DAA

Assignment - 20
Midsem - 30
End sem - 50

Algorithm: A sequence of logically related instructions to solve a computational problem

Input → Output

Characteristics:

I/P, O/P

Finiteness: It must end sometime

Definiteness: Each instruction must be unambiguous ($x=5$ ✓, $y=5*4x$)

Include "Basic arithmetic instructions"

* Many algorithms are possible for a problem if it is solvable

Solvable

How many different algos are possible
How do we compare
How to pick the efficient one

① Correctness - Whether it performs the intended task or not

② Time and Space complexity

③ optimality

A_1 - 10s
A_2 - 20s
A_1 is least

Is there an algo where time < 10s
Can we optimize it?

Algorithm Design

Iterative Algo — Top-down design
Recursive Algo — Bottom up design

Factorial (n)

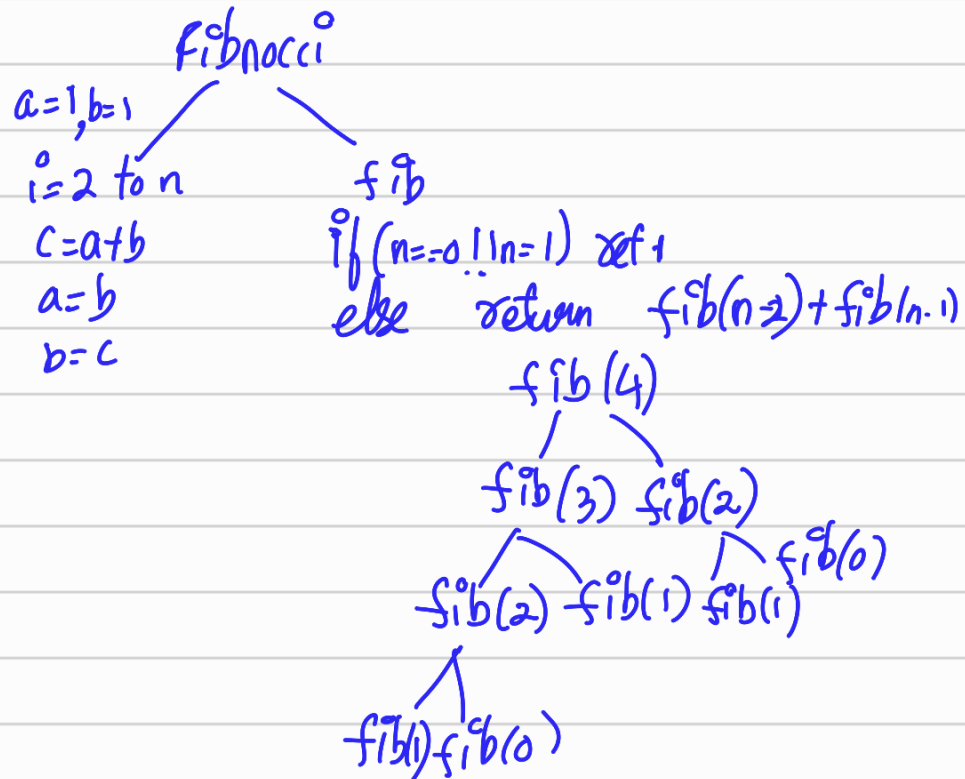
1×2 — $1 \times 2 \times 3 .. n$
n — $n \times (n-1)!$

```
for(i=1; i<=n; i++)
  Iterative
```

```
if(n==1) then return 1
else n*factorial(n-1)
```

Which is efficient?

For every iterative algo, there is an equivalent recursive algorithms



recursive code works more

∴ Iterative method is preferred

Time/Space → locations/bytes used by Algo

↓

System time ??

No

Intel I5

A_1 - Avg (I_1, I_2 .. I_K)

:

A_K - Avg (I_1, I_2 .. I_K)

AMD — A_1 minimum

I5 — A_5 is minimum

(or)

In I5 itself — 10 AM — 15:00

Maybe — 10 ms — 5 ms

So, we can't conclude anything from system time

Step count - The count/frequency of fundamental operations in algorithms.

Find Max

```
Max = A[1]          - Assignment - 1
for i=2 to n        - Assignment, comparision, Increament - (n) (n) (n)
  if (A[i] > max)   - comparision - (n-1)
    max = A[i]      - Assignment - (n-1) (worst case)
return max ;
```

n-2+1+1 = n

$1+n+n+n+n-1+(n-1)+1 = 5n$

↳ Is it fare to add all the step count of assignment comparision ..... ?

Ans: NO, Even step count is not 100% accurate

a=1	- 1	- 1
b=1	- 1	- 1
for i=2 to n	- n	- 2n
c=a+b	- n-1	- 2(n-1) (Addition, assign)
print c	- n-1	- n-1
a=b	- n-1	- n-1
b=c	- n-1	- n-1
	5n-2	8n-3
	✗	✓

```
for i=1 to n          -3(n+1)
{                      i=1      i=2          i=r
   for j=1 to n        (n+1) + (n+1) ... + (n+1) = 3n(n+1)
   {                   i=1,j=1...n  i=2,j=1...n   i=n,j=1..n
      print i,j        n          + n  .... + n = n^2
   }
}
```

$$3(n+1) + 3n(n+1) + n^2$$

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

Recursive

```
fact(int n)                    - n=1
If (n==1) then ret 1           1(if)
else ret n * fact(n-1)         1(ret 1)
                               2
```

$n \geq 2$

(1) If
(1) (multiplication)
(1) (ret)

$T(n-1)$

↓

cost of recursive sub problem of size $(n-1)$

$T(n)$ = cost/step count of recursive subproblem of size n

$$T(n) = \begin{cases} 2 & T(1) \\ 3 + T(n-1) & n \geq 2 \end{cases}$$

$$T(2) = 3 + T(2) = 3 + 3 + T(1) = 3 + 3 + 2 = 8$$

Similarly, Fib $T(n) = \begin{cases} 2 & n=0 \text{ or } 1 \\ 3+T(n-1)+T(n-2) & n \geq 2 \end{cases}$

