DAA

- What is asymptotic?

If something is asymptotic, it means its nature is of an asymptote.

- What is an asymptote?

Straight line that is limit value of an curve; can be considered as tangent at infinity.

- What is asymptotic efficiency?

In An algorithm's asymptotic efficiency, concern is how running time of an algorithm increases with size of an input in limit, as size of input increases without bound.

Different asymptotic notations

Θ O Ω o ω

Θ notation

$\Theta(g(n)) = \{ f_n : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$ $c_1, c_2, n_0 > 0 \}$

eg. $f(n) = \frac{1}{2}n^2 - 3n$

Let $g(n) = n^2$

We must determine +ve constants c_1, c_2 and n_0 such that

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$0 \leq c_1 n^2 \leq \frac{n^2}{2} - 3n \leq c_2 n^2$$

Dividing by n^2 gives

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

Now for $c_2 \geq \frac{1}{2}$, right hand equality holds for any value of $n \geq 1$

Now,

n	f(n)	n	f(n)
1	-5/2	7	1/14
2	-1	8	1/8
3	-1/2	9	1/6
4	-1/4	10	1/5
5	-1/10		
6	0		

Now for $c_1 \leq \frac{1}{14}$, left hand equality holds for any value of $n \geq 7$.

Thus by choosing $c_1 = 1/4$, $c_2 = 1/2$ & $n_0 = 7$, we can verify

$$\frac{1}{2}n^2 - 3n = \Theta(n^2).$$

We shall often use notation $\Theta(1)$ to either mean constant or constant function with respect to a varicble.

O notation

$O(g(n)) : \{ f(n) : 0 \le f(n) \le cg(n)$ for all $n \ge n_0$, $c, n_0 > 0 \}$

→ $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$
i.e. $\Theta(g(n)) \subseteq O(g(n))$

→ Since O notation describes an upper bound, when we use it to bound the worst case running time of an algo, we have bound on running time of algo on every input.

→ However $\Theta(n^2)$ on worst case running time of insertion sort doesn't imply $O(n^2)$ bound on running time of insertion sort for every input.
eg. When input is already sorted, insertion sort runs in $\Theta(n)$ time.

Date: / /

Ω notation

$\Omega(g(n)) : \{ f(n) : 0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$
$c, n_0 > 0 \}$

→ $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and
$f(n) = \Omega(g(n))$

→ When best case running time of some algo is $\Omega(g(n))$, it means running time of that algo for every input is $\Omega(g(n))$.

O notation

$O(g(n)) : \{ f(n) : 0 \leq f(n) < cg(n)$ for all $n \geq n_0$
$c, n_0 > 0 \}$

→ $2n = o(n^2)$ but $2n^2 \neq o(n^2)$

ω notation

$\omega(g(n)) : \{ f(n) : 0 \leq cg(n) < f(n)$ for all $n \geq n_0$
$c, n_0 > 0 \}$

→ $\frac{n^2}{2} = \omega(n)$ but $\frac{n^2}{2} \neq \omega(n^2)$

Analyzing Insertion Sort.

	INSERTION-SORT (A)	cost	times
1	for $j = 2$ to A.length	c_1	n
2	Key = A[j]	c_2	$n-1$
3	// Insert A[j] into sorted seq A[1..j-1]	0	$n-1$
4	$i = j-1$	c_4	$n-1$
5	while $i>0$ and A[i] > key	c_5	$\sum_{j=2}^{n} t_j$
6	A[i+1] = A[i]	c_6	$\sum_{j=2}^{n} (t_j-1)$
7	$i = i-1$	c_7	$\sum_{j=2}^{n} (t_j-1)$
8	A[i+1] = key	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1)$$

$$+ c_7 \sum_{j=2}^{n} (t_j - 1) + c_8 (n-1)$$

Best case R.T (i.e. Array is already sorted)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

Worst case R.T. (i.e. Array in reverse order)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n}{2}(n+1) - 1\right) + c_6\left(\frac{n}{2}(n-1)\right) + c_7\left(\frac{n}{2}(n-1)\right) + c_8(n-1)$$

$$\left[\because \sum_{j=2}^{n} j = \frac{n}{2}(n+1) - 1 \quad \text{and} \quad \sum_{j=2}^{n}(j-1) = \frac{n}{2}(n-1)\right]$$

$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} + \left(-\frac{c_6}{2}\right) - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

Average case R.T.

→ On avg, half elements in $A[1 \cdots j-1]$ are less than $A[j]$, and half elements are greater.

→ On avg, therefore we check half of subarray $A[1 \cdots j-1]$, and so t_j is about $\#. j/2$.

→ The resulting average case running time turns out to be quadratic function of input size, just like worst case R.T.

Order of growth

→ We used simplifying abstractions to ease our analysis of Insertion sort.

→ First we ignore actuals costs of each statement using constants c_i to represent these costs.

→ Then we observed that these constants give us more detail than we really need. We thus ignore not only actual statement costs, but also abstract costs c_i.

→ As it is order of growth that really interests us, we therefore consider only leading term of formula since lower order terms are relatively insignificant for large values of n.

→ We also ignore leading term's constant coefficient, since constant factors are less significant than rate of growth.

→ For insertion sort, when we ignore lower order terms & leading terms constant coefficient, we are left with factor n^2 from leading term.

→ We write insertion sort has worst case RT of $\Theta(n^2)$

Analysis of algorithms

Q What is recurrence or recurrence equation ?

A A recurrence or recurrence equation is an equation or inequality that describes a function in terms of its value on smaller inputs.

Solving Recurrences :-

METHOD 1 " Homogenous Linear Recurrence with constant coefficients "

$$a_0 t_n + a_1 t_{n-1} + \cdots + a_k t_{n-k} = 0$$

Charachteristic equation of the recurrence is

$$p(x) = a_0 x^k + a_1 x^{k-1} + \cdots + a_k$$

The eqⁿ will have k roots $r_1, r_2, \cdots r_k$

$$\therefore \quad t_n = \sum_{i=1}^{k} c_i r_i^n$$

More generally,

$$t_n = \sum_{i=1}^{l} \sum_{j=0}^{m_i - 1} c_{ij} n^j r_i^n$$

satisfies recurrence for any choice of constants $c_1, c_2, \ldots, c_k$.

EG 1 Consider $f_n = \begin{cases} n & \text{if } n=0 \text{ or } n=1 \\ f_{n-1} + f_{n-2} & \text{otherwise} \end{cases}$

Lets rewrite eq^n :- $f_n - f_{n-1} - f_{n-2} = 0$

$\therefore$ Charachteristic poly is $x^2 - x - 1$

Roots are :- $r_1 = \frac{1+\sqrt{5}}{2}$, $r_2 = \frac{1-\sqrt{5}}{2}$

Multiplicity are :- $m_1 = 1$, $m_2 = 1$

$\therefore$ General solution if of form :-

$$f_n = c_1 r_1^n + c_2 r_2^n \quad \text{---- i)}$$

Now, we know initial conditions i.e. f_0 and f_1. Substituting in (i), we get 2 eq^{n}s.

$$c_1 + c_2 = 0$$
$$r_1 c_1 + r_2 c_2 = 1$$

Solving these eq^n, we get $c_1 = \frac{1}{\sqrt{5}}$ & $c_2 = \frac{-1}{\sqrt{5}}$

Thus, $$\boxed{f_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]}$$

EG 2 Consider $t_n = \begin{cases} 0 & \text{if } n=0 \\ 5 & \text{if } n=1 \\ 3t_{n-1} + 4t_{n-2} & \text{otherwise} \end{cases}$

First lets rewrite eqn :- $t_n - 3t_{n-1} - 4t_{n-2} = 0$

∴ Charachteristic poly is $x^2 - 3x - 4 = 0$

$\Rightarrow (x+1)(x-4) = 0$

Roots are :- $r_1 = -1$, $r_2 = 4$

Multiplicity are :- $m_1 = 1$, $m_2 = 1$

∴ General solution is of of form :-

$$t_n = c_1 (-1)^n + c_2 4^n \quad \text{---------} \quad (i)$$

Now using initial conditions, we get

$c_1 + c_2 = 0$ for $n = 0$

$-c_1 + 4c_2 = 5$ for $n = 1$

Solving eqns, we get $c_1 = -1$ and $c_2 = 1$

Therefore, $\boxed{t_n = 4^n - (-1)^n}$.

METHOD 2 " Inhomogenous Recurrences "

$$a_0 t_n + a_1 t_{n-1} + \cdots + a_k t_{n-k} = b_1^n p_1(n) + b_2^n p_2(n) + \cdots$$

Here b is a constant
$p(n)$ is polynomial in n of degree d.

Corresponding charachteristic poly is

$$(a_0 x^k + a_1 x^{k-1} + \cdots + a_k)(x-b_1)^{d_1+1}(x-b_2)^{d_2+1}\cdots$$

EG 1 Consider the recurrence

$$t_n = \begin{cases} 0 & \text{if } n = 0 \\ 2t_{n-1} + n + 2^n & \text{otherwise} \end{cases}$$

First we rewrite recurrence as :-

$$t_n - 2t_{n-1} = n + 2^n$$

Here $b_1 = 1$, $p_1(n) = n$, $b_2 = 2$, $p_2(n) = 1$

∴ Charachteristic polynomial is $(x-2)(x-1)^2(x-2)$
$= (x-2)^2(x-1)^2$

Roots are : $r_1 = 1$ $r_2 = 2$
Multiplicity are : $m_1 = 2$ $m_2 = 2$

EG3 Consider recurrence

$$t_n = \begin{cases} n & \text{if } n = 0, 1 \text{ or } 2 \\ 5t_{n-1} - 8t_{n-2} + 4t_{n-3} & \text{otherwise} \end{cases}$$

First we rewrite recurrence as :-

$$t_n - 5t_{n-1} + 8t_{n-2} - 4t_{n-3} = 0$$

∴ Charachteristic polynomial is $x^3 - 5x^2 + 8x - 4$

$$= (x-1)(x-2)^2$$

Roots are :- $r_1 = 1 \quad r_2 = 2$

Multiplicity are :- $m_1 = 1 \quad m_2 = 2$

∴ General solution is

$$t_n = c_1 1^n + c_2 2^n + c_3 n 2^n$$

Using initial conditions, we get following eqns :-

$$c_1 + c_2 = 0 \quad \text{for } n = 0$$
$$c_1 + 2c_2 + 2c_3 = 1 \quad \text{for } n = 1$$
$$c_1 + 4c_2 + 8c_3 = 2 \quad \text{for } n = 2$$

