسی از بین پردازش نشده‌ها به گونه‌ای انتخاب می‌کنیم که کوتاه‌ترین مال یکی از راس‌های پردازش شده داشته باشیم این راس را j در نظر می‌گیریم j را از مجموعه‌ی پردازش نشده حذف و به مجموعه‌ی پردازش شده اضافه می‌کنیم



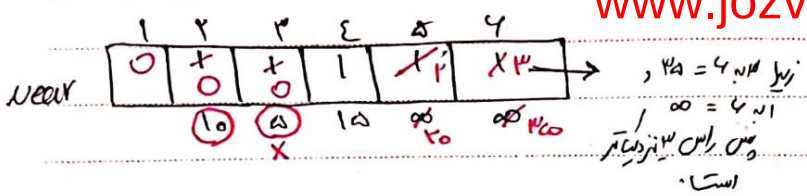
near[j] = 0 ⇐ آیا راس j پردازش شده است

near[j] = i ≠ 0 ⇐ نشده است ~ ~ ~ ~

i ∈ P

P4PCO

مینیمم cost[i][j]



Algorithm Prim (n, cost, T, mincost)

تعداد رئوس — ماتریس هزینه — ماتریس درختی — ماتریس خروجی

near[1] ← 0 , mincost ← 0 O(1)

for i ← 2 to n do near[i] ← 1 Repeat O(n)

for i ← 1 to n-1 do O(n)

{

راس j را به گونه‌ای انتخاب کن که near[j] ≠ 0 و cost[j, near[j]] مینیمم باشد

}

(T[i,1] , T[i,2]) ← (j , near[j])

min cost ← min cost + cost[j, near[j]]

near[j] ← 0

for k ← 2 to n do O(n)

if (near[k] ≠ 0) and (cost[k,j] < cost[k, near[k]]) then near[k] ← j

تغییر دادن خانه‌ای که 1 بود به 3 شامل

پیچیدگی = O(n)

مثال: با استفاده از الگوریتم پریم درخت پوشای مینیموم گراف قبلی را بدست آورید

j	near	cost[j, near[j]]
۱	۰	—
۲	۱	۱۰
۳	۱	۵
۴	۱	۱۵
۵	۱	∞
۶	۱	∞

j = ۳ ⇒

j	near	cost[j, near[j]]
۱	۰	—
۲	۱	۱۰
۳	۰	—
۴	۱	۱۵
۵	۱	∞
۶	۴	۴۵

j = ۲ ⇒

j	near	cost[j, near[j]]
۱	۰	—
۲	۰	—
۳	۰	—
۴	۱	۱۵
۵	۲	۲۰
۶	۲	۲۵

j = ۴ ⇒

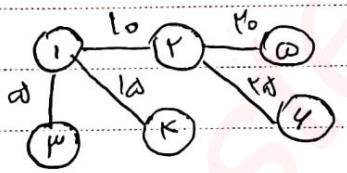
j	near	cost[j, near[j]]
۱	۰	—
۲	۰	—
۳	۰	—
۴	۰	—
۵	۲	۲۰
۶	۲	۲۵

j = ۵ ⇒

j	near	cost[j, near[j]]
۱	۰	—
۲	۰	—
۳	۰	—
۴	۰	—
۵	۰	—
۶	۲	۲۵

j = ۶

j	near	cost
۱	۰	—
۲	۰	—
۳	۰	—
۴	۰	—
۵	۰	—
۶	۰	—



minCost = ۷۵

الگوریتم کراسکال

با استفاده از الگوریتم کراسکال درخت پوشای مینیموم گراف قبل را بدست آورید

مرحله اول: مرتب سازی یال ها به صورت صعودی براساس هزینه یال ها.

مرحله دوم: با شروع از ابتدای این لیست مرتب شده یالی را انتخاب می کنیم که اضافه کردن آن به مجموعه یال های قبلاً انتخاب شده دور تشکیل ندهد این عمل تعداد انتخاب ۱-n امین یال پایان می یابد.