Order of growth

Step count method $= 3n-1$

n^2+5n-2 (comp, Inc, cost of ret/recursive call)

Ex: $3n^2 + n^3 + n-5$ which operation is frequent
Quad Cubic Linear

frequent
Primitive
dominant
$\} n^3$

Order of growth - Identify the dominant operation

↳ Asymptotic Analysis
- ↳ upper bound (Big-oh 'O')
- ↳ Lower bound (Big Omega Ω)
- ↳ Tight bound (Theta Θ)

Step count $3n^2+5n-2$

$3n^2+5n-2 \leq 5n^2 \quad \forall n \geq 4 \quad (O)$

$3n^2+5n-2 \geq 100n \quad \forall n \geq 100 \quad (\Omega)$

$n^2 \leq 3n^2+5n-2 \leq 5n^2 \quad (\Theta)$

Big-oh 'O'

$f(n), g(n)$ Non-negative
↓ step count

$f(n) = O(g(n))$ iff $\forall n > n_0 \; \exists c > 0 \; (f(n) \leq c \cdot g(n))$

iff $\exists n_0 > 0 \; \exists c > 0 \; (f(n) \leq c\, g(n))$

Ex:

$f(n) = 3n+6 \quad 3n+6 \leq 5n \quad c = 5, \forall n \geq 3, n_0 = 3$

$3n+6 = O(n)$

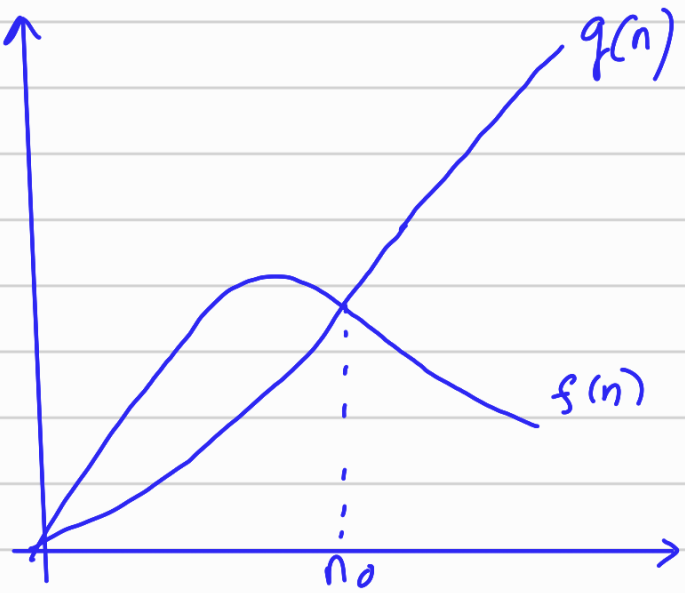
$3n+6 \leq 15n \quad (c = 15, n_0 = 1, \forall n \geq 1)$

↳ $O(n)$

$3n^2+4n-2 \leq (3+\epsilon)n^2$

for some

$\epsilon > 0$ & $n_0 > 0$



$3n^2+4n-2 \neq O(n)$

$\neq O(n^{1.5})$

$\neq O(n^{2-\varepsilon})$

$\varepsilon > 0$

$5n^3+6n^2 \cdot 2^n - 100 \neq 10n^2 \cdot 2^n$ $c=10, n_0=4$ ✗

$\neq O(n^2 \cdot 2^n)$ ✗

$\neq O(2^n)$

$O(n^{2+\varepsilon} \cdot (2+\delta)^n)$ ✓

$\varepsilon \geq 0 \quad \delta \geq 0$

$O(2.001^n)$ ✓

$O(3^n)$ ✓

$n \log n + 10 = o(n \log n)$

$O(n^2)$

$O(n^{1.1})$

$\neq O(n)$

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

Big Omega

$f(n) = \Omega(g(n))$ iff $\exists c>0 \; \exists n_0>0 \; \forall n \geq n_0 \; (f(n) \geq c \cdot g(n))$

$3n-5 \geq 2n$ $\quad \Omega(n)$

$3n-5 \geq 2\log_2 n$ $\quad \Omega(\log_2 n)$

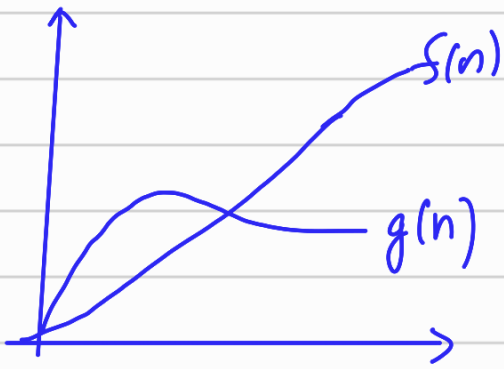
$4n^2-n+2 \geq 2 \cdot n^2$

$\geq 0.01\, n^2$ $\quad \Omega(n^2)$

$\geq 5\sqrt{n}$ $\quad \Omega(\sqrt{n})$

$\geq n^{2-\varepsilon}$ $\quad O(n^{2-\varepsilon})$

$\varepsilon \geq 0$



Tight Bounds (theta Notation)

$f(n)$ $g(n)$

$f(n) = \Theta(g(n))$ iff $\exists c_1>0 \; \exists c_2>0 \; \exists n_0>0 \; \forall n \geq n_0 \; (c_1 g(n) \leq f(n) \leq c_2 g(n))$

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

$0.5n^2 \leq n^2 - 3n + 100 \leq 4n^2$

$\forall n \geq 6 \quad c_1 = 0.5 \quad c_2 = 4$

$n^2 - 3n + 100 = \Theta(n^2)$

$0.5n^2 3^n \leq 2^n + n^{100} + n^2 3^n \leq 6n^2 3^n$

$\Downarrow$

$\Theta(n^2 3^n)$

$3n+5 = O(n)$

$= O(n^2)$

$= O(n^3)$

→ there exists for O

$3n+5 \leq c \cdot n^2 \quad \exists c>0 \quad \exists n_0>0 \quad \forall n \geq n_0$