Solving these eqns, we get $c_1 = -2$, $c_2 = 2$ and $c_3 = -\frac{1}{2}$

Thus, $\boxed{t_n = 2^{n+1} - n2^{n-1} - 2}$

∴ General solution is

$$t_n = c_1 1^n + c_2 n 1^n + c_3 2^n + c_4 n 2^n \quad \text{— (i)}$$

Substituting (i) in original recurrence, we get

$$n + 2^n = c_1 + c_2 n + c_3 2^n + c_4 n 2^n - 2\left(c_1 + c_2(n-1) + c_3 2^{n-1} + c_4(n-1)2^{n-1}\right)$$

$$= c_1 + c_2 n + c_3 2^n + c_4 n 2^n - 2c_1 - 2c_2 n + 2c_2 - c_3 2^n - c_4 n 2^n + c_4 2^n$$

$$n + 2^n = 2c_2 - c_1 - c_2 n + c_4 2^n$$

Equating coefficients on both sides we get,

$$c_4 2^n = 2^n \qquad -c_2 n = n$$

$$\Rightarrow \boxed{c_4 = 1} \qquad \Rightarrow \boxed{c_2 = -1}.$$

$$2c_2 - c_1 = 0$$

$$\Rightarrow c_1 = 2c_2$$

$$\boxed{c_1 = -2}$$

For finding c_3, use initial condition in (i),

$$\therefore t_0 = c_1 + c_2(0) + c_3 + c_4(0)$$

$$\therefore 0 = c_3 - 2$$

$$\Rightarrow \boxed{c_3 = 2}.$$

∴ General solution is

$$t_i = c_1 3^i + c_2 2^i$$

Now $T(2^i) = t_i$

∴ When $n = 2^i$, $T(n) = t_{\log n}$

∴ $T(n) = c_1 3^{\log n} + c_2 2^{\log n}$

$$T(n) = c_1 n^{\log 3} + c_2 n \quad \text{— (i)}$$

Now $T(1) = 1$ is given

$T(2) = 3T(1) + 2$

$= 5$.

Substituting $n = 1, 2$ in eqⁿ (i), we get

$c_1 + c_2 = 1$ for $n = 1$

$3c_1 + 2c_2 = 5$ for $n = 2$

Solving these eqⁿs we get, $c_1 = 3$ and $c_2 = -2$

∴ $\boxed{T(n) = 3n^{\log 3} - 2n}$

$$\therefore \quad t_n = c_1 (1)^n + c_2 (n) 1^n + c_3 2^n + c_4 n 2^n$$

$$\therefore \quad t_n = -2 + (-1) n + 2 \cdot 2^n + n 2^n$$

$$\therefore \quad t_n = 2^n (2+n) - (2+n)$$

$$\therefore \boxed{t_n = (2+n)(2^n - 1)}$$

METHOD 3 "Change of Variable"

EG 1 Consider recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(n/2) + n & \text{if } n \text{ is power of } 2,\ n > 1 \end{cases}$$

To transform it into known form, replace n by 2^i

We introduce new recurrence t_i, $t_i = T(2^i)$

$$\therefore \quad t_i = T(2^i) = 3T(2^{i-1}) + 2^i$$
$$= 3t_{i-1} + 2^i$$

Let's rewrite as $t_i - 3t_{i-1} = 2^i$

Here $b_1 = 2$, $d_2 + 1 = 1$

$\therefore$ Charachteristic polynomial is $(x-3)(x-2)$

Roots are : $r_1 = 3 \quad r_2 = 2$

Multiplicity are : $m_1 = 1 \quad m_2 = 1$

EG 2 Consider recurrence

$T(n) = 4T(n/2) + n^2$ when n is power of 2, $n \geq 2$.

Let $t_i = T(2^i)$

$\therefore t_i = 4T(2^{i-1}) + (2^i)^2$

$= 4t_{i-1} + 4^i$

Rewriting this recurrence,

$t_i - 4t_{i-1} = 4^i$

Here $b = 4$, $d + 1 = 1$

$\therefore$ Charachteristic polynomial is $(x-4)(x-4)$

$= (x-4)^2$

$\therefore$ Roots are $r_1 = 4$

Multiplicity $\rightarrow m_1 = 2$.

$\therefore$ General solution is $t_i = c_1 4^i + c_2 i 4^i$

Now $T(2^i) = t_i$

$\therefore$ When $n = 2^i$, $T(n) = t_{\log n}$

$\therefore T(n) = c_1 n^2 + c_2 n^2 \log n$ — (i)

Subs. (i) in original recurrence,

∴ General solution is:

$$t_i = c_1 2^i + c_2 i 2^i + c_3 i^2 2^i$$

Now when $i = \log n$, we get

$$T(n) = c_1 n + c_2 n\log n + c_3 n\log^2 n$$

Subs. this in original recurrence gives

$$n\log n = T(n) - 2T(n/2)$$
$$= c_1 n + c_2 n\log n + c_3 n\log^2 n - 2\left[c_1(n/2) + c_2(n/2)\log(n/2) + c_3(n/2)\log^2(n/2)\right]$$

$$n\log n = (c_2 - c_3)n + 2c_3 n\log n$$

Comparing coefficients, $2c_3 = 1$

$$\therefore \boxed{c_3 = 1/2}.$$

Also $c_2 = c_3$

$$\therefore \boxed{c_2 = 1/2}.$$

$\therefore T(n) \in \Theta(n\log^2 n \mid n \text{ is power of } 2)$,

regardless of initial conditions.

$$n^2 = T(n) - 4T(n/2)$$
$$= c_1 n^2 + c_2 n^2 \log n - 4\left(c_1 (n/2)^2 + c_2 (n/2)^2 \log(n/2)\right)$$
$$= \cancel{c_1 n^2} + c_2 n^2 \log n - \cancel{c_1 n^2} - c_2 n^2 \log\left(\frac{n}{2}\right)$$
$$= c_2 n^2 \left(\log n - \log\left(\frac{n}{2}\right)\right)$$
$$= c_2 n^2 \left(\log \frac{n}{n/2}\right)$$
$$n^2 = c_2 n^2$$

Comparing coefficients, $c_2 = 1$

$\therefore T(n) \in O(n^2 \log n \mid n \text{ is power of } 2)$,

Regardless of initial conditions

EG 3 Consider recurrence

$$T(n) = 2T(n/2) + n \log n \quad \text{when } n \text{ is power of } 2, n \geq 2.$$

Let $t_i = T(2^i) = 2T(2^{i-1}) + i2^i$
$= 2t_{i-1} + i2^i$

Rewrite as $t_i - 2t_{i-1} = i2^i$

$\therefore$ Here $b = 2 \quad d = 1$

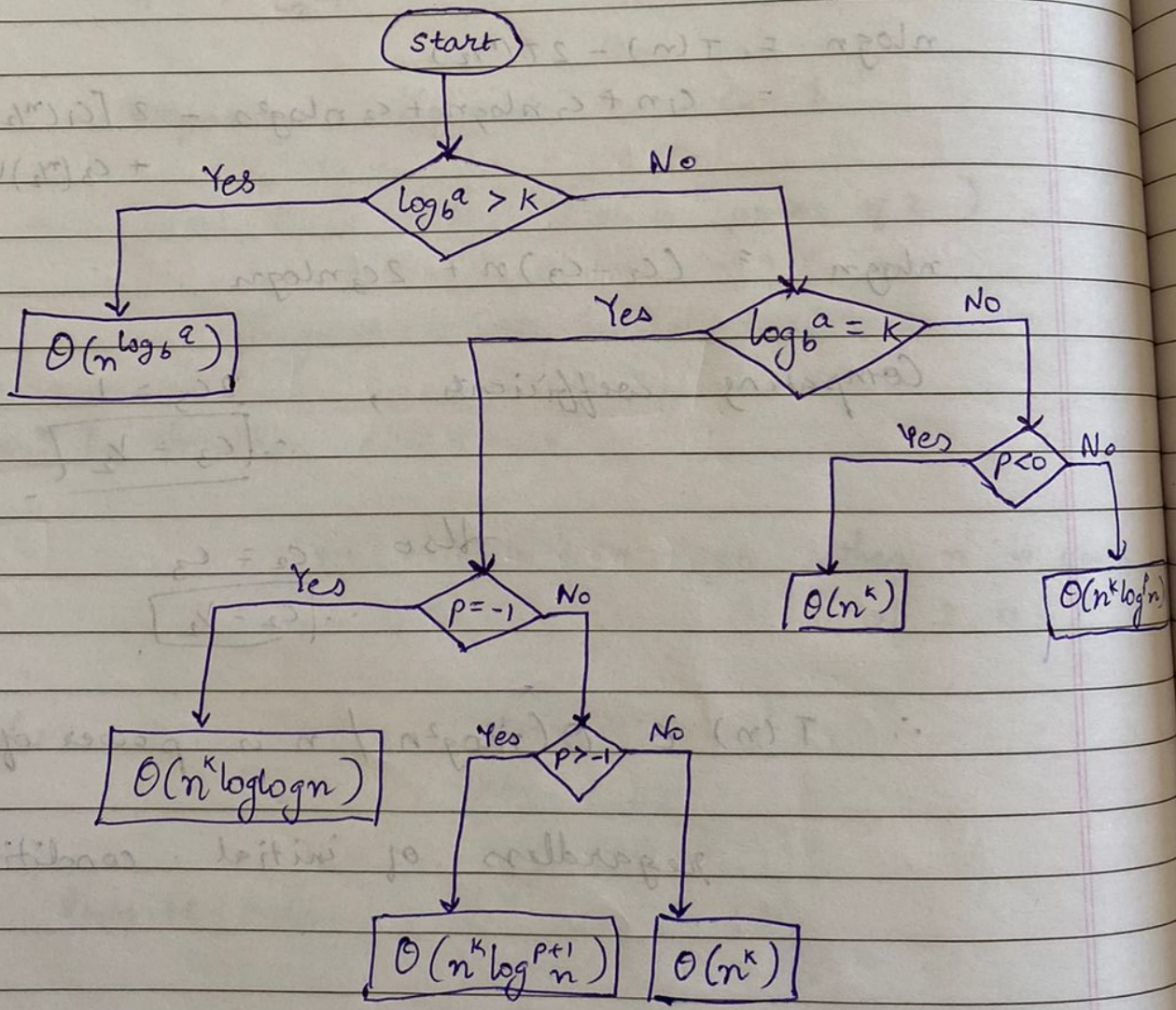
$\therefore$ Characteristic polynomials is $(x-2)(x-2)^2$
$= (x-2)^3$

METHOD 4 "Master method"