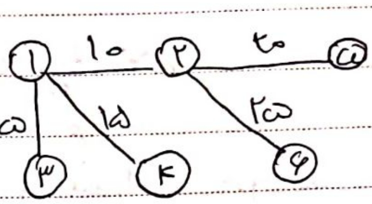
جدول مرتب شده یال ها:

شماره یال	هزینه	یال
۱	۵	۱,۳
۲	۱۰	۱,۲
۳	۱۵	۱,۴
۴		
۵		
۶		
۷		

$n-1$ ≤ انتخاب می‌یال

تعداد رأس‌ها = ۶

شماره یال i	هزینه	یال
۱	۵	۱,۳
۲	۱۰	۱,۲
۳	۱۵	۱,۴
۴	۲۰	۲,۵
۵	۲۵	۲,۶
۶	۳۰	۵,۶
۷	۳۵	۳,۶
۸	۴۰	۲,۶
۹	۴۵	۴,۵
۱۰	۵۰	۳,۴
۱۱	۵۵	۴,۶
۱۲	۶۰	۲,۴



minCos = ۷۵

تعداد یال‌ها: n = Edge

$O(\overset{n}{edge}.\log(\overset{n}{edge}))$

بهترین حالت: $n-1$ یال اول جدول انتخاب شوند — انتخاب یال

بدترین حالت: یال $n-1$ ام یال آخر جدول باشد

(یال‌ها ← edge، رأس ← n، ماتریس هزینه ← cost: ورودی) (درخت ← T، مینیمم هزینه ← mincost: خروجی)

```
Algorith kruskal(edge, n, cost, T, mincost)
i = 1 , Nedge = 0 , min Cos = 0
        (یال‌های مشخص)
while Nedge < n-1 do
{
  (k,L) = (edge[i,1], edge[i,2])
  if ( اضافه کردن (k,l) به T تشکیل دور ندهد ) then
    Nedge = Nedge + 1
    (T[Nedge,1], T[Nedge,2]) = (k,l)
    mincost = mincost + cost[k,l]
}
```

مسئله انتخاب فعالیت‌ها (زمانبندی فعالیت‌ها) Activity selection

فرض: مجموعه‌ای از n فعالیت $S=\{1,2,\dots,n\}$ (برنامه) و یک منبع (CPU) واحد مفروض است

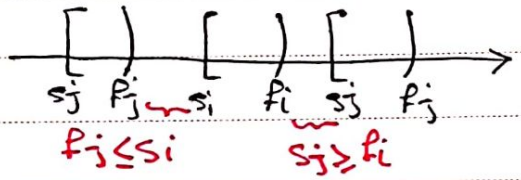
فاصله‌ی زمانی برای استفاده فعالیت i از منبع مفروض به صورت $[s_i, f_i)$ است

s_i = زمان شروع f_i = زمان پایان

در هر لحظه فقط یک فعالیت می‌تواند از منبع استفاده کند

هدف: انتخاب بیشترین تعداد فعالیت سازگار است

تعریف: دو فعالیت را سازگار گوییم هرگاه برای استفاده از منبع همپوشانی زمانی نداشته باشند



الگوریتم مربوطه: ۱- مرتب سازی فعالیت‌ها به صورت صعودی بر اساس زمان پایانشان f_i

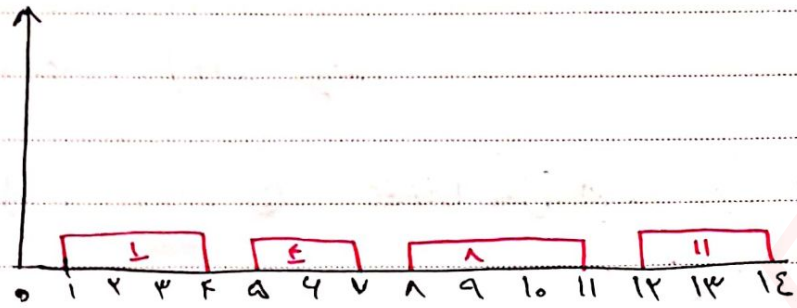
ورودی: دو آرایه

```
Algorithm Activity-selection (s, f)
A={1}, j=1 → فعالیت اول        O(n)        برای مرتب سازی n log(n) ⇒ n log n
for i=2 , to n do → از فعالیت دوم آنهایی که همپوشانی
    if s[i] ≥ f[j]   نداشته باشد می‌رود سراغ بعدی
    {
      A = A∪{i}
      j=i
    }
return A
```

مثال) مراحل مختلف برای الگوریتم انتخاب فعالیت ها را برای فعالیت‌های زیر بنویسید

i	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱
شروع s_i	۱	۳	۰	۱۲	۲	۵	۳	۸	۸	۵	۶
پایان f_i	۴	۵	۶	۱۴	۱۳	۹	۸	۱۲	۱۱	۷	۱۰
	①	④	③	⑪	⑩	⑥	⑤	⑨	⑧	④	⑦
	✓	X	X	✓	X	X	X	X	✓	✓	X

مرحله ۱: براساس f_i مرتب‌سازی



۹۸/۳/۴

الگوریتم کوتاه‌ترین مسیرهای هم مبدأ (الگوریتم دایکسترا)