$3n+5 \leq c \cdot n^2 \quad \forall c>0 \quad \exists n_0>0 \quad \forall n \geq n_0$

↳ But in this condition it works for any c

$3n+5 = O(n)$ → work for some $c>0$

$3n+5 = O(n^{1.1})$ → work for any $c>0$

$O(n^{1+\epsilon}) \quad \epsilon>0$

Little oh

$f(n) = o(g(n))$ iff $\forall c>0 \quad \exists n_0>0 \quad \forall n \geq n_0 \quad (f(n) \leq c \cdot g(n))$

↳ In O notation $\exists c>0$

In o notation $\forall c>0$

$3n+5 = O(n)$ → For some $c>0$, it works

$\neq o(n)$ → For all $c>0$, it won't work

Ex: $c=0.1, 2.9 \ldots$

$3n+5 = o(n^{1.1})$

$= o(n^2)$

$= o(n^{1+\epsilon}) \quad \epsilon>0$

	Big oh	Little oh
n^2+6n+5	$= O(n^2)$	$\neq o(n^2)$
	$= O(n^{2+\epsilon})\ \epsilon \geq 0$	$= o(n^{2+\epsilon}),\ \epsilon > 0$
	$= O(c^n),\ c > 1$	$= o(c^n),\ c > 1$

Little omega

$f(n) = \omega(g(n))$ iff $\forall c > 0\ \exists n_0 > 0\ \forall n \geq n_0\ (f(n) \geq c \cdot g(n))$

$n^2 + 2^n - n = \Omega(n) = \omega(n)$

$= \Omega(n^2) = \omega(n^2)$

$= \Omega(2^n) \neq \omega(2^n)$

↳ If $c = 0.01$ ✓

$c = 10^4$ ✗

$n^3 + 4n^2 - n = \Omega(n^3)$

$= \Omega(n^{3-\epsilon}),\ \epsilon \geq 0$

$= \omega(n^{3-\epsilon}),\ \epsilon > 0$

c : constant

$\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c \quad f(n) = O(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} \geq c \quad f(n) = \Omega(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad f(n) = \Theta(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n))$ Small oh

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

1. Reflexivity

$f(n) = O(f(n))$ $\quad n = \Theta(n)$

$f(n) = \Omega(f(n))$ $\quad n = O(n)$

$f(n) = \Theta(f(n))$ $\quad n = \Omega(n)$

2. Symmetric

$f(n) = O(g(n)) \not\Rightarrow g(n) = O(f(n))$ $\quad n = O(n^2) \not\Rightarrow n^2 = O(n)$

$f(n) = \Omega(g(n)) \not\Rightarrow g(n) = \Omega(f(n))$ $\quad n = \Omega(1) \not\Rightarrow 1 \neq \Omega(n)$

Θ is symmetric

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

3. Transitive

$f(n) = O(g(n)),\ g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$ $\quad n^2 = O(n^4),\ n^4 = O(2^n) \Rightarrow n^2 = O(2^n)$

$f(n) = \Omega(g(n)),\ g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$ $\quad n^2 = \Omega(n \log n),\ n \log n = \Omega(1) \Rightarrow n^2 = \Omega(1)$

$f(n) = \Theta(g(n)),\ g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$

4 Transpose symmetry

$f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$ $\quad n^3 = O(2^n) \Rightarrow 2^n = \Omega(n^3)$

$f(n) = \Omega(g(n)) \Rightarrow g(n) = O(f(n))$ $\quad n^3 = \Omega(n^2) \Rightarrow n^2 = O(n^3)$

Recurrence Relations

- Substitution method
- Recurrence Tree method
- Master's theorem

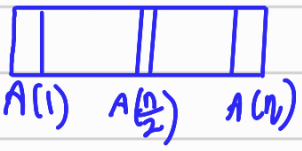
Linear search : $T(n) = 1 + T(n-1), \; T(1) = 1$

$$T(n) = 1 + 1 + T(n-2)$$
$$= 1 + 1 + 1 + T(n-3)$$
$$= \underbrace{1 + 1 + 1 \cdots + 1}_{n-1} + T(n-(n-1))$$
$$= n - 1 + T(1) = n - 1 + 1 = n$$

Best case : $\theta(1) = 1, 2 \ldots k$

Worst case : $\theta(n)$

Binary search :



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

$$= 1 + 1 + T(n/4)$$

$$\vdots$$

$$= \underbrace{1 + 1 + 1 + \cdots 1}_{k} + T\left(\frac{n}{2^k}\right)$$

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

Assume $n = 2^k$

$k = \log_2 n$

$= \log_2 n + T\left(\frac{2^k}{2^k}\right)$

$= \log_2 n + T(1)$

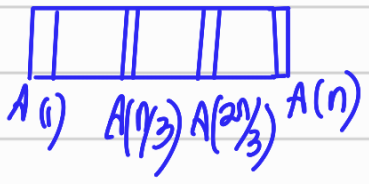
$= \log_2 n + 1$

Best case : $n/2, n/4 \ldots$

$\Theta(1) = 1$

Worst case : $\Theta(\log_2 n)$

Ternary Search



$T(n) = T(n/3) + 2, \quad T(1) = 1$

$= T(n/3^2) + 2 + 2$

$= T(n/3^k) + \underbrace{2 + 2 \ldots}_{k}$

If $n = 3^k \Rightarrow k = \log_3 n$ $\quad T\left(\frac{3^k}{3^k}\right) + 2k \Rightarrow T(n) = 2k + 1$

$= 1 + 2\log_3 n$

Comparision : $n = 81$

Binary Search	Ternary Search
$1 + \log_2 81$	$1 + 2\log_3 n$
$1 + \log_2 81 \leq 1 + \log_2 128$ ≤ 8 comp	$1 + 2\log_3 81 = 9$ comp

As per step count $1 + \log_2 n \leq 1 + 2\log_3 n$

As per asymptotic sense

$$\log_2 n = \frac{\log n}{\log 2} = \frac{\log n}{\log 3} \times \frac{\log 3}{\log 2}$$

$$\log_2 n = \log_3 n \times \underbrace{\log_2 3}_{1 < c < 2}$$

$$\log_2 n = c \cdot \log_3 n$$

$$\log_2 n = \theta(\log_3 n)$$

$\therefore \theta(\log n)$

↳ no need of mentioning the base bcoz in asymtotic notation all the $\log_{\text{base}}$ are equal.
We can mention the base but it will be of no use.

Finding Max — compare (<, >) - swap

$T(n) = 1 + T(n-1), T(n) = 0$

$T(n) = 1 + T(n-1)$

$= 1 + 1 + T(n-2)$

$= \underbrace{1 + 1 \ldots 1}_{n-1} + T(n-(n-1))$

$= n-1$

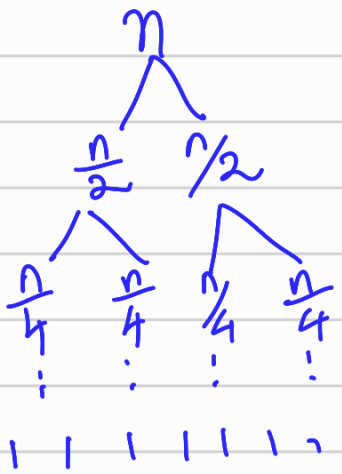
Comparisions: Best case, Worst case } $n-1$ comps, $\Theta(n)$

Swap: Best case: 0 swaps

Worst case: $(n-1)$ swaps

Comparisions will be the dominant one

Using Divide and conquer



$T(n) = T(\frac{n}{2}) + T(n/2) + 1$ (comparing both the max)

$T(n) = 2T(n/2) + 1 \quad T(1) = 0$

$= 2\left(2T\left(\frac{n}{4}\right) + 1\right) + 1$

$= 2^2 T(n/4) + 2 + 1$

$= 2^3 T(n/8) + 2^2 + 2 + 1$

$= 2^k T(n/2^k) + 2^{k-1} + 2^{k-2} \ldots + 2 + 1$

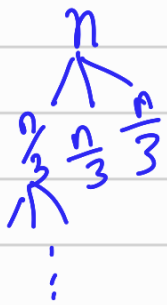
$n = 2^k$

$2^0 + 2^1 + 2^2 \ldots + 2^{k-1} = 2^k - 1$

$\Rightarrow 2^k T(1) + 2^k - 1$

$\Rightarrow T(n) = n - 1$

3-way Find max



$T(n) = 3T\left(\frac{n}{3}\right) + 2 \quad , T(1) = 0$

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

$= 3^2 T(n/3^2) + 3 \cdot 2 + 2$

$= 3^k T(n/3^k) + 3^{k-1} \cdot 2 + 3^{k-2} \cdot 2 \ldots + 2$

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

If $n = 3^k$

$1 + r \ldots + r^{k-1} = \frac{r^k - 1}{r - 1} \quad r = 3, \quad \frac{3^k - 1}{3 - 1} = \frac{3^k - 1}{2}$

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

Substitution method: change of Variable Technique

$T(n) = T(\sqrt{n}) + 1$

$n = 2^m \quad T(2^m) = T(\sqrt{2^m}) + 1 = T(2^{m/2}) + 1$

$S(m) = T(2^m) = S(m) = S(m/2) + 1$

$\Rightarrow \Theta(\log_2 m)$

$S(m) = \Theta(\log m)$

$T(n) = T(2^m) = \Theta(\log_2 m)$

$= \Theta(\log_2 \log_2 n)$

$T(n) = T(\sqrt{n}) + n$

$T(2^m) = T(\sqrt{2^m}) + 2^m$

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

$= S\left(\frac{m}{4}\right) + 2^{m/2} + 2^m$

$= S\left(\frac{m}{8}\right) + 2^{m/4} + 2^{m/2} + 2^m$

$= S\left(\frac{m}{2^k}\right) + 2^{\frac{m}{2^{k-1}}} + \cdots 2^m$

$\frac{m}{2^{k-1}} \quad \frac{2^k}{2^{k-1}}$

$2^{\frac{m}{2^{k-1}}} = 2^{\frac{2^k}{2^{k-1}}} = 2^2$

$m = 2^k$

$\underbrace{S(1)}_{\text{constant}} + 2^{m/2^{k-1}} + \cdots 2^m$

$2^m \leq 2^m + 2^{m/2} \cdots 2^2 + 2^1 \leq 2 \cdot 2^m$

$$= \theta(2^m)$$

$$T(n) = T(2^m) = \theta(n)$$

Recurrence tree

Binary search $T(n) = T(\frac{n}{2}) + 1, T(1) = 1$

I/p size | Solution tree

n $\frac{n}{2^0}$ level 0

$n/2$ $n/2^1$ level 1

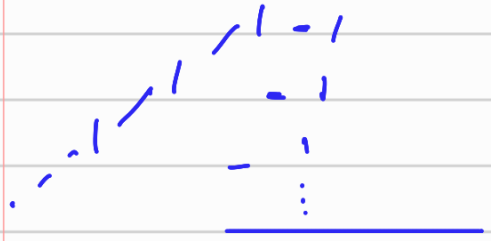
$n/4$ $n/2^2$ level 2

$n/8$ $n/2^3$

$\frac{n}{2^k} = 1$ level k : $\log_2 n$

$k = \log_2 n$ # levels : $\log_2 n + 1$

Computation tree

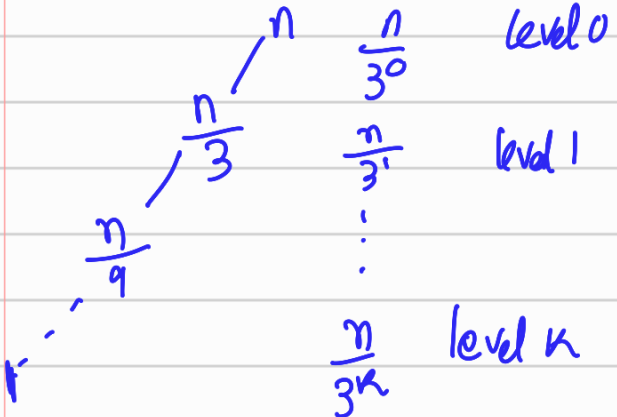


$(\log_2 n + 1) \times 1 = \log_2 n + 1$

$T(n) = 1 + \log_2 n$

Ternary search $T(n) = T(n/3) + 2$, $T(1) = 1$

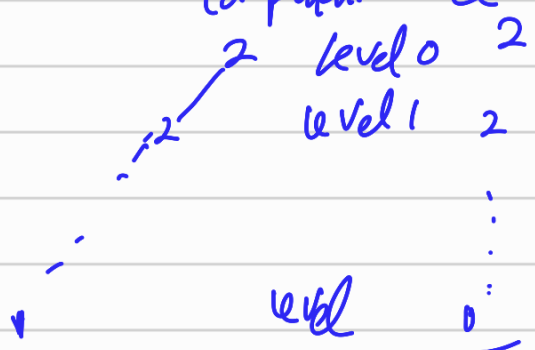
I/p size reduction Tree



$k = \log_3 n$

levels $= k+1 \Rightarrow \log_3 n + 1$

computation tree



$(2+2+\dots 2)+1$

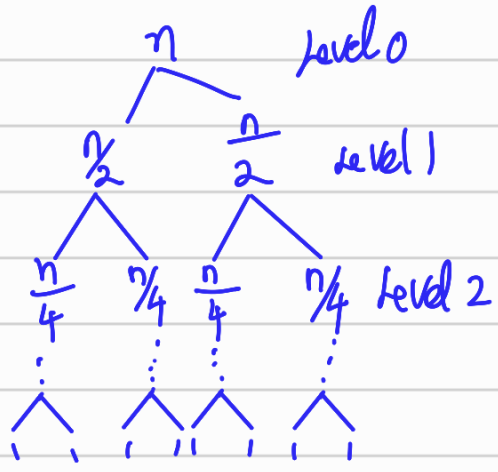
$= 2k+1$

$= 2(\log_3 n + 1$

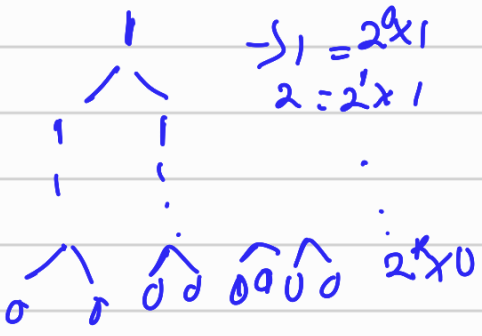
$= \Theta(\log_3 n)$

$= \Theta(\log_2 n)$

Find max



computation tree



$\rightarrow 1 = 2^0 \times 1$

$2 = 2^1 \times 1$

$2^0 + 2^1 + \dots 2^{k-1}$

$= 2^{\log_2 n - 1 + 1} - 1 = 2^{\log_2 n} - 1$

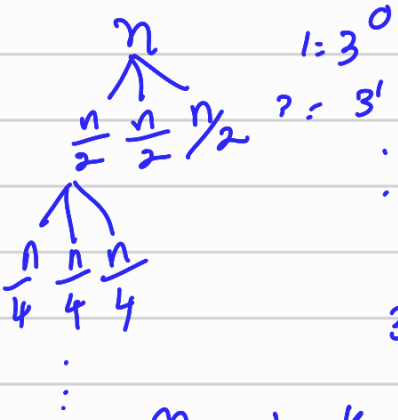
$= n - 1$

I/P size reduction tree

$\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

levels $= k+1 = \log_2 n + 1$

$T(n) = 3T(n/2) + n^2, \; T(1) = 1$



$1 = 3^0$

$? = 3^1$

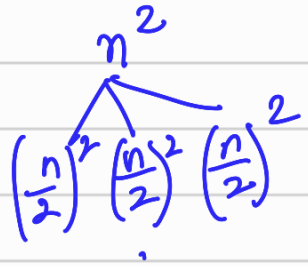
3^k leaf nodes

$\frac{n}{2^k} = 1, \; k = \log_2 n$

$3^k = 3^{\log_2 n} = n^{\log_2 3}$

Levels $= k+1 = \log_2 n + 1$

computation tree



$(\frac{3}{4})^0 n^2$

$= \frac{3n^2}{4} = (\frac{3}{4})^1 n^2$

$\frac{9n^2}{16} = (\frac{3}{4})^2 n^2$

$= (\frac{3}{4})^{k-1} n^2 = (\frac{3}{4})^{\log_2 n - 1}$

$T(1) \; T(1) \; T(1)$

leaves $\cdot T(1) = 3^{\log_2 n} \times T(1) = n^{\log_2 3}$

$= n^2 \left((\frac{3}{4})^0 + (\frac{3}{4})^1 \cdots (\frac{3}{4})^{\log_2 n - 1} \right) + n^{\log_2 3}$

$= n^2 \underbrace{\left[\frac{1 - (\frac{3}{4})^{\log_2 n}}{1 - \frac{3}{4}} \right]} + n^{\log_2 n}$

$(0.75)^1 = 0.75$

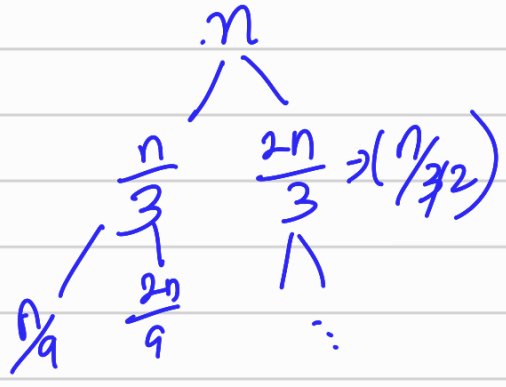
$(0.75)^2 = 0.5625$ $n \uparrow$ value $\downarrow$

$\therefore$ constant

$$n^2 \leq n^2\left[\frac{1-\left(\frac{3}{4}\right)^{\log_2 n}}{1-\frac{3}{4}}\right] + n^{\log_2 3} \leq 4 \cdot n^2$$

$$\theta(n^2)$$

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n \quad T(1) = 1$$



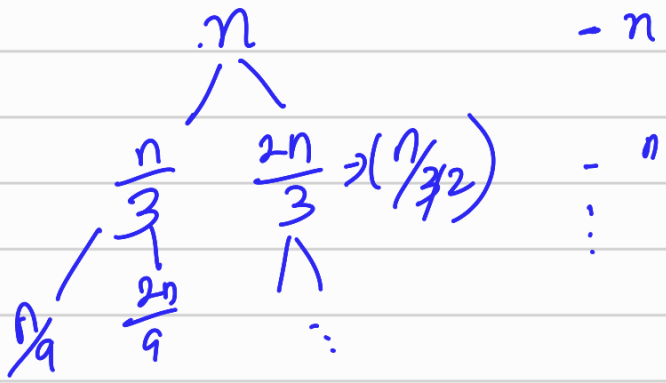
left $\frac{n}{3^k} = 1 \Rightarrow k = \log_3 n$

right $\frac{n}{\left(\frac{3}{2}\right)^{k'}} = 1 \Rightarrow k' = \log_{\frac{3}{2}} n$

$$\log_3 n < \log_{\frac{3}{2}} n$$

leaves start at $\log_3 n$ (i.e comes first) and end at $\log_{\frac{3}{2}} n$

Computation tree



Min height = $\log_3 n$ Max height = $\log_{\frac{3}{2}} n$

$$n \log_3 n \leq T(n) \leq n \cdot \log_{\frac{3}{2}} n$$

$T(n) = \Omega(n \log_3 n)$, $T(n) = O(n \log_{\frac{3}{2}} n)$

$$\log_3 n = \Theta(\log_{\frac{3}{2}} n)$$

$$T(n) = \Theta(n \log_3 n) = \Theta(n \log n)$$

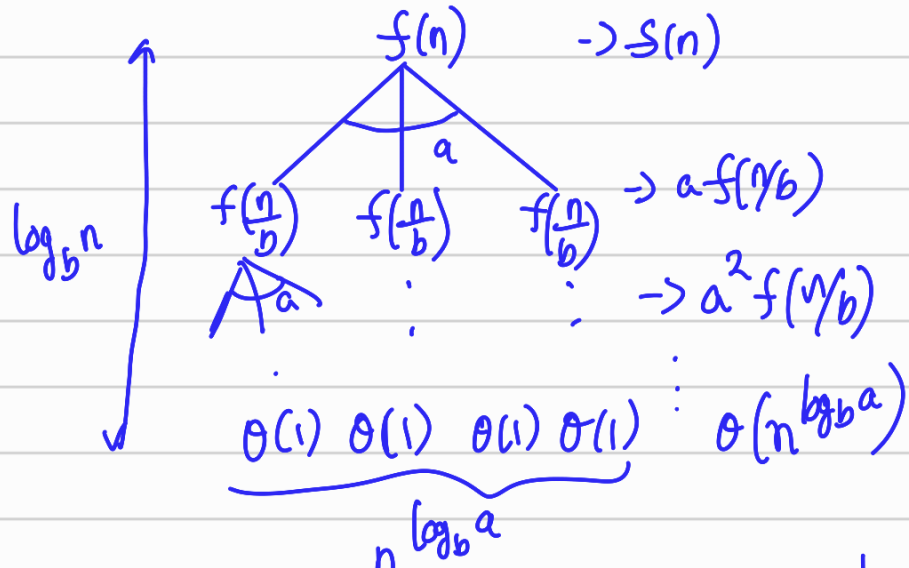
$$\Theta(n \log n)$$

Master's Theorem

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

↑ containing cost

Proof



$n^{\log_b a} > f(n)$ → $n^{\log_b a}$

$n^{\log_b a} < f(n)$ → $f(n)$

$n^{\log_b a} = f(n)$ → $\theta(n^{\log_b a} \log_b n) = \theta(f(n) \log_b n)$

Case 1: If $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ for $\varepsilon > 0$

then $T(n) = \theta\left(n^{\log_b a}\right)$

Ex: $T(n) = 3T(n/2) + n$

$n = O\left(n^{\log_2 3 - \varepsilon}\right)$ $\varepsilon > 0$

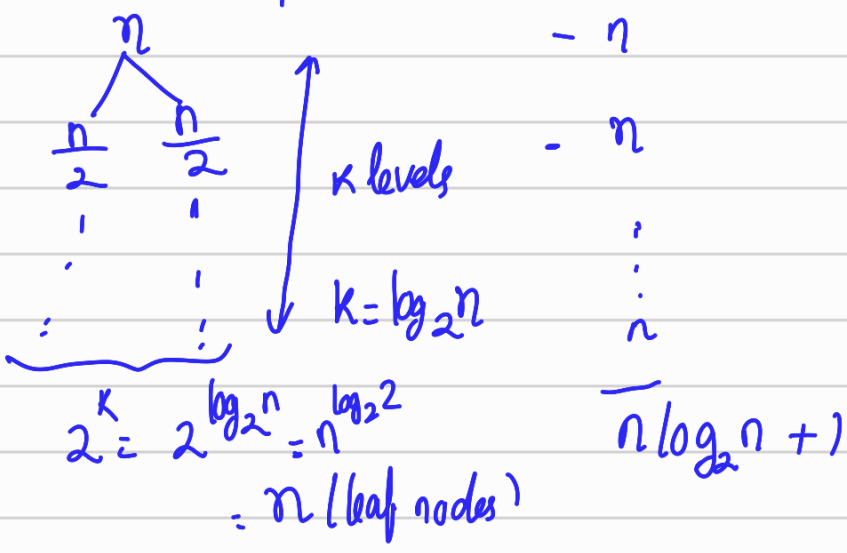
$\Rightarrow T(n) = \theta\left(n^{\log_2 3}\right)$

Case 2: If $f(n) = \Theta(n^{\log_b a})$

then $T(n) = \Theta(n^{\log_b a} \log_b n) \,/\, \Theta(f(n) \log_b n)$

Ex1: $T(n) = 2T(\frac{n}{2}) + n$

I/P tree method



$\Theta(n \log_2 n)$

Master's theorem

$n = n^{\log_2 2} \Rightarrow T(n) = \Theta(n \log_2 n)$

Case 3: If $f(n) = \Omega(n^{\log_b a + \varepsilon})$, $\varepsilon > 0$ & $\underbrace{a f(\frac{n}{b}) \le c \cdot f(n), c<1}_{\text{Regularity condition}}$

then $T(n) = \Theta(f(n))$

Ex: $T(n) = 3T(n/2) + n^2 \Rightarrow n^2 = \Omega(n^{\log_2 3 + t})$

$a f(\frac{n}{b}) = \frac{3n^2}{4} \le c \cdot n^2, \; c = \frac{3}{4} < 1$

$\Rightarrow T(n) = \Theta(n^2)$

Ex $T(n) = 2T(\frac{n}{2}) + n\log n$

$f(n) = n\log n \quad n^{\log_b a} = n^{\log_2 2} = n^1$

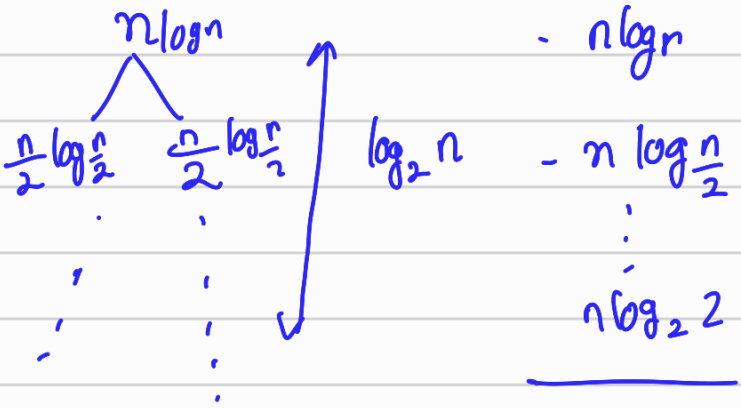
case 1: $n\log n \neq O(n^{1-\epsilon}) \quad \epsilon > 0$

case 2: $n\log n \neq \theta(n)$

case 3: $n\log n \neq \Omega(n^{1+\epsilon}), \epsilon > 0$

* $\log n$ cannot be compared asymtotically with any of the polynomial but $n\log n, n^2\log n$ can be compared.

Recurrence tree method



$n(k + k-1 + k-2 \ldots 1)$

$n\,\frac{k(k+1)}{2}$

$\theta(nk^2) = \theta(n(\log_2 n)^2)$

Advanced Master's theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$f(n) = \Theta\left(n^{\log_b a} \log^k n\right) \quad k \geq 1$$

$$\Rightarrow T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$$

Quick sort

6 10 13 5 8 3 2 11

```
Partition (A, p, q)        A[p...q]
  x = A[p]  -> pivot
  i = p
  for j <- p+1 to q

    do if A[j] <= x
      then i = i+1
           swap (A[i] A[j])

    if A[j] > x
      then j = j+1;

Swap(A[p], A[i]);
```

Counting sort

$\Theta(k)$ { for $i \leftarrow 1$ to k (→ max element in A)
do $C[i] \leftarrow 0$

$\Theta(n)$ { for $j \leftarrow 1$ to k
do $C[A[i]] = C[A[i]] + 1$;

$\Theta(k)$ { for $i \leftarrow 2$ to k
do $C[i] = C[i] + C[i-1]$;

$\Theta(n)$ { for $j \leftarrow n$ downto 1
do $B[C[A[j]]] \leftarrow A[j]$
$C[A[j]] \leftarrow C[A[j]] - 1$

$\Theta(n+k)$

Counting sort is not a comparision sort

Counting sort is a stable sort,

Radix sort

It has to started from least significant digit with auxilary stable sort.

counting sort $\Theta(n+k)$

Here, If each b-bit word is broken into r bit pieces each pass of counting sort takes

$O(n+2^r)$ time. Since there $\frac{b}{r}$ passes

$$\Rightarrow T(n,b) = \frac{b}{r} O((n+2^r))$$

$$2^r \gg n$$

$$r = \log n$$

$$T(n,b) = \frac{b}{\log n} O(n+n)$$

$$= O\left(\frac{bn}{\log n}\right)$$

$$T(n) = O(dn) \qquad d = \frac{b}{\log n}$$

Nowdays according to steep memory quick sort is better than radix and counting sort

Heap sort - Weakly ordered

Binary search tree

Worst case

Insertion: $\theta(n)$

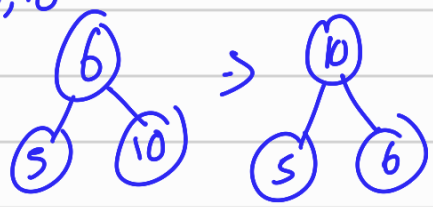
Search: $\theta(n)$

Deletion: $\theta(n)$

Heap are used for implementing priority queue

6, 5

6, 5, 10



Insertion $\theta(\log n)$

Build Heap: $\theta(n \log n)$

Deletion: $\theta(\log n)$

Heapsort: $\theta(n \log n)$

Element i child $-2i$, $2i+1$

Element j Parent $\cdot \text{int}\left(\frac{I}{2}\right)$

	B.C	W.C	Inplace	Stable
Bubble	$\theta(n^2)$	$\theta(n^2)$	✓	✓
Selection	$\theta(n^2)$	$\theta(n^2)$	✓	X
Insertion	$\theta(n)$	$\theta(n^2)$	✓	✓
Quick	$\theta(n\log n)$	$\theta(n^2)$	✓	X
Merge	$\theta(n\log n)$	$\theta(n\log n)$	X	✓
Heap	$\theta(n\log n)$	$\theta(n\log n)$	✓	X
Counting	$\theta(n+k)$	$\theta(n+k)$	X	✓
Radix	$\theta(dn)$	$\theta(dn)$	X	✓

Still we can make it stable

Is it possible to improve sorting algorithm better than $\theta(n\log n)$

Lower bound theory: Any comparision based sorting algorithm will be $\Omega(n\log n)$

Greedy Algorithm

1 Pruning - Linear search: n, 1, n-1, 1, n-2 ; Binary search: n, 1, n/2, 1, n/4

2 Incremental Design - Selection, Bubble, Insertion

3 Divide and Conquer - Merge, quick

4 Greedy

5 Dynamic

Greedy Algorithm
↳ Optimisation Problems
Minimisation Maximisation

↳ Decision Problem
I/P: Integer P
=> Is P prime or not
O/P: Yes/No

Greedy algorithm: Global optimum
↳ min/max

local opt 1 -> local opt 2 -> ... -> Global optimisation

Coin Change Problem

I/p x, Denomination : 5, 4, 2, 1

Supply max

if x = 13

$\lfloor \frac{13}{5} \rfloor = 2$ $\quad 13 - 2 \times 5 = 3$

$\lfloor \frac{3}{2} \rfloor = 1$ $\quad 3 - 2 \times 1 = 1$

(2, 0, 1, 1) -> Greedy Strategy - 4 coins

Is it optimal?

No

(1, 2, 0, 0) is the optimal

Greedy strategy

Works for some $(d_1, d_2 \ldots k)$ + Proof of correctness = Greedy algorithm

Doesn't work for some $(d_1', d_2', \ldots d_k')$

→ Brute force
→ Dynamic Programming

Knapsack Problem

I/P: $S = \{x_1, x_2 \ldots x_n\}$ objects
$w_1, w_2, \ldots w_n$ - weights
$P_1, P_2 \ldots P_n$ Profits

Objective: Find $s' \subseteq S$
Profit (s') is maximum

constraint: Weight $(s') \leq W$

Greedy strategy

① Pack max no of items

$w_1 < w_2 < w_3 \ldots < w_n$

But the profit may not be max

So it fails

② Greedy w.r.t Profit

Pack the highest profit item first

$S = \{x_1, x_2, x_3, x_5\}$

$\{10, 25, 40, 50\}$

$\{1, 2, 3, 4\}$

Optimal = $40 + 25 = 65$ Greedy $50 + 10 = 60$

$2 + 3 = 5 \leq 6$ $1 + 4 \leq 6$

So it fails

③ Greedy w.r.t $\frac{P_i}{w_i}$

Sort $\frac{P}{w}$ in decreasing order

$\frac{P_1}{w_1} \geq \frac{P_2}{w_2} \geq \cdots \geq \frac{P_n}{w_n}$

$w = \{1 \ 2 \ 4 \ 5\}$ $P = \{10 \ 25 \ 40 \ 50\}$

$\frac{P_i}{w_i} = \{10 \ 12.5 \ 10 \ 10\}$

$S' = \{x_2, x_3\}$ $S = \{x_2, x_1\}$

$25 + 40 = 65$ $25 + 10 = 35$

So it fails

Interval Scheduling

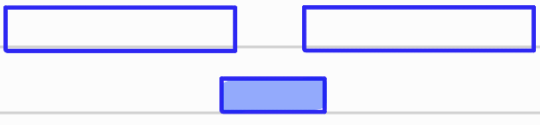
- Job j start at s_j and finishes at f_j
- Two jobs compatible if they don't overlap
- Goal: Find maximum subset of mutually compatible jobs

Greedy strategies

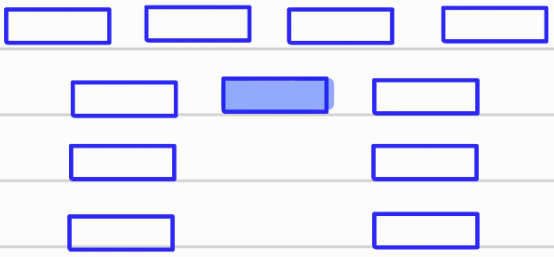
i) Earliest start time
ii) Earliest finish time
iii) Shortest interval
iv) Fewest conflicts

i)

iii)



iv)

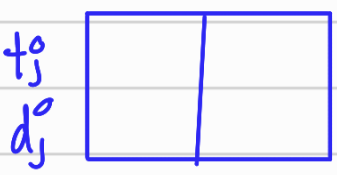


All the above strategies fail because it doesn't have a optimal solution

ii) sort jobs by finish time

```
A ← ∅
for (i = 1 to n)
    if (j is compatible with A)
        A ← A ∪ {j}
```

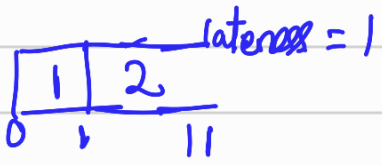
Minimizing Lateness



i) Shortest Processing time — Fails
ii) Smallest slack value ($d_j - t_j$) — Fails
iii) Earliest deadline first.

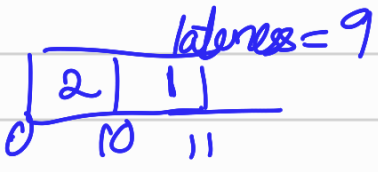
i)

	1	2
t_j	1	10
d_j	100	10

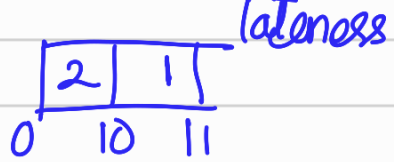


ii)

	1	2
t_j	1	10
d_j	2	10



Optimal



Optimal