$T(n) = aT(n/b) + f(n)$ where $f(n) = \Theta(n^k \log^p n)$

Constraints :- $a \geq 1$ $b > 1$

(i) Calculate $\log_b a$ and k



Page No.
Date: / /

Ex1 $T(n) = 9T(n/3) + n$

Here $a = 9$, $b = 3$, $f(n) = n$, $\log_b a = \log_3 9 = 2$

$k = 1$

$\therefore \log_b a > k$

$\therefore$ Case (I) applies $\Rightarrow$ $\boxed{\Theta(n^2)}$

Ex2 $T(n) = T(2n/3) + 1$

Here $a = 1$, $b = 3/2$, $f(n) = 1$, $\log_b a = \log_{3/2} 1 = 0$

$k = 0$

Here $\log_b a = k$

$\therefore$ Case (II) applies. Also $p > -1$

$\therefore \boxed{T(n) = \Theta(\log n)}$

Ex3 $T(n) = 3T(n/4) + n\log n$

Here $a = 3$, $b = 4$, $f(n) = n\log n$, $k = 1$, $p = 1$, $\log_b a = \log_4 3$

Here $\log_b a < k$

$\therefore$ Case (III) applies. Also $p > 0$

$\therefore \boxed{T(n) = \Theta(n\log n)}$

Ex 7 $T(n) = 7T(n/2) + \Theta(n^2)$

Here $a = 7$, $b = 2$, $k = 2$, $p = 0$

$\log_b a = \log_2 7$

Now $\log_b a > k$

$\therefore$ Case (I) applies $\Rightarrow$ $\boxed{T(n) = \Theta(n^{\log 7})}$

METHOD 5 "Range Transformations"

Ex 1 Consider following recurrence :-

$$T(n) = \begin{cases} 1/3 & \text{if } n = 1 \\ n\,T^2(n/2) & \text{otherwise.} \end{cases}$$

Let $t_i = T(2^i)$

$\therefore t_i = T(2^i) = 2^i T^2(2^{i-1})$

$= 2^i t_{i-1}^2$

Now $u_i = \log_b i = i + 2\log t_{i-1}$

$= i + 2u_{i-1}$

Now rewrite as $u_i - 2u_{i-1} = i$

$\therefore$ Characteristic polynomial is

$(x-2)(x-1)^2$ $(\because b = 1, d = 1)$.

Ex 4 $T(n) = 2T(n/2) + n\log n$

Master method cannot be applied.

Ex 5 $T(n) = 2T(n/2) + \Theta(n)$

Here $a = 2$, $b = 2$, $k = 1$, $p = 0$

$\log_b a = \log_2 2 = 1$

Now $\log_b a = k$

$\therefore$ Case (II) applies. Further $p > -1$

$\therefore$ $\boxed{T(n) = \Theta(n\log n)}$

Ex 6 $T(n) = 8T(n/2) + \Theta(n^2)$

Here $a = 8$, $b = 2$, $k = 2$, $p = 0$

$\log_b a = \log_2 8 = 3$

Now $\log_b a > k$

$\therefore$ Case (I) applies $\Rightarrow$ $\boxed{T(n) = \Theta(n^3)}$

Now determine c_1 using initial condition

$$T(1) = 1/3$$

$$\therefore \frac{2^{c_1}}{4} = \frac{1}{3}$$

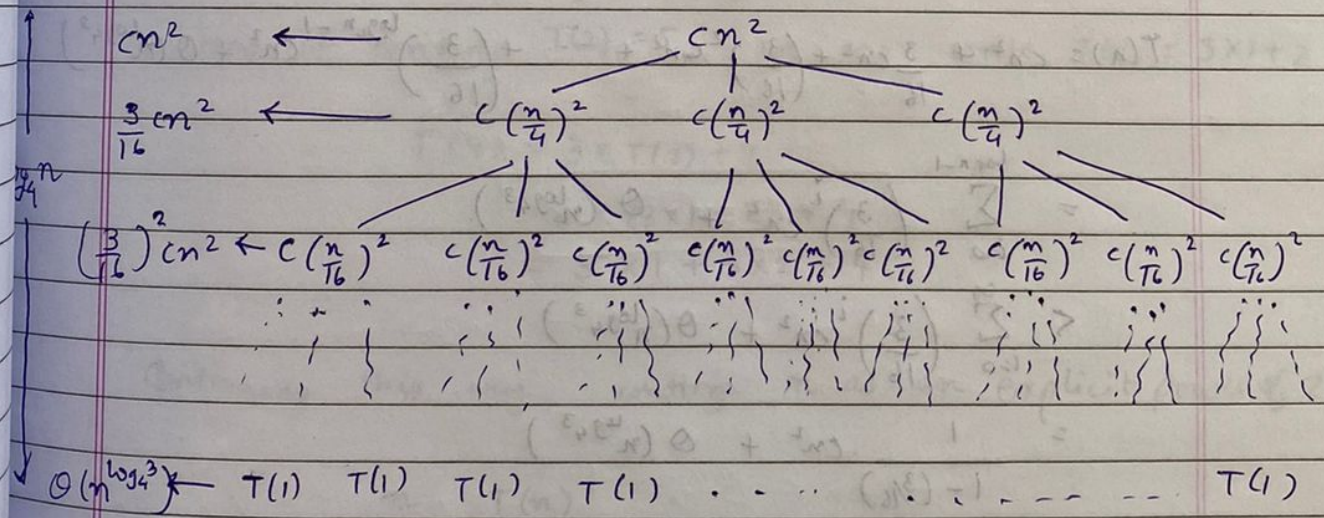
$$\Rightarrow c_1 = \log\left(\frac{4}{3}\right)$$

$$\Rightarrow c_1 = 2 - \log 3$$

$\therefore$ Final solution, $\boxed{T(n) = \frac{2^{2n}}{4n3^n}}$

METHOD 6 "Recursion tree method"

EX1 $T(n) = 3T(n/4) + cn^2$



→ Subproblem size decreases by factor of 4

→ Subproblem size for node at depth $i = \frac{n}{4^i}$

→ $n/4^i = 1$ means sub problem size has reached 1.

⇒ $\log_4 n$ is depth of tree

General solution is

$$u_i = c_1 2^i + c_2 1^i + c_3 i 1^i$$

Substituting this, we get

$$i = u_i - 2u_{i-1}$$

$$= c_1 2^i + c_2 + c_3 i - 2(c_1 2^{i-1} + c_2 + c_3 (i-1))$$

$$= (2c_3 - c_2) - c_3 i$$

Comparing coefficients,

$$\therefore \underline{c_3 = -1}. \qquad \underline{c_2 = 2c_3 = -2}.$$

$\therefore$ General solution is $u_i = c_1 2^i - i - 2$

$\therefore$ General solution for t_i :-

$$t_i = 2^{u_i}$$
$$= 2^{c_1 2^i - i - 2}$$

$\therefore$ General solution for $T(n)$:-

$$T(n) = t_{\log n} = 2^{c_1 n - \log n - 2}$$
$$= \frac{2^{c_1 n}}{4n}$$

METHOD 7 "Intelligent guess work"

EX 1 $$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

First, tabulating values for few powers of 2

n	$T(n)$
1	1
2	5
4	19
8	65
16	211
32	665

Instead of writing $T(2) = 5$, we can write $T(2) = 3 \times 1 + 2$

$$T(4) = 3 \times T(2) + 4$$
$$= 3 \times (3 \times 1 + 2) + 4$$
$$= 3^2 \times 1 + 3 \times 2 + 4$$

Continuing this way, writing n as an explicit power of 2.

n	$T(n)$
1	1
2	$3 \times 1 + 2$
2^2	$3^2 \times 1 + 3 \times 2 + 2^2$
2^3	$3^3 \times 1 + 3^2 \times 2 + 3 \times 2^2 + 2^3$
2^4	$3^4 \times 1 + 3^3 \times 2 + 3^2 \times 2^2 + 3 \times 2^3 + 2^4$
2^5	$3^5 \times 1 + 3^4 \times 2 + 3^3 \times 2^2 + 3^2 \times 2^3 + 3 \times 2^4 + 2^5$

→ Each level has 3X nodes than above one.

∴ Nodes at depth "i" → 3^i

Each node at depth i has cost $c\left(\frac{n}{4^i}\right)^2$

∴ Total cost over all nodes at depth "i" →

$$3^i \times c \times \left(\frac{n}{4^i}\right)^2$$

$$= cn^2\left(\frac{3}{16}\right)^i$$

→ Bottom level (at $i = \log_4 n$) has $3^i = 3^{\log_4 n}$

$$= n^{\log_4 3} \text{ nodes}$$

each contributing cost $T(1)$, for total cost $n^{\log_4 3} \times T(1)$ which is $\Theta(n^{\log_4 3})$.

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{1}{1-(3/16)}\, cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13}\, cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$

Pattern is obvious now

$$T(2^k) = 3^k 2^0 + 3^{k-1} 2^1 + \cdots + 3^0 2^k$$

$$= \sum_{i=0}^{k} 3^{k-i} 2^i$$

$$= 3^k \sum_{i=0}^{k} \left(\frac{2}{3}\right)^i$$

$$= 3^k \left(\frac{1 - (2/3)^{k+1}}{1 - 2/3} \right)$$

$$= 3^{k+1} - 2^{k+1}$$

Let $n = 2^k$

$\Rightarrow k = \log n$

$$\therefore T(n) = T(2^{\log n})$$

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

Using fact that $3^{\log n} = n^{\log 3}$,

$$\boxed{T(n) = 3n^{\log 3} - 2n}$$

Using conditional asymptotic notation, we conclude that

$$T(n) \in O(n^{\log 3} \mid n \text{ is powers of } 2).$$

Check Tut 3 for more practice probs

Insertion in heap

→ Insert at leftmost empty position.
→ Swap the new value with its parent until its parent is smaller than itself.

Heapify

Method heapify makes binary tree rooted at i a heap by moving A[i] down the heap.

Heap of n nodes has height O(logn)

R.T. analysis :- Insert

→ We might have to move element way all the way to the top.
→ Hence atmost O(logn) steps are required

Heapify

→ Element might be moved all way to last level
→ Hence requires O(logn) time.

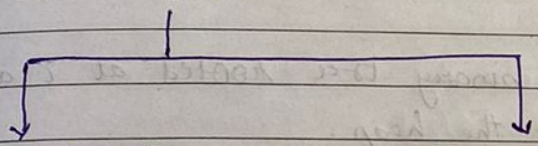
Del min in heap

→ Replace top element with last element of heap.
→ Then heapify (1).

Binary Heap

* Binary Tree → any node can have max 2 children
* Strictly binary tree → Binary tree whose every node has 0 or 2 children (Full binary tree)
* Complete binary tree → Binary tree in which every level except possibly last is completely filled and all nodes are as left as possible.