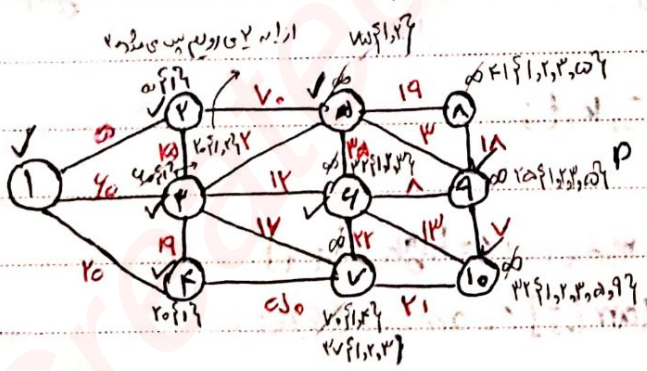
Dijkstra shortest Paths

فرض: $G=(V,E,Cost)$ $V=\{1,2,\dots,n\}$ V_0 را مبدأ

هدف: پیداکردن کوتاه‌ترین مسیرها از مبدا V_0 به سایر رئوس گراف

* برای راحتی معمولاً V_0 را راس ۱ درنظر می‌گیرند

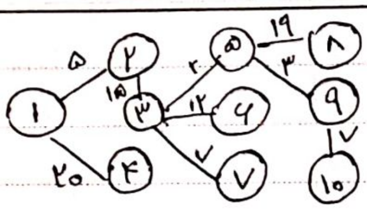
مثال) توضیح الگوریتم همراه با مثال:



	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
P	۱	۰	۰	۰	۰	۰	۰	۰	۰	۰
	①	②	④	③	⑤	⑧	⑨	⑩	⑥	⑦

$P[i]=0$ اگر پردازش نشده باشد

$P[i]=1$ اگر پردازش شده باشد



درخت پوشای کوتاه‌ترین مسیرهای هم مبدأ SPST

$O(n^2)$ = مرتبه زمانی

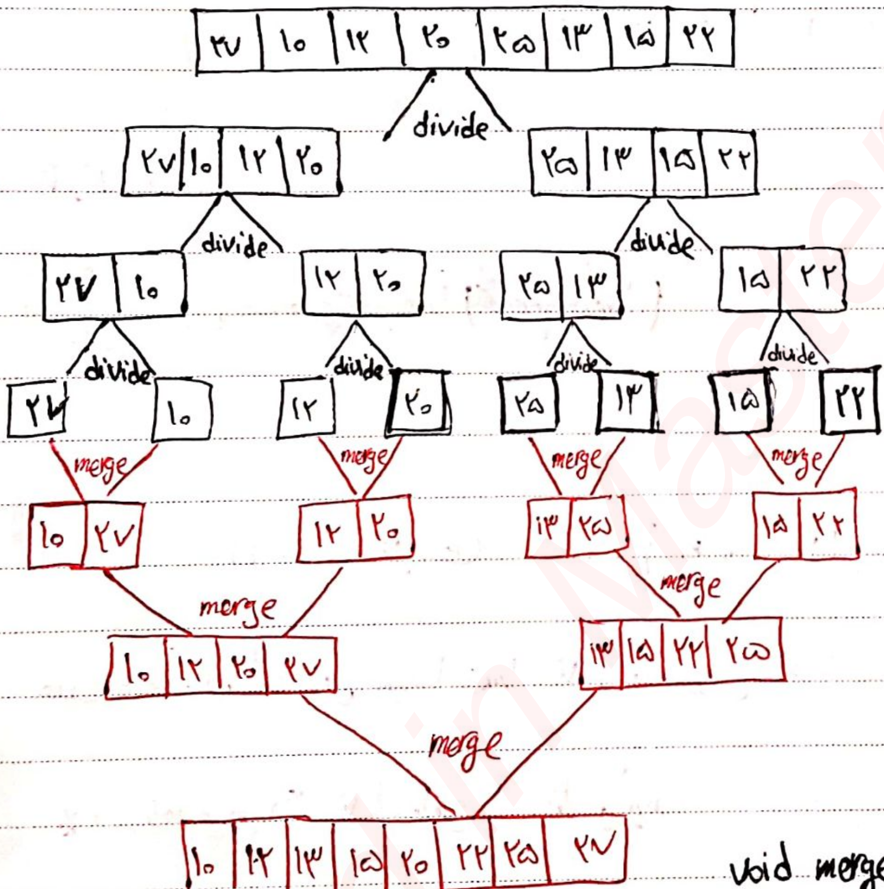
نکته: لزوماً SPST با MST یکسان نیست

* تمرین MST را بدست بیاوریم (با پریم یا کراسکال)

ادامه روش های تقسیم و غلبه: (D&C)

مرتب سازی ادغامی (Merge Sort)

مثال) توضیح با مثال: دو مرحله: ① Divide شکستن آرایه تا به آرایه تک عضوی برسیم ② merge ترکیب جواب ها تا رسیدن به جواب نهایی




```
void merge sort(int arr[], int low, int high)
{
if low>=high
return;
int mid=(low+high)/2;
merge sort(arr, low, mid);
merge sort(arr, mid+1, high);
merge(arr, low, mid, high)
```

الگوریتم merge:

```
void merge(int arr[], int low, int mid, int high)
{
int i,j,k,t;
j=low;
for(i=mid+1; i<=high, i++)
{while (arr[j]<=arr[i] && j<i)
j++;
if(i==j)
```

$t = arr[i];$

```
for (k=i ; k>j ; k--)
arr[k] = arr[k-1]
arr[j] = t;
```

هزینه زمانی مرتب سازی ادغامی:

$t(n) = 2t(n/2) + cn$

مرحله ادغام n عدد

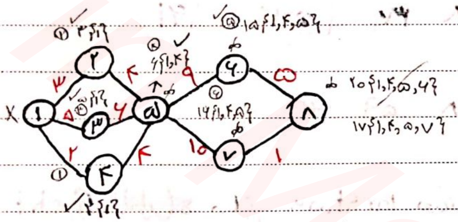
$a=2 \quad b=2 \quad k=1$

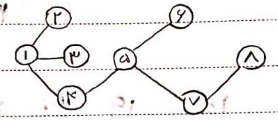
$a \quad b^k$

$2 = 2^1 \Rightarrow \theta(n \log n)$

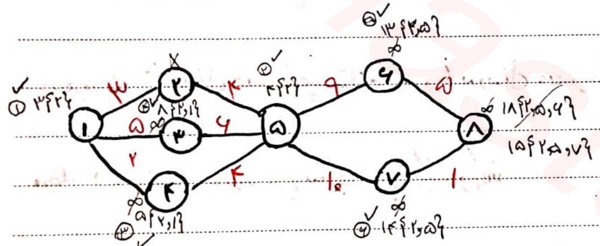
۹۸/۱۴/۱۱

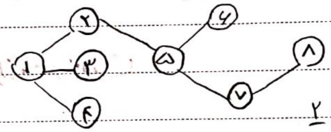
SPST





SPST به مبدا راس ۱





SPST به مبدا راس ۲

مرتب سازی سریع θ (Quick sort) به روش تقسیم و غلبه

مرتب سازی سریع بهترین رفتار را در حالت میانگین بین روش‌های مرتب سازی دارد. ایده الگوریتم به این صورت است

هر بار یک عنصر محوری (pivot) انتخاب شده و سایر عناصر در لیست به صورتی جابه‌جا می‌شوند که کلیه‌ی عناصر

کوچک‌تر از عنصر محوری در یک طرف و عناصر بزرگ‌تر از آن در طرف دیگر آن قرار بگیرد و سپس ۲ زیر لیست دو طرف

عنصر محوری به همین ترتیب مرتب سازی می‌شود (به صورت بازگشتی)

* معمولاً اولین عنصر لیست را به عنوان عنصر محوری درنظر می‌گیرند

مثال
توضیح کلاس

۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۲۶ (Pivot)	۵	۳۷	۱	۶۱	۱۱	۵۹	۱۵	۴۸	۱۹

عنصر بزرگ‌تر از محور — عنصر کوچک‌تر از عنصر محوری

* جای i و j عوض می‌شود

|۱۱ ۵ ۱۹ ۱ ۱۵| |۲۶| |۵۹ ۶۱ ۴۸ ۳۷|

جای j با عنصر محوری عوض می‌شود
i و j اشاره می‌کنند

|۱ ۵ | * | ۱۱ | ۱۹ ۱۵| |۴۸ ۳۷| |۵۹| |۶۱|

j به محتوای ۱ اشاره می‌کند که از ۱۱ کوچک‌تر است و stop می‌کند

i به محتوای ۱۹ اشاره می‌کند که از ۱۱ بزرگ‌تر است و stop می‌کند

محتوای i و j جابجا می‌شود

در آخر ۱ ۵ ۱۱ ۱۵ ۱۹ ۳۷ ۴۸ ۵۹ ۶۱

هم i و j هر دو به خونه‌ی شماره‌ی ۲ که محتوای آن ۵ است اشاره می‌کنند

Partition (افراز): در تابع Partition دو متغیر i و j به ترتیب به حد بالا و پایین لیست اشاره می‌کنند
و وقتی که هر دو به یک خونه اشاره می‌کنند جای آن با pivot عوض می‌شود.

در هر مرحله j دنبال عنصر کوچک‌تر از محور و i به دنبال عنصر بزرگ‌تر از محور است

مرحله ۲ i و j به صورت زیر به طرف هم حرکت می‌کنند

① به i افزوده می‌شود تا زمانی که list[i] > Pivot

② j کاهش می‌یابد تا زمانی که list[j] < Pivot

③ اگر i<j باشد آنگاه محتویات list[i] با list[j] عوض می‌شود و اگر j<=i آنگاه
جای Pivot با list[i] عوض می‌شود

الگوریتم صفحه بعد

```
void Quick sort (element list[ ], int left, int right)
{
int Pivot, i, j;
element temp;
if( left < right) {
i = left, j = right+1;
do
{
do
i++;
while ( list [i].key < Pivot);
do
j--;
while (list[j].key > Pivot);
if(i<j)
SWAP (list[i], List[j], temp);
}
while (i<j);
SWAP (list[left], list[j], temp;

Quick sort (list, left, j-1);
Quick sort (list, j+1, right);
}
}
```

Partition

$t(n) = 2t(n/2) + cn$ → مرتبه خطی

$a=2 \; ; \; b=2 \quad k=1 \quad 2=2^1 \quad O(n \log n)$

مرتبه کلی

کدهافمن Huffman coding برای فشرده سازی اطلاعات

کدگذاری هافمن یکی از روش‌های فشرده‌سازی برای ارسال فایل‌ها به‌صورت کم حجم در بستر شبکه است.

کاراکتر	تعداد تکرار	فضای اشغالی
A	۲۰	۲۰×۲ = ۴۰
B	۱۰	۱۰×۲ = ۲۰
C	۵	۵×۲ = ۱۰
		۷۰

کدگذاری اسکی ← ۷ بیتی A,B,C

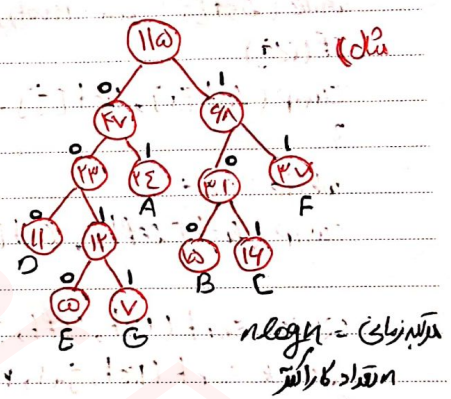
توضیح الگوریتم: مرحله ۱: ابتدا تعداد تکرار هر کاراکتر درون را حساب می‌کنیم مرحله ۲: دو کاراکتر با کمترین تعداد تکرار را انتخاب می‌کنیم مرحله ۳: کاراکترهای مرحله ۲ را مجموع تکرار آن‌ها جایگزین می‌کنیم مرحله ۴: تا زمان رسیدن به یک کاراکتر مرحله ۲ را تکرار می‌کنیم مرحله ۵: از چالیات فوق یک درخت حاصل می‌شود. بر روی این درخت هر مسیر به چپ را صفر و هر مسیر به راست را با یک وزن دهی می‌کنیم مرحله ۶: کد هر کاراکتر را از ریشه تا آن کاراکتر حساب می‌کنیم

مثال:

تکرار

A	۲۴	E	۵
B	۱۵	G	۷
C	۱۶	D	۱۱
D	۱۱	B	۱۵
E	۵	C	۱۶
F	۳۷	A	۲۴
G	۷	F	۳۷

مرتب‌سازی →



حرف	تعداد تکرار	کد هافمن	تعداد بیت مورد نیاز با هافمن	تعداد بیت مورد نیاز در حالت اسکی
A	۲۴	۰۱	۲×۲۴	۷×۲۴
B	۱۵	۱۰۰	۳×۱۵	۷×۱۵
C	۱۶	۱۰۱	۳×۱۶	۷×۱۶
D	۱۱	۰۰۰	۳×۱۱	۷×۱۱
E	۵	۰۰۱۰	۴×۵	۷×۵
F	۳۷	۱۱	۲×۳۷	۷×۳۷
G	۷	۰۰۱۱	۴×۷	۷×۷
			۲۷۰ بیت	۱۰۵۷ بیت

درآمد زمانی = nlogn
n تعداد کاراکتر