__/__/__

ANALYSIS AND DESIGN OF ALGORITHMS

Unit = Algorithm Analysis:

* Problem solving: main steps
1) Problem definition
2) Algorithm design / Algorithm specification.
3) Algorithm analysis.
4) Implementation.
5) Testing
6) Maintenance

Algorithm: finite set of instructions, that if followed accomplishes a particular task.
It can be described as: natural language / pseudo-code / diagrams etc.
Criteria to follow:
+ Input: Zero or more quantities (externally pro
+ Output: ~~atleast~~ one or more quantities.
+ Definiteness: Clarity, precision.
+ Finiteness: Algorithm has to stop after a finite time.
+ Effectiveness:

Time Complexity
More imp than Space complexity.

Space Complexity
- fixed
- variable

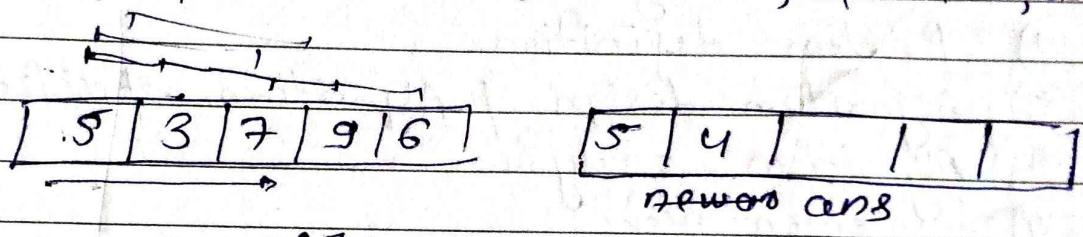
Best case? How to predict →
Input dependent
Hardware dependent

Sol^n: Doubling no. of elements. → growth ↑ → complexity ↑

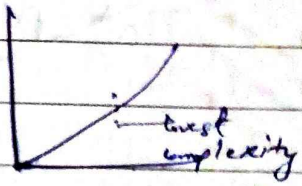
__/__/____

Assign: prefix averages. System call

1) Given an array X.
2) Compute the array A such that A[i] is the average of elements X[0] .. X[i], for i=0, .. n-1



(ii)
```
sum = a[i];
sum = 0;
for (i=0; i<n; i++){
    sum += a[i];
}
ans[i] = sum/i;
```



(i)
```
for (i=0; n; i++){
```

18/22 # Limitation of Empirical Study -

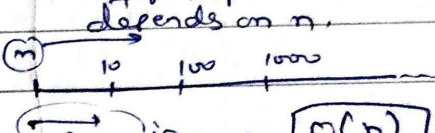
→ Necessary to implement the algorithm, which may be difficult.
→ Results may not be indicative of running time on other inputs not included in the experiment.
→ To compare the algos. the same hardware & software environments must be used.

* Asymptotic Notations –

Best Case (Element at first position) — O(1)

Average Case (Accessing half of elements) — O(n/2)

worst case (O) { upper bound of complexity } — for loop depends on n.



Q.
```
int find (int a[], int n, int x)
  { int i;
    for (i=0; i<n; i++)
      { if (a[i] == x)
          return i;
      }
    return 0;
  }
```

Sequential search of an unordered array.

O(n)

Q. Binary Search (Sorted Array)

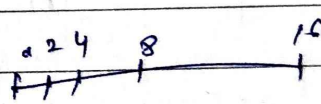
Log n

$n \rightarrow 2^n$

$16 \rightarrow 2^4$ (4) comparisons

16 elements

$8 \rightarrow 2^3$



4 comparison.

Avg – log n

worst – log n (upper bound)

Q. Selection sort : Select minimum & put in actual position

```
for( —  ) —O(n)
  for(  ) —O(n)
```

* Independent loops in functions complexity is / max of them.
  & nested for loop

```
for(  ){
  for(  )
    {
      }}
for(  )
```

separate

max of two

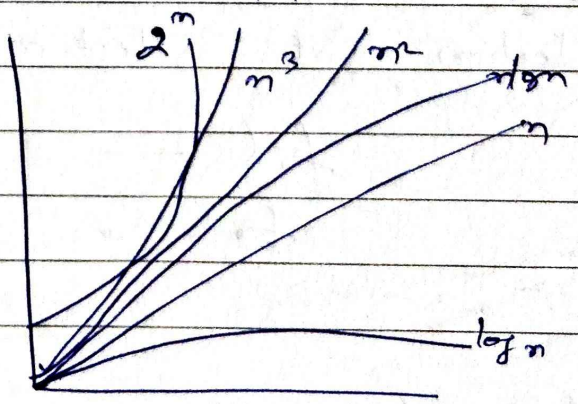
don't consider addition.

* Heap Sort → Implement DSA Priority Queue

max & min

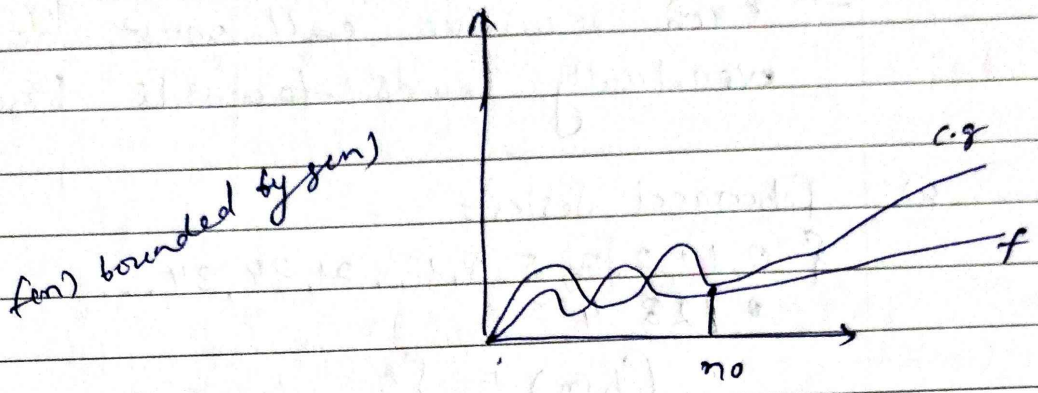
Comparison of Growth of common functions

n	log n	n log n	n^2	n^3	2^n	n!
1	0	0	1	1	2	1
2	1	2	4	8	4	2
4	2	8	16	64	16	24
8	3	24	64	512	256	40320
16	4	64	256	4096	.	.
32	5	160	1024	32768	.	∞
64	6	384	4096	262144	∞	∞



4/8/22

O(1) : No growth
O(N) : Grows linearly by doubling the value of N.



point after which distinguishable after 'no'.

* logn < n < nlogn n^2 n^3 ..... n^k 2^n < n!

Complexity -

$N = O(2^N)$ 2^N is not $O(N)$; 2^N upper bound
2^{10N} is not $O(2^N)$; N has lower bound

* Relatives of Big-oh (worst case)

Big omega (best case)
big. theta (avg case)

big { O, Ω, Θ
small { o, omega

f(n) is O(g(n)) if f(n) is asym <= g(n)
Ω(g(n)) >= g(n)
O(g(n)) =
o(g(n)) strictly.
ω(g(n))

__/__/____

* **Recurrence relation:** An equation which is defined in terms of itself.

Two fundamental rules –

→ Must always have a base case. to quit the recursion.

→ Each recursive call must be a case that eventually leads towards base case.

Ex: Fibonacci Series –

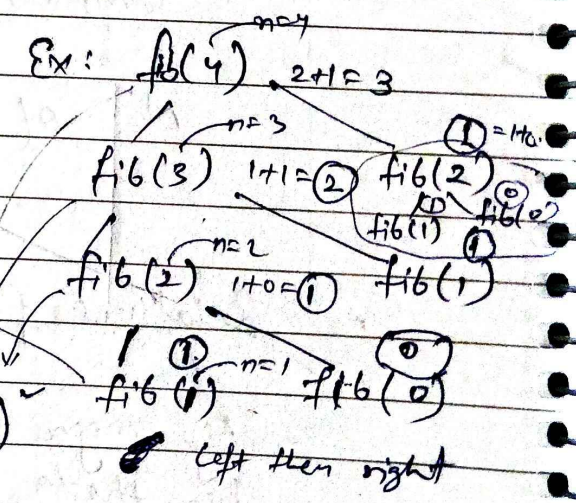
{ 0, 1, 1, 2, 3, 5, 8, 13, 21, 24, 34 ...
  0 1 2 3 4 5 6 ...

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fib(n-1) + fib(n-2) & \text{if } n>1 \end{cases}$$

Recurrence Tree

$\boxed{f(4) = 3}$

Algo fib(n)
  if n=0 then
    return(0)
  if n=1 then
    return(1)
  return (fib(n-1) + fib(n-2))



Recurrence Relations ~ Time of Recursive Program

T(n) : no. of 'Hi' on input of size = n.

Base condition.
T(1) = 1
T(n) = aT(n/b) + f(n)
f(n) → Upper bound

Q. $S(n) = \begin{cases} 0 & , n=0 \\ c + S(n-1), & n>0 \end{cases}$

Q. $S(n) = \begin{cases} 0 & , n=0 \\ n + S(n-1), & n>0 \end{cases}$

Q. $T(n) = \begin{cases} c & , n=1 \\ 2T\left(\frac{n}{2}\right) + c, & n>1 \end{cases}$

Q $T(n) = \begin{cases} c & , n=1 \\ aT\left(\frac{n}{b}\right) + c, & n>1 \end{cases}$

* Recursion Methods –
1) Substitution Method
2) Recursion Tree or Iteration Method.
3) Master method

Substitution Method:
1) Guess the form of solution.
2) General form of polynomial
$T(n) = an^2 + bn + c$

Q. R.Rel^n: $T(1)=1$ & $T(n)=4T(n/2)+n$

Guess $T(n)=an^2+bn+c$

LHS	RHS
$T(1)=a+b+c$	1
$T(n)=an^2+bn+c$	$4T(n/2)+n$
	$=4\left[a\left(\frac{n}{2}\right)^2+b\left(\frac{n}{2}\right)+c\right]+n$
	$=an^2+(2b+1)n+4c$

Equate the coefficients & find a, b, c.

$a=2$, $b=-1$, $c=0$

$\therefore a+b+c=1$

$\therefore 2b+1=b$

$b=-1$

$4c=c$

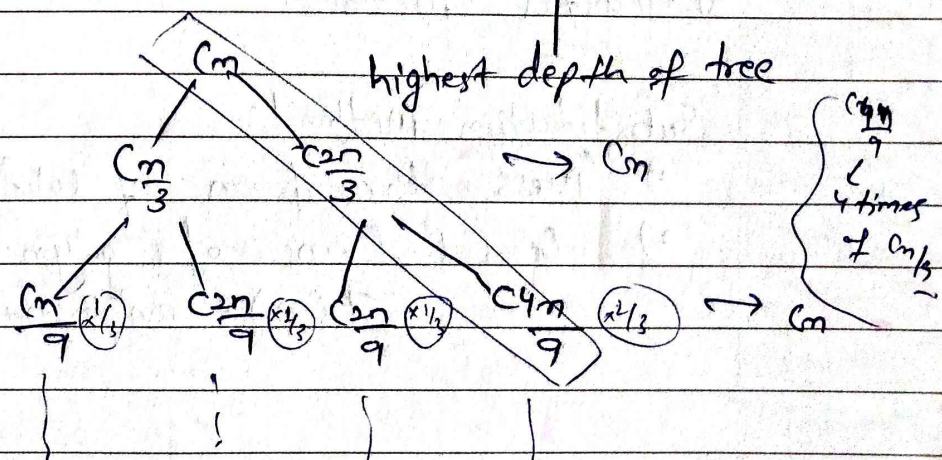
$3c=0$

$c=0$

$\boxed{T(n)=2n^2-n}$ Ans.

Recursion Tree –

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



most no. of levels in Tree + Cost at end level.

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

$n \rightarrow \frac{2}{3}n \rightarrow \left(\frac{2}{3}\right)^2 n$

$\downarrow$

$\left(\frac{2}{3}\right)^k n = 1$ one element in end.

$k \rightarrow \log_{3/2} n$

$\Rightarrow (\log_{3/2} n) * cn = cn \log_{3/2} n$

$= O(n \log_2 n)$

Q. Complexity of Fibbonacci Series -

In recursion tree left side has longer depth than right side.

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

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

$T(1) \rightarrow T(0) \Rightarrow O(1)$

$T(n-2) \cong T(n-1)$

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

$2T(n-1) + c$

$2(2T(n-1-1)+c)+c = 4T(n-2)+3c$

mathematical Induction.

$\downarrow$

again M.I.

$= 8T(n-3) + 7c$

$= 2^k (n-k) + (2^k - 1)c$

put $n-k=0$

$n=k$

$= 2^n T(0) + (2^n - 1)c$

$= 2^n + (2^n - 1)c$ $\quad \{T(0) = 1$

$= O(2^n)$

$2^n + 2^n \cdot c - c$

$2^n (1 + c) - c$

$\approx 2^n$.

3) Master Method -

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

$a \geq 1$ & $b > 1$

ϵ = const.
$\epsilon > 0$

Case I: If $f(n) = O(n^{\log_b a - \epsilon})$ $\epsilon > 0$ then

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

gen > f(n) Ω

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

then $T(n) = \Theta(n^{\log_b a} \cdot \log_2 n)$

f(n) = gen

Case III: If $f(n) = \Omega(n^{\log_b a + \epsilon})$, $\epsilon > 0$

& If $a \cdot f\left(\frac{n}{b}\right) \leq c f(n)$ then

(regularity condition)

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

f(n) > gen Ω

Polynomially Larger - A function $f(x)$ is polynomially larger than $g(x)$ if it is greater only by polynomial factor. $n^2, n^3, n^{\log n}$

But this exponent should be constant not a function of n.

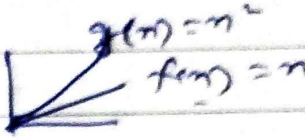
Q. $T(n) = 9T\left(\frac{n}{3}\right) + n$ ($f(n)$)

Sol^n: $a = 9$, $b = 3$ — $f(n)$, $g(n)$

$n^{\log_b a} = n^{\log_3 9} = (n^2)$.

$f(n) = n$ (smaller)

$g(n) = n^2 = n^c$ (greater)



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

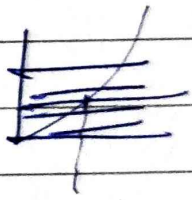
Q. $T(n) = T\left(\frac{2n}{3}\right) + 1$

$a = 1$, $b = \frac{3}{2}$, $f(n) = 1$.

Now $n^{\log_b a} = n^{\log_{\frac{3}{2}}(1)} = n^{\log_3 / \log_2} = n^0 =$

~~$g(n) = n^{\log_2 3} = n^c$~~

$f(n) = g(n)$



$\boxed{T(n) = \Theta(1 \cdot \log_2 n)}$

Q. $T(n) = 3T\left(\frac{n}{4}\right) + n \log_2 n$

$a = 3$, $b = 4$, $f(n) = n \log_2 n$

$g(n) = n^{\log_b a} = n^{\log_4 3} = n^{0.293}$ ~~const~~

$f(n) > n$.

$= O(n^{0.293})$

Check regularity condition –

$$af\left(\frac{n}{b}\right) \le c\,f(n)$$

$$3f\left(\frac{n}{4}\right) \le c \cdot n\log_2 n$$

$$3\cdot\left(\frac{n}{4}\right)\log_2\left(\frac{n}{4}\right) \le \frac{3}{4}\cdot n\log_2 n \qquad \text{let } c = 3/4$$

$$\log_2 \frac{n}{4} \le \log_2 n. \qquad \text{(III rd case)}$$

$$\boxed{T(n) = \Theta(n\log_2 n). \quad = \Theta(f(n)).}$$

Q. $T(n) = 2T\left(\frac{n}{2}\right) + n\log_2 n.$

$a = 2, \quad b = 2. \quad f(n) = n\log_2 n.$

Now $n^{\log_b a} = n^{\log_2 2} = n^{(1)} = n$

III rd case:–
Master Theorem can't be applied.

0, 0, 2, $+(n)$, $+(n)$ $+t$

1, 2, 3

Q. $n^2 n^5 = n^3 \log n$

polynomially greater

should be const.

$n(\log_2 n) = n$ asymptotically greater not by polynomially.

Q. $n^{(n)} = n^2$

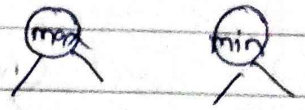
Asymtotically greater not polynomially.

In General $n\log n > n$

Heap Sort

Both are complete binary tree

max heap min heap



- How to store binary tree in memory?

→ Using a linked list.

Node * left, right

→ Using array.

Space wastage

Complete binary tree → very less space wastage.

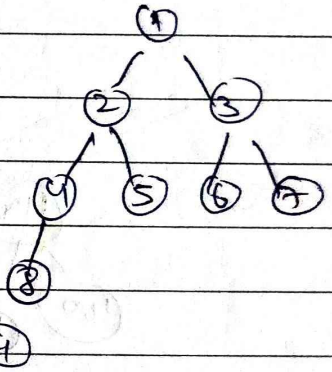
Every node will have two children. & last child of last level will be in left.

→ An node at 'i'th position then -

Parent → $\lfloor \frac{i}{2} \rfloor$

Left → $2i$

right child → $2i+1$

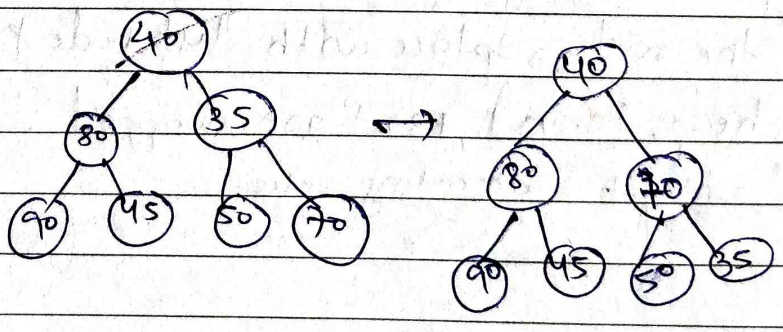


1	2	3	4	5	6	7	8

$8/2 = 4$

Q. 40, 80, 35, 90, 45, 50, 70

Complete binary array - max heap

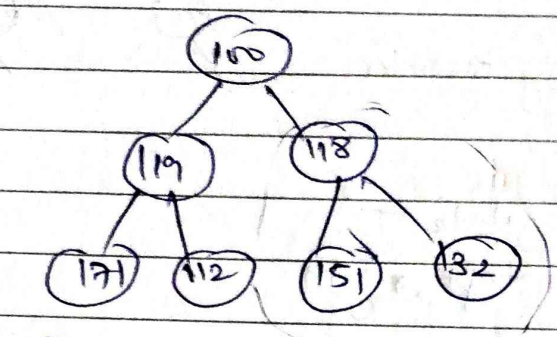


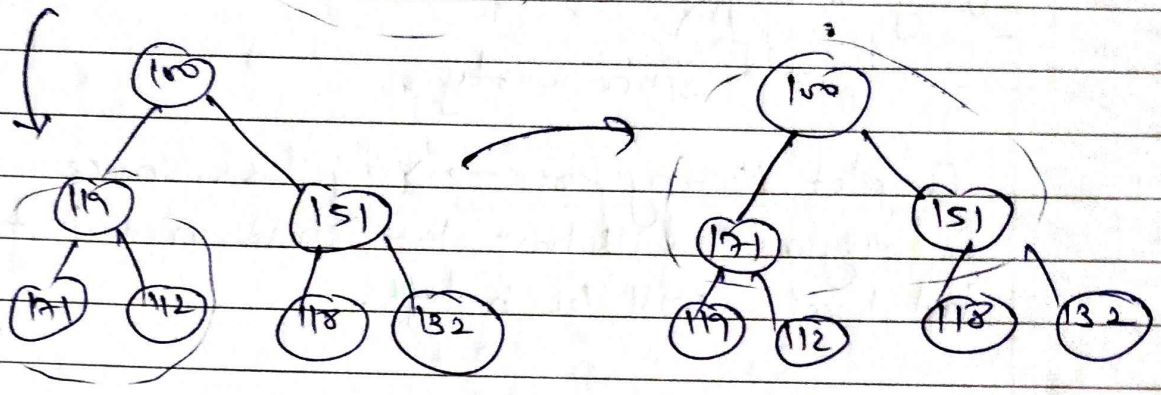
we'll solve later