Binary heaps → B.T. that stores priorities pairs at nodes



Structural property	Heap property
All levels except last are full. Last level is left filled	Priority of node is at least as large as that of its parent.

Find min element

→ Element with smallest priority is always at root of heap.
→ Hence min element can be found in O(1) time.

Height of heap

→ Suppose heap of n nodes has height h ∴ $\boxed{h = \lfloor \log_2 n \rfloor}$

Heap property → $A[Parent(i)] <= A[i]$
Children links → Children of node i are $2i$ and $2i+1$

Heap-sort

→ Create heap
→ Do del min repeatedly until heap becomes empty
→ To do an in place sort, we move deleted element to end of heap.

```
HEAPSORT(A)
    BUILD MAX-HEAP(A)
    for i = A.length downto 2
        exchange A[1] with A[i]
        A.heap-size = A.heap-size - 1
        MAX-HEAPIFY(A, 1)

MAX-HEAPIFY(A, i)
    l = LEFT(i)
    r = RIGHT(i)
    if l ≤ A.heap-size and A[l] > A[i]
        largest = l
    else largest = i
    if r ≤ A.heap-size and A[r] > A[largest]
        largest = r
    if largest ≠ i
        exchange A[i] with A[largest]
        MAX-HEAPIFY(A, largest)
```

Running time of heap operations :-

Insert	$O(\log n)$	Delete min	$O(\log n)$
Heapify	$O(\log n)$	Build heap	$O(n)$
Find min	$O(1)$	Heap sort	$O(n \log n)$

Building heap

→ We start from bottom and move up
→ All leaves are heaps to begin with

R.T : $\lfloor \frac{n}{2} \rfloor$ calls to heapify = $n\,O(\log n) = O(n \log n)$

We ~~cap~~ can provide better $O(n)$ bound :-

→ Our tighter analysis relies on properties that an n-element heap has height $\lfloor \log n \rfloor$ and atmost $\lceil \frac{n}{2^{h+1}} \rceil$ nodes of any height h.

→ Time req. by heapify when called on node of height h is $O(h)$ and so we can express total cost of BUILD-HEAP by

Total num of swaps required :-

$$\sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h}\right) \rightarrow \sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = 2$$

$$\left(\because \sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2} \text{ for } |x|<1 \right)$$

$$\therefore O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h}\right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right)$$

$$= O(n)$$

Merging two binary heaps → Just put two arrays together and create new heap out of them which takes $O(n)$.

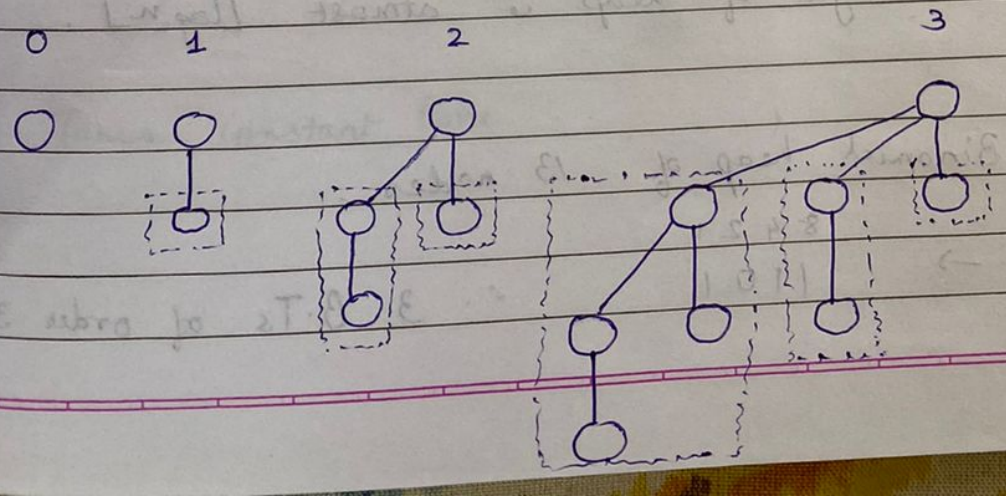
R.T comparison.

Operation	Binary	Binomial	Fibonacci
find-min	$O(1)$	$O(\log n)$	$O(1)$
delete-min	$O(\log n)$	$O(\log n)$	$O(\log n)$
insert	$O(\log n)$	$O(1)$	$O(1)$
decrease-key	$O(\log n)$	$O(\log n)$	$O(1)$
merge	$O(n)$	$O(\log n)$	$O(1)$

BINOMIAL HEAPS

Binomial tree.

→ Binomial tree of order 0 is single node.
→ Binomial tree of order k (B_k) has root node whose children are roots of binomial trees of orders k-1, k-2, ..., 2, 1, 0



Finding mean.

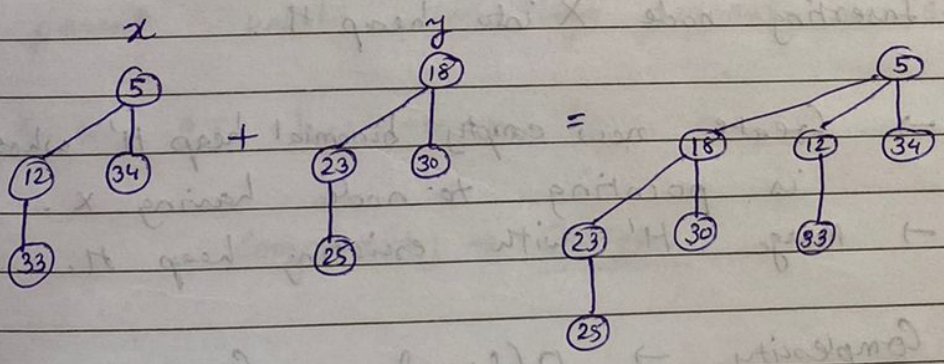
→ Root of min heap B·T contains min data
→ If there are m trees ⇒ find min of m items
→ If n is total nodes in binomial heap, there are atmost $1 + \lfloor \log n \rfloor$ BTs.

∴ R·T. → $\boxed{O(\log n)}$

Merging

→ Repeatedly merge 2 B·T·s of same degree until B·T·s have a unique degree.
→ To merge 2 BTs of same degree, do :-

* Compare roots of trees (x & y). Find smallest tree. Let x is tree with smallest root.
* Make x's root parent of y's root.



→ Takes constant time.

B.T → binomial tree

Properties:

Binomial tree of order k has

(i) 2^k nodes
(ii) height k
(iii) ${}^kC_d$ nodes at depth d
(iv) degree of root node in BT is k.
(v) deleting root yields k binomial trees: $B_{k-1}, B_{k-2}, \ldots, B_0$

* Binomial heap is collection of binomial trees such that:

⊙ Each B.T in heap obeys min heap property
⊙ There can be atmost one B.T. for each order

→ Smallest key in entire heap is one of the roots.

→ Binomial heap with n nodes consists of atmost $1 + \lfloor \log n \rfloor$ binomial trees.

→ Height of heap is atmost $\lfloor \log_2 n \rfloor$.

Binomial heap of 13 nodes.

8 4 2 1
→ 1 1 0 1 ∴ 3 B.Ts of order 3, 2 & 0.

Delete min.

→ Search thru roots and find root with smallest key & call it x. Remove x from tree
→ Create new empty heap H'
→ Reverse order of x's children and set head of H's to point to head of resulting list.
→ Merge H' with H.

Complexity → O(log n).

Decrease Key.

→ After decreasing key, it may become smaller than key of its parent resulting in violation of min heap.

→ If this is case, exchange element with its parent until min heap property is satisfied.

→ Each BT has atmost $\log_2 n$ height.

∴ Complexity → O(log n).

Delete

→ To delete, decrease its key to $-\infty$ & del min key in heap.

Steps for merging :-

(i) Merge two binomial heaps without worrying about trees with same degree i.e. put trees in increasing order of degree.

(ii) Starting from head, repeatedly merge trees with same degree until all trees in heap have unique degree.

Complexity of merge → $O(\log n)$.

At most $\lfloor \log n_1 + \log n_2 \rfloor + 2$ number of binomial trees we traverse roots of these trees constant number of times. That gives complexity of $O(\log n)$.

Insert a node.

Inserting node X into heap H.

→ Create new empty binomial heap H' whose head is pointing to node having x.
→ merge H' with existing heap H.

Complexity → $O(\log n)$. Creating new heap takes only constant time & merging two heaps takes $O(\log n)$.

Divide & Conquer

In DnC, prob is solved recursively, applying three steps at each level of recursion:-

→ "Divide" prob into num of subproblems
→ "Conquer" subprobs by solving them recursively. If subprobs size are small enough, solve them in straightforward manner.
→ Combine sol of subprobs into sol for original prob.

General template

function DC(x)
    if x is sufficiently small or simple then return adhoc(x)
    decompose x into smaller instances $x_1, x_2, \ldots, x_l$
    for i ← 1 to l do $y_i \leftarrow DC(x_i)$
    recombine the y_i's to obtain solution y for x
    return y.

→ Let $t(m)$ be time to call binrec $(T[i..j], x)$, where $m = j - i + 1$.

→ When $m > 1$, algo takes constant amount of time in addition to one recursive call on $\lceil m/2 \rceil$ or $\lfloor m/2 \rfloor$, depending on whether or not $x \in T[k]$.

$\therefore$ $t(m) = t(m/2) + g(m)$, where $g(m) \in O(1)$
$= O(m^0)$

Now using master method,

a	$= 1$	k	$= 0$
b	$= 2$	p	$= 0$
$f(n)$	$= 1$	$\log_b a$	$= \log_2 1$
			$= 0$

Here $k = \log_b a$ $\therefore$ Case (II) applies
Also $p > -1$

$\therefore$ $T(m) = \Theta(m^k \log^{p+1} m)$

$\therefore$ $\boxed{T(m) = \Theta(\log m)}$

Sequential search (Not a dnc algo)

```
function sequential(T[1...n], x)
    {Sequential search for x in array T}
    for i ← 1 to n do
        if T[i] ≥ x then return i
    return n+1
```

→ Takes $\Theta(r)$ time where r is index returned

→ Takes $\Omega(n)$ in worst case and $O(1)$ in best case
On avg, takes time in $\Theta(n)$.

Binary Search

```
function binsearch(T[1..n], x)
    if n=0 or x > T[n] then return n+1
    else return binrec(T[1..n], x)

function binrec(T[i..j], x)
    if i=j then return i
    k ← (i+j) ÷ 2
    if x ≤ T[k] then return binrec(T[i..k], x)
                else return binrec(T[k+1..j], x)
```

Quick sort.

Divide : Partition arr $A[p \ldots r]$ into two subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ such that each element of $A[p \ldots q-1]$ is less than or equal to $A[q]$.
Compute q as part of partitioning procedure.

Conquer : Sort two subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ by ~~recursively~~ recursive calls to quicksort.

Combine : Because subarrays are already sorted, no work is needed to combine them: the entire array $A[p \ldots r]$ is now sorted.

```
QUICKSORT (A, p, r)
  if p < r
    q = PARTITION (A, p, r)
    QUICKSORT (A, p, q-1)
    QUICKSORT (A, q+1, r)
```

To sort entire array A, initial call is QUICKSORT (A, 1, A.length).

```
PARTITION (A, p, r)
  x = A[r]
  i = p - 1
  for j = p to r-1
    if A[j] ≤ x
      i = i+1
      exchange A[i] with A[j]
  exchange A[i+1] with A[r]
  return i+1
```

→ Running time of PARTITION on subarray $A[p..r]$ is $\Theta(n)$ where $n = r - p + 1$

Worst case

→ When array is sorted, its worst. Two subproblems with $n-1$ & 0 elements are formed.

→ Partitioning costs $\Theta(n)$ time.

→ Since recursive call on array of size 0 just returns, $T(0) = \Theta(1)$, recurrence for running time is

$$T(n) = T(n-1) + T(0) + \Theta(n)$$
$$= T(n-1) + \Theta(n)$$

Solving this inhomogenous recurrences gives,

$\boxed{T(n) = \Theta(n^2)}$ ← worst case T.C.

Best case

→ When most even possible split is done. Two subproblems with $\lfloor n/2 \rfloor$ & $\lceil n/2 \rceil - 1$ elements are formed

→ Recurrence for R.T. is $T(n) = 2T(n/2) + \Theta(n)$

Using master method, $\boxed{T(n) = \Theta(n \log n)}$ ← Best case T.C.

$T(n) = T(\quad) + T(0.7n) + O(n)$

$\Rightarrow T(n) \leq T(\quad) + T(0.7n) + cn$ — ①

How to find good pivot?

A (n): | 8 5 11 21 3 | 18 26 12 6 14 | ... | ... | |

B1 B2 B3 ... $B_{n/5}$

$m1 = 8$ $m2 = 12$ $m3$ ... $m_{n/5}$

$\rightarrow M = \{m1, m2, \ldots, m_{n/5}\}$

$n \rightarrow n/5$

$O(n)$

$O(1) + O(n)$

$= O(5) \Rightarrow O(n)$

my claim: x is median of M

1) How to prove claim is correct?
2) If claim is correct, what is T.C to compute x?

Ans 2) $x = \text{select}(M, \frac{|M|}{2})$ → $T(n/5)$

$T(n) \leq T\left(\frac{n}{5}\right) + T(0.7n) + cn$ — ②.

Ans 1) If $m_i \leq x$, it means i^{th} block contributes atleast 3 elements to A1

otherwise i^{th} block contributes atleast 3 elements to A2

$A1 \leftarrow 3 \times \frac{n}{10} = \frac{3n}{10}$

$A2 \leftarrow 3 \times \frac{n}{10} = \frac{3n}{10}$

A1 will have elements $\geq 0.3n$

A2 will have ele. $\geq 0.3n$.

So if $A1 \geq 0.3n \Rightarrow A2 \leq 0.7n$

& if $A2 \geq 0.3n \Rightarrow A1 \leq 0.7n$.

Finding Median.

Naive : - Sort the array ← $O(n \log n)$
- Pick $\lceil \frac{n}{2} \rceil$ element ← $O(1)$
So overall → $O(n \log n)$

Can we do better ?

Can we find median from an array of size n in linear time ? → Yes.

Selection Problem :

Input : an array of size $n \rightarrow A[1 \ldots n]$ & $k \in [1, n]$
Output : Return k^{th} smallest value

Select (A, k)
- Find a "good" pivot $x \in A$, such that A can be partitioned into $A_1 = \{A[i] \leq x\}$ & $A_2 = \{A[i] > x\}$

$|A_1|$ & $|A_2| \leq 0.7n$

- if $k \leq |A1|$
  Select (A1, K)
  else
  select (A2, K-|A1|)
- report

For block size 7

$$T(n) = T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7}\right) + cn \leftarrow O(n)$$

If $m_i \leq x$

$$A_1 \leftarrow 4\frac{n}{7}\frac{1}{2} = \frac{2n}{7}$$

$$A_2 \leftarrow 4\frac{n}{7}\frac{1}{2} = \frac{2n}{7}$$

If $A_1 = \frac{2n}{7} \Rightarrow A_2 = n - \frac{2n}{7} = \frac{5n}{7}$

If $A_2 = \frac{2n}{7} \Rightarrow A_1 = n - \frac{2n}{7} = \frac{5n}{7}$

For block size 9

$$T(n) = T\left(\frac{n}{9}\right) + \quad + cn$$

If $m_i \leq x$

$$A_1 \leftarrow 5 \times \frac{n}{9} \times \frac{1}{2} = \frac{5n}{18}$$

$$A_2 \leftarrow 5 \times \frac{n}{9} \times \frac{1}{2} = \frac{5n}{18}$$

If $A_1 = \frac{5n}{18} \Rightarrow n - \frac{5n}{18} = \frac{13n}{18} \approx 0.72$

$A_2 = \frac{5n}{18} \Rightarrow n - \frac{5n}{18} = \frac{13n}{18} \approx 0.72$

For block size 3

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$

If $m_i \leq x$

$$A_1 \leftarrow 2 \times \frac{n}{3} \times \frac{1}{2} = \frac{n}{3} \Rightarrow A_2 = \frac{2n}{3}$$

$$A_2 \leftarrow 2 \times \frac{n}{3} \times \frac{1}{2} = \frac{n}{3} \Rightarrow A_2 = \frac{2n}{3}$$

So we have proved that x is a good pivot.

$$T(n) \leq T\left(\frac{n}{5}\right) + T(0.7n) + cn$$

We will get $T(n)$ using sub method.

(1) Make a guess for $T(n)$
(2) P.T. guess is correct (mathematical induction)

Lets assume $T(n) = O(n) \Rightarrow T(n) \leq Bn$

Now lets assume our guess is true for $n = 1$ to $k-1$

$\therefore T(1) \leq B$
$T(2) \leq 2B$
$\vdots$
$T(k-1) \leq (k-1)B$

Now we need to prove that $P(n)$ or $T(n)$ is true for $n = k$

$$\begin{aligned} RHS &= T\left(\tfrac{k}{5}\right) + T(0.7k) + ck \\ &= T(k/5) + T(0.7k) + ck \\ &\leq 0.2kB + 0.7kB + ck \\ &= 0.9kB + ck \\ &= (0.9B + c)k \\ &= (0.9 \times 10c + c)k \quad (\text{Assume } B = 10c) \\ &= 10ck \\ &= Bk \end{aligned}$$

$\therefore RHS \leq Bk$
$\Rightarrow LHS \leq Bk$
$\Rightarrow T(k) \leq Bk$
$\Rightarrow T(n) \leq Bn$
$\therefore \underline{T(n) = O(n)}$

Guess : $T(n) = O(n) \Rightarrow T(n) \leq Bn$

We assume $T(1)$ to $T(k-1)$ to be true.
That is
$T(1) \leq B$
$T(2) \leq 2b \ldots T(k-1) \leq (k-1)B$ is true.

$$RHS = T(k/3) + T(2k/3) + ck$$
$$\leq \tfrac{1}{3}kB + \tfrac{2}{3}kB + ck$$
$$= kB + ck$$
$$\Rightarrow RHS \leq kB$$
$$RHS \nleq kB$$
$\Rightarrow T(k)$ is false
$\Rightarrow$ Our ~~guess~~ guess $T(n) = O(n)$ is false.

Multiplying Large Integers :

→ Assumption : $5 \times 3 \rightarrow O(1)$

→ Input :

MSD ↓ LSD →

$A : (a_{n-1}, a_{n-2}, \ldots, a_2, a_1, a_0)$
$B : (b_{n-1}, b_{n-2}, \ldots, b_2, b_1, b_0)$

→ Output :

$C = AB$ where $C = (c_n, c_{n-1}, \ldots, c_2, c_1, c_0)$

→ Basic / Naive algo : $A \times B = O(n^2)$
$(n) \quad (n)$

- Can we do better ? Yes by DnC

$A : (a_{n-1}, a_{n-2}, \cdots, a_{n/2}, a_{n/2-1}, \cdots, a_2, a_1, a_0)$

$A = (a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_{n/2}x^{n/2}) + (a_{n/2-1}x^{n/2-1} + \cdots + a_2x^2 + a_1x^1 + a_0x^0)$

$A = x^{n/2}(a_{n-1}x^{n/2-1} + a_{n-2}x^{n/2-2} + \cdots + a_{n/2}) + (a_{n/2-1}x^{n/2-1} + \cdots + a_2x^2 + a_1x^1 + a_0x^0)$

$a_{n-1}x^{n/2-1} + a_{n-2}x^{n/2-2} + \cdots + a_{n/2}$: sum = A1

$a_{n/2-1}x^{n/2-1} + \cdots + a_2x^2 + a_1x^1 + a_0x^0$: A2

$\therefore A = (A1)x^{n/2} + A2$

$B = (B1)x^{n/2} + B2$

Now, $(A \cdot B) = (A1\,x^{n/2} + A2)(B1\,x^{n/2} + B2)$

$\Rightarrow C = (A \cdot B) = A_1B_1x^n + A_2B_1x^{n/2} + A_1B_2x^{n/2} + A_2B_2$ — ①

$T(n) = 4T(n/2) + O(n)$

By masters theorem, $a = 4,\ b = 2,\ f(n) = O(n),\ k = 1,\ p = 0$

$\log_b a = 2$

Here $\log_b a > k$ ∴ Case (I) applies ∴ $T(n) = \theta(n^{\log_b a})$

$\boxed{T(n) = \theta(n^2)}$

Notice

$(A_1 + A_2)(B_1 + B_2) = A_1B_1 + (A_1B_2 + A_2B_1) + A_2B_2$

$\Rightarrow A_1B_2 + A_2B_1 = (A_1 + A_2)(B_1 + B_2) - (A_1B_1 + A_2B_2)$ — (2)

Substituting (2) in (1).

$C = A_1B_1 x^n + x^{n/2}[(A_1B_1 + A_2B_2) + (A_1 + A_2)(B_1 + B_2)] + A_2B_2$

(1) (2) (3)

$T(n) = 3T(n/2) + O(n)$ $\Rightarrow a = 3, b = 2, f(n) = O(n)$

$n^{\log_b a} = n^{\log_2 3} = n^{1.59}$ $\Rightarrow$ case 1 applies

$\therefore T(n) = \theta(n^{\log_b a})$

$= \theta(n^{\log_2 3}) = \boxed{O(n^{1.59})}$ better than C.T. $O(n^2)$

Karatsuba's Algorithm

$A \to m$

$B \to n$

Naive $\to O(mn)$

D & C and karatsuba algo

$A \to m$, $B \to n$

$A \to A_1, A_2$ ($m/2$, $m/2$) $\quad B \to B_1, B_2$ ($n/2$, $n/2$)

$T(m, n) = 3T(\frac{mn}{4}) + O(n)$

$p \quad p/4$

$$T(p) = 3T\left(\frac{p}{4}\right) + O(m+n)$$

$$\therefore T(mn) = 3T\left(\frac{mn}{4}\right) + \Theta(mn)$$

$a = 3, \quad b = 4, \quad (mn)^{\log_b a} = (mn)^{\log_4 3}$

$= m^{\log_4 3} n^{\log_4 3}$

$= m^{0.79} n^{0.79}$.

$\therefore \underline{\Theta(m^{\log_4 3} n^{\log_4 3})}$ better that $\Theta(mn)$.

Matrix Chain Multiplication

$A(n \times n)$ → 3×3 × $B(n \times n)$ → 3×3 = $C(n \times n)$ → 3×3 ← C has n^2 elements

Can we do better? Yes → D & C

1-50 | 51-100

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

(2) (1) (2) (1)

$A(n \times n)$ $B(n \times n)$ $C(n \times n)$

Each of these n^2 elements requires n multiplications

$O(n^3)$

$\therefore T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)$

no. of elements in resultant matrix is n^2 & each involves 1 addition which takes $O(1)$ time.

$\therefore O(n^2)$

$a = 8, \quad b = 2, \quad f(n) = O(n^2)$

$n^{\log_b a} = n^{\log_2 8} = n^3$ ⇒ case 1 applies

$T(n) = \Theta(n^{\log_b a})$

$\therefore \underline{T(n) = \Theta(n^3)}$.

Strassen

$p1 = a(f-h)$ $\quad p2 = (a+b)h$

$p3 = (c+d)e$ $\quad p4 = d(g-e)$

$p5 = (a+d)(e+h)$ $\quad p6 = (b-d)(g+h)$

$p7 = (a-c)(e+f)$

The $A \times B$ can be calc. using above seven multiplications following are values of submatrices :-

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} p5+p4-p2+p6 & p1+p2 \\ p3+p4 & p1+p5-p3-p7 \end{bmatrix}$$

from strassens algo,

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$$

here $a = 7, b = 2, f(n) = O(n^2)$

$\Rightarrow n^{\log_b a}$

$= n^{\log_2 7}$

$= n^{2.81}$ $\Rightarrow$ case I applies

$\therefore T(n) = \Theta(n^{\log_b a}) = \boxed{\Theta(n^{2.81})}$

Exponentation : (a^n)

Assume no. of digits in a is m

$12^5 = 12 \times 12 \times 12 \times 12 \times 12$

```
function expo_seq(a, n)
    r = a
    for i = 1 to n-1
        r = r * a
    return r
```

i (at beginning)	digits in a	digits in r (upper bound)	T.C. of r * a assuming simple multiplication	T.C. of r * a assuming karatsuba's algorithm
1	m	m	m^2	$m^{\log_4 3} m^{\log_4 3}$
2	m	$2m$	$2m^2$	$m^{\log_4 3} (2m)^{\log_4 3}$
3	m	$3m$	$3m^2$	$m^{\log_4 3} (3m)^{\log_4 3}$
⋮	⋮	⋮	⋮	⋮
$(n-1)$	m	$(n-1)m$	$(n-1)m^2$	$m^{\log_4 3} ((n-1)m)^{\log_4 3}$

$T(n) \leq m^2 + 2m^2 + 3m^2 + \cdots + (n-1)m^2$

(under classic multiplication) $= m^2 (1 + 2 + 3 + \cdots + (n-1))$

$< m^2 (1 + 2 + 3 + \cdots + n)$

$= m^2 \left(\frac{n}{2} (n+1) \right)$

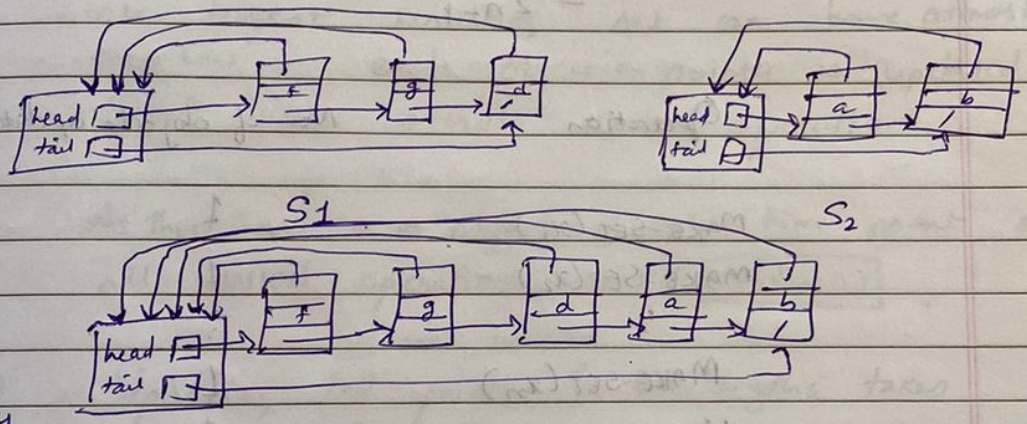
$\Rightarrow T(n) < m^2 \left(\frac{n}{2} (n+1) \right) \Rightarrow \boxed{T(n) \in O(m^2 n^2)}$

→ Representative of resulting set is any member of $S_x \cup S_y$.

FIND-SET(x)

→ Returns pointer to representative of unique set containing x.

Linked List representation



S1

Set object has : head pointing to 1st object in list
tail pointing to last object

Each object in list : set member, pointer to next member & pointer back to set object

Representative → First object in the list.

$T(n) \leq m^{\log_4 3} m^{\log_4 3} + m^{\log_4 3}(2m)^{\log_4 3} + \cdots + m^{\log_4 3}((n-1)m)^{\log_4 3}$

(under Karatsuba's Algo multiplication)

$= m^{\log_4 3} m^{\log_4 3}\left(1 + 2^{\log_4 3} + \cdots + (n-1)^{\log_4 3}\right)$

$< m^{2\log_4 3}\left(1 + 2 + \cdots + (n-1)\right)$

$= m^{\log_2 3}\left(1 + 2 + \cdots + n\right)$

$= m^{\log_2 3}\left(\frac{n}{2}(n+1)\right)$

$\Rightarrow T(n) < m^{\log_2 3}\left(\frac{n^2+n}{2}\right)$

$\Rightarrow \boxed{T(n) \in O(m^{\log_2 3} n^2)}$

DISJOINT SET (Data structure)

→ Maintain a collection of disjoint dynamic sets

$S = \{S_1, S_2, \cdots, S_k\}$

→ Identify each set by representative, which is some member of set.

→ Each element of set is represented by an object.

$x \rightarrow$ object

Operations :- MAKE-SET(x)

→ creates a new set whose only member is x

UNION(x, y)

→ Unites dynamic sets that contains x & y, say S_x and S_y, into a new set that is union of these two sets

Weighted union heuristic

→ Achieve tighter bound on m MAKESET, UNION & FIND SET.

→ We observe that for any $k \leq n$, after x's pointer has been updated $\lceil \log k \rceil$ times, resulting set must have atleast k members.

→ Since largest resulting set can have atmost n members, each object's pointer is updated atmost $\lceil \log n \rceil$ times overall UNION operations.

→ As there are n objects, total time spent over all UNION operations → $O(n \log n)$.

→ Updating tail pointer & list lengths takes $O(1)$ time per UNION op.

Total time spent in all UNION ops is thus $O(n \log n)$

→ Time for entire sequence of m operations follows easily. Each MAKESET & FIND-SET op takes $O(1)$ time & there are $O(m)$ of them

∴ Total time → $O(m + n \log n)$.

MAKE-SET & FIND-SET → O(1) time

Simple representation of union

→ Suppose we have objects $x_1, x_2, \cdots x_n$

↪ Executed n MAKE-SET op^{n}s to create these objects.

→ Followed by n−1 UNION operations.

$\therefore m = n + n - 1$

$= 2n - 1$

Operation	No. of objects updated
MAKE-SET(x_1)	1
MAKE-SET(x_2)	1
⋮	⋮
MAKE-SET(x_n)	1
UNION(x_2, x_1)	1
UNION(x_3, x_2)	2
UNION(x_4, x_3)	3
⋮	⋮
UNION(x_n, x_{n-1})	n−1.

For n MAKE-SET op^{n}s → O(n) time

Total no. of objects updated by all n−1 UNION op^{n}s is

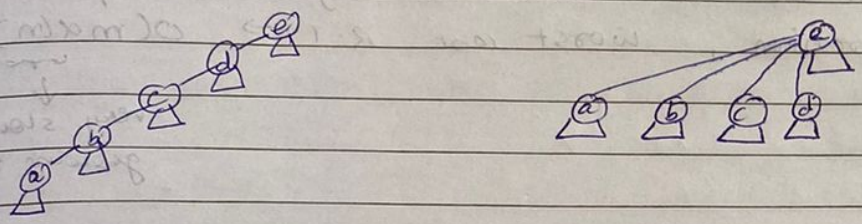
$$\sum_{i=1}^{n-1} i = \Theta(n^2)$$

$\therefore O(n) + \Theta(n^2) = \boxed{\Theta(n^2)}$

Considering m MAKESET & FIND-SET, total $\boxed{O(n+n^2)}$

Amortised Cost of each operation $= \frac{\Theta(n^2)}{2n-1} = \boxed{\Theta(n)}$

On the ~~fight~~ find path point directly to the root. We do not change any ranks.

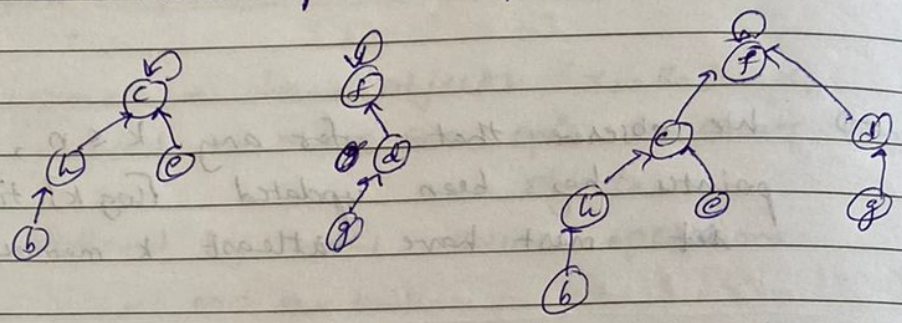


Pseudocode:

```
MAKE-SET(x)
    x.p = x
    x.rank = 0

UNION(x, y)
    LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
    if x.rank > y.rank
        y.p = x
    else x.p = y
        if x.rank == y.rank
            y.rank = y.rank + 1

FIND-SET(x)
    if x ≠ x.p
        x.p = FIND-SET(x.p)
    return x.p
```

Rooted tree representation

→ Each member points its parent. Each tree is a set



→ Root contains representative & is its own parent

→ We achieve asymptotically optimal disjoint-set DS by using "union by rank" & "path compression".

MAKE-SET → constant time
FIND-SET → depends on height of tree or rank of root
UNION → depends on find set

(*) Union by rank

→ Make root of tree with less height point to root of tree with more height.
→ We make root with smaller rank point to root with larger rank during UNION.

rank → Upper bound on height of node

(*) Path compression

→ We use it during FIND-SET operations to make each node

Date: / /

R.T. analysis:

→ When we use both union by rank & path compression, worst case R.T → $O(m\alpha(n))$
where $\alpha(n)$ is very slowly growing function

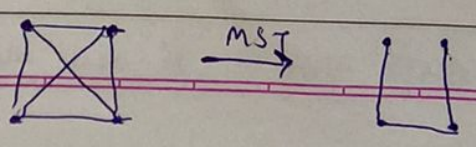
→ In any conceivable application of disjoint-set DS, $\alpha(n) <= 4$; thus R.T is almost linear in m in all practical situations.

GREEDY ALGORITHMS

→ Used to solve optimization problems.
→ Always makes locally optimal choice in the hope that this choice will lead to a globally optimal solution.
→ Do not always obtain optimal solutions but for many problems they do
→ Easy to implement, easy to invent, and when they work efficient

Minimum Spanning Trees

→ Subgraph and more precisely a tree which consists |V| (i.e. all) vertices and |V|-1 edges connecting all vertices of a graph.



KRUSKAL'S ALGORITHM

```
MST-KRUSKAL (G, w)
O(1) ←  A = ϕ
O(V) ←  for each vertex v ∈ G.V
            MAKE-SET (v)
O(ElogE) ← sort edges of G.E into non decreasing order by weight w
        for each edge (u,v) ∈ G.E, taken in non decreasing order by weight
O(E) ←      if FIND-SET (u) ≠ FIND-SET (v)
                A = A ∪ {(u,v)}
                UNION (u,v)
    return A
```

$\therefore O(1) + O(E \log E) + O(V+E) = \boxed{O(E \log E)}$

(As $|E| >= |V|-1$ & therefore $E \log E$ is certainly Asymptotically larger once $|V| >= 6$)

PRIM'S ALGORITHM

```
MST-PRIM (G, w, r)
    for each u ∈ G.V                    ]
        u.key = ∞                       ] → O(V)
        u.π = NIL                       ]
    r.key = 0                           → O(1)
    Q = G.V                             → O(V)  [Build heap]
    while Q ≠ ϕ                         ] → O(VlogV)
        u = EXTRACT-MIN (Q)             [ ∵ extracting minimum from heap takes O(logV)
```

→ Suppose edges of the graph have weights or lengths.

$$\text{weight(tree)} = \sum \text{weight(edges)}$$

Spanning tree with min length or weight is a minimum spanning tree.

→ Find an acyclic subset $T \subseteq E$ that connects all of vertices and whose total weight

$$w(T) = \sum_{(u,v) \in T} w(u,v) \text{ is minimized}$$

* Problem of determing tree T is MST problem
2 algos are :-

→ Kruskal's
→ Prim's

→ Each of them run in time $O(E \log V)$ using ordinary binary heaps.

→ By using Fibonacci heaps, Prims run in $O(E + V \log V)$, which improves over binary heap implementation if $|V|$ is much smaller than $|E|$.

MAKE-SET(x)
→ new set whose only member is x
→ disjoint property

FIND-SET(x)
→ Returns a pointer to representative of unique set containing x.

UNION(x, y)
→ Unites S_x & S_y
→ Representative
→ Disjoint property.

```
for each v ∈ G.adj[u]
    if v ∈ Q and w(u,v) < v.key
        v.π = u
        v.key = w(u,v)
```

→ O(E log V)

[for loop in ~~total iterates for~~ O(E) time & it involves implicit decrease key operation]

Total time → O(V log V + E log V)
= [O(E log V)]

(assymptotically same as our implementation of Kruskal's algorithm).

By using Fibonacci heap, extract-min takes O(log V) amortised time & Decrease-key takes O(1) time.

∴ R.T. of Prims becomes → [O(E + V log V)]

Single source shortest paths - Dijkstra Algo

→ In single-source shortest-paths problem, given a graph G(V, E), we want to find a shortest path from a given source vertex s ∈ V to each vertex v ∈ V.

```
INITIALIZE-SINGLE-SOURCE (G, s)
    for each vertex v ∈ G.V
        v.d = ∞
        v.π = NIL
    s.d = 0
```

T.C → O(V)

```
RELAX(u, v, w)
    if v.d > u.d + w(u,v)
        v.d = u.d + w(u,v)
        v.π = u
```

T.C → $O(1)$

```
DIJKSTRA(G, w, s)                                   →
1   INITIALIZE-SINGLE-SOURCE(G,s)                   → O(V)
2   S = φ                                           → O(1)
3   Q = G.V                                         → O(|V|)
4   while Q ≠ φ                                     → O(|V|+1)
5       u = EXTRACT-MIN(Q)                          → O(|V| log|V|)
6       S = S ∪ {u}                                 → O(|V|)
7       for each vertex v ∈ G.adj[u]                → O(|E| log|V|)
8           RELAX(u, v, w)
```

$O(E+V)\log V$

or

$O(E)\log V$

(i) If we maintain min priority queue by taking advantage of vertices being numbered 1 to $|V|$.
INSERT & DECREASE-KEY takes $O(1)$ time.
EXTRACT-MIN takes $O(V)$ time (∵ we search through entire array)

∴ Total time → $O(V^2+E)$

$= \boxed{O(V^2)}$

DYNAMIC PROGRAMMING

→ Applies when subproblems overlap - that is, when subproblems share subsubproblems.

→ DP solves each subsubproblem just once & saves answer in table, thereby avoiding repeated work.

→ DC is top down method. DP is bottom up.

Matrix Chain Multiplication

→ How we parenthesize a chain of matrices can have dramatic impact on the cost of evaluating product

```
MATRIX-MULTIPLY(A, B)
if A.columns ≠ B.rows
    error "incompatible dimensions"
else let C be a new A.rows x B.columns matrix
    for i = 1 to A.rows
        for j = 1 to B.columns
            c_ij = 0
            for k = 1 to A.columns
                c_ij = c_ij + a_ik · b_kj
    return C
```

→ Time to compute C is dominated by number of scalar multiplications in line 8, which is pqr.

(ii) If we implement min priority queue by binary min heap,

EXTRACT-MIN takes $O(\log V)$ time & there are $|V|$ such ops.
DECREASE_KEY takes $O(\log V)$ time & at most $|E|$ such ops.

Time to build binary min-heap is $O(V)$

$\therefore$ Total R-T $\rightarrow$ $O((V+E)\log V)$

$= \boxed{O(E \log V)}$

(iii) If we implement min priority queue by Fibonacci heap.

EXTRACT-MIN takes $O(\log V)$ time & there are $|V|$ such ops
DECREASE-KEY takes $O(1)$ time & there are $|E|$ such ops

$\therefore$ Total R-T $\rightarrow$ $\boxed{O(V \log V + E)}$.

Problem :- Given a chain $\langle A_1, A_2, \cdots, A_n \rangle$ of n matrices, where $i = 1, 2, \cdots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1, A_2, \cdots A_n$ in a way that minimizes the no. of scalar multiplications.

Recursive definition for the minimum cost of parenthesizing product $A_i A_{i+1} \cdots A_j$ becomes

$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j \end{cases}$$

Define $s[i,j]$ to be value of k at which we split product $A_i A_{i+1} \cdots A_j$ in an optimal parenthesization.

$s[i,j]$ equals value k such that

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$$

$m[i,j]$ values gives costs of optimal solutions to subproblems

```
MATRIX-CHAIN-ORDER(p)
1  n = p.length - 1
2  let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
3  for i = 1 to n
4      m[i, i] = 0
5  for l = 2 to n                    // l is chain length
6      for i = 1 to n - l + 1
7          j = i + l - 1
8          m[i, j] = ∞
9          for k = i to j - 1
```

```
PRINT-OPTIMAL-PARENS (s, i, j)
1  if i == j
2     print "A i"
3  else print "("
4     PRINT-OPTIMAL-PARENS (s, i, s[i,j])
5     PRINT-OPTIMAL-PARENS (s, s[i,j]+1, j)
6     print ")"
```

For PRINT-OPTIMAL-PARENS (s, 1, 6) prints $((A_1(A_2A_3))((A_4A_5)A_6))$.

Longest Common Subsequence

Given two sequences X and Y, we say that a sequence Z is common subsequence of X and Y if Z is a subsequence of both X and Y.

eg. $X = \{A, B, C, B, D, A, B\}$
$Y = \{B, D, C, A, B, A\}$
∴ Sequence $\{B, C, A\}$ is a common subsequence of both X and Y.
But it's not LCS of X and Y, however since it has length 3 and sequence $\{B, C, B, A\}$ which is also common to both X and Y, has length 4.
Sequence $\{B, C, B, A\}$ is an LCS of X & Y, as is the sequence $\{B, D, A, B\}$, since X & Y have no common subsequence of length 5 or greater.

```
10              q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
11              if q < m[i,j]
12                  m[i,j] = q
13                  s[i,j] = k
14      return m and s
```

→ Much more efficient than exponential time method of enumerating all possible parenthesizations and checking each ~~oth~~ one.

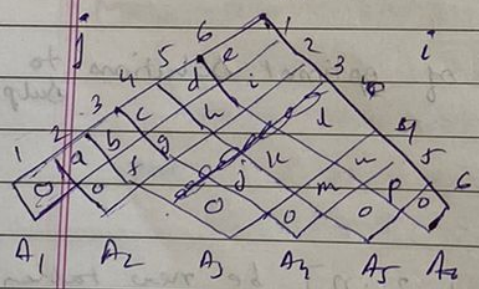
eg:

Matrix	A_1	A_2	A_3	A_4	A_5	A_6
Dimension	30×35	35×15	15×5	5×10	10×20	20×25

$n = 6$.

Computed row-wise from ~~bottom to top~~

m



s.

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 & = 13000 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 & = 7125 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 & = 11375 \end{cases}$$

$= 7125$

$X = \{x_1, x_2, \cdots, x_m\}$
$Y = \{y_1, y_2, \cdots, y_n\}$
Find LCS of X and Y.

→ Let $c[i,j]$ to be length of LCS of sequences x_i & y_j.

→ If $i=0$ or $j=0$, one of sequences has length 0 and so LCS has length 0.

→ Optimal substructure of LCS problem gives recursive formula

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1, j-1]+1 & \text{if } i,j>0 \ \& \ x_i = y_j \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j>0 \ \& \ x_i \neq y_j \end{cases}$$

```
LCS-LENGTH (X,Y)
    m = X.length
    n = Y.length
    let b[1..m, 1..n] and c[0..m, 0..n] be new tables
    for i = 1 to m
        c[i, 0] = 0
    for j = 0 to n
        c[0, j] = 0
    for i = 1 to m
        for j = 1 to n
            if x_i == y_j
                c[i,j] = c[i-1, j-1] + 1
                b[i,j] = "↖"
            elseif c[i-1, j] ≥ c[i, j-1]
                c[i,j] = c[i-1, j]
                b[i,j] = "↑"
```

else $c[i,j] = c[i,j-1]$
$b[i,j] = "\leftarrow"$
return c and b.

R.T → $\boxed{\Theta(mn)}$ since each table entry takes $\Theta(1)$ time to compute

```
PRINT-LCS (b, X, i, j)
  if i == 0 or j == 0
      return
  if b[i, j] == "↖"
      PRINT-LCS (b, X, i-1, j-1)
      print x_i
  elseif b[i, j] == "↑"
      PRINT-LCS (b, X, i-1, j)
  else PRINT-LCS (b, X, i, j-1)
```

Procedure takes time $O(m+n)$ since it decrements at least one of i and j in each recursive call.

Refer one table from DP PPT, slide num 36.

* Principle of Optimality :- It states that - in an optimal sequence of decisions or choices, each subsequence must be optimal.

* Optimal Substructure property :- Problem is said to have optimal substructure if an optimal solution can be constructed from optimal solutions of its subproblems.

Tabulation

* Bottom up approach
* Typical implementation of DP
* Typically iterative
* More time & Space efficient

Memoization

* Top down approach
* Non typical
* Recursive
* Stack overflow

```
fact(n) = n x fact(n-1) ;
```

```
int tab_fact (int n, int dp[])
{
    if(n==0)
      return 1;
    dp[0] = 1;
    for(int i=1; i<=n; i++)
    {
      dp[i] = i * dp[i-1];
    }
    return dp[n];
}
```

```
// all ele of dp[] array are ini. to -1.
int mem_fact (int n, int dp[])
{
    if(n==0)
      return 1;
    if(dp[n] != -1)
    {
      return dp[n];
    }
    dp[n] = n * mem_fact(n-1);
    return dp[n];
}
```

Backtracking

→ Many problems translate to searching for specific node, path or pattern in associated graph.

→ In its basic form, backtracking resembles DFS in a directed graph.

→ Graph is implicit only here

⊙ 0/1 Knapsack Problem:

→ Certain objects & knapsack

→ n types of objects & adequate no. of objects of each type.

eg: 4 types of objects.

w	2	3	4	5
v	3	5	6	10

Knapsack → 8 weight

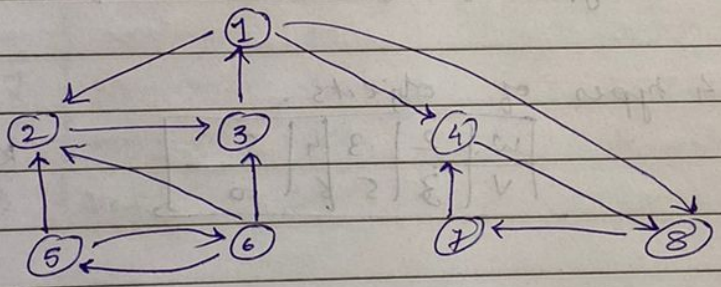
```
function backpack(i, x)
  { calculates value of best load that can
    be constructed using items of types i
    to n and whose total weight
    does not exceed x }
  b ← 0
  { Try each allowed kind of item in turn }
  for K ← i to n do
    if w[K] ≤ x then
      b ← max(b, v[K] + backpack(K, x - w[K]))
  return b
```

To find value of best load, call backpack(1, W)

Graphs

DFS

```
procedure dfssearch(G)
    for each v ∈ N do mark[v] ← not-visited
    for each v ∈ N do
        if mark[v] ≠ visited then dfs(v)

procedure dfs(v)
    {Node v has not previously been visited}
    mark[v] ← visited
    for each node w adjacent to v do
        if mark[w] ≠ visited then dfs(w)
```




```
dfs(1)          initial call
  dfs(2)        recursive call
    dfs(3)      recursive call; progress is blocked
  dfs(4)        neighbour of node 1 hasn't been visited
    dfs(8)      recursive call
      dfs(7)    recursive call; progress is blocked
dfs(5)          new starting point
  dfs(6)        recursive call; progress is blocked.
```

Branch and Bound

⊙ Assignment Problem

Given complete matrix of costs, problem is to assign agents to tasks so as to minimize the total cost of executing the n tasks.

	1	2	3
a	4	7	3
b	2	6	1
c	3	9	4

Hungarian algo for assignment problem.

Eg

		New destinations D	K	M
Current destinations	J	2500	4000	3500
	P	4000	6000	3500
	B	2000	4000	2500

Where would you send each salesperson to minimize air fare?

Cost Matrix :-

2500	4000	3500
4000	6000	3500
2000	4000	2500

(I) Subtract min of every row :

0	1500	1000
500	2500	0
0	2000	500

(II) Subtract min of every column :

0	0	1000
500	1000	0
0	500	500

(III) Cover all zeroes with min. no. of horizontal & vertical lines

~~0~~	[0]	1000
500	1000	[0]
[0]	500	500

Each row & each column has exactly one fixed zero.

(IV) Since we need 3 lines to cover all zeroes, we have found optimal assignment.

2500	[4000]	3500
4000	6000	[3500]
[2000]	4000	2500

∴ Optimal cost is $4000 + 3500 + 2000 = 9500$

eg2 Cost Matrix:

1800	4000	4500
2000	6000	3500
2000	4000	2500

(I) →

0	2200	3000
0	4000	1700
0	2000	500

(II) →

0	500	2500
0	2000	1000
0	0	0

(III) →

[0]	500	2500
~~0~~	2000	1000
~~0~~	[0]	~~0~~

Since we only need 2 lines to cover all zeroes, we have NOT found optimal assignment.

(V) Subtract smallest uncovered entry from all uncovered rows

−500	0	2000
−500	1500	500
0	0	0

(VI) Add smallest entry to all covered columns

0	0	2000
0	1500	500
500	0	0

(VII)

~~0~~	[0]	2000
[0]	1500	500
500	~~0~~	[0]

∴ Optimal cost
= 2000 + 4000
+ 2500
= 8500.

Each row & each column has exactly one 0 ⇒ optimal solution.

Branch and Bound

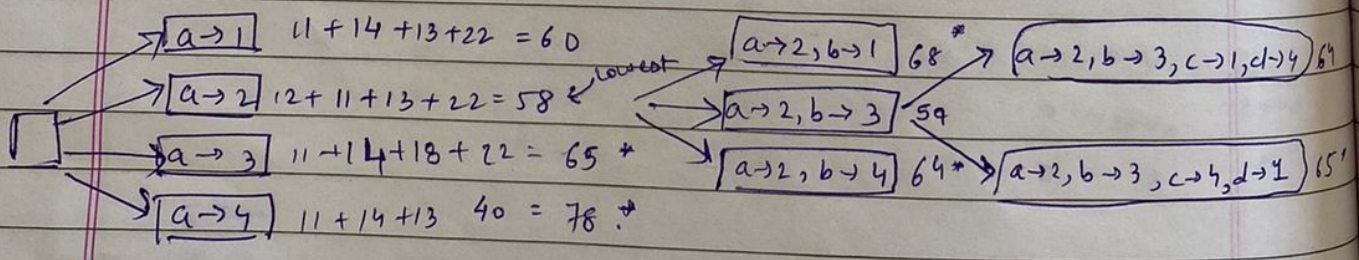
	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Get lower bound by adding smallest elements in each column, $11 + 12 + 13 + 22 = 58$

To obtain upper bound on answer, note that $a \to 1, b \to 2, c \to 3, d \to 4$ is one possible solution whose cost is $11 + 15 + 19 + 28 = 73$.

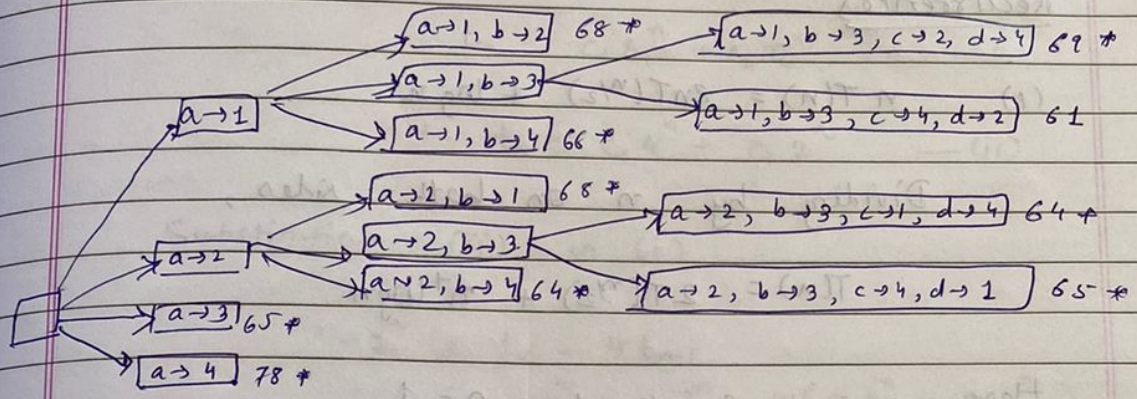
Optimal solution cannot cost more than this.

∴ Answer to our instance lies somewhere in $[58 \ldots 73]$.



* → marking dead

Tree completely explored :



Self Study : Amortized Analysis & Normal level ques.
Randomised Alg.

For Backtracking, Branch & Bound, check tut-9.
For DP, see some extra problems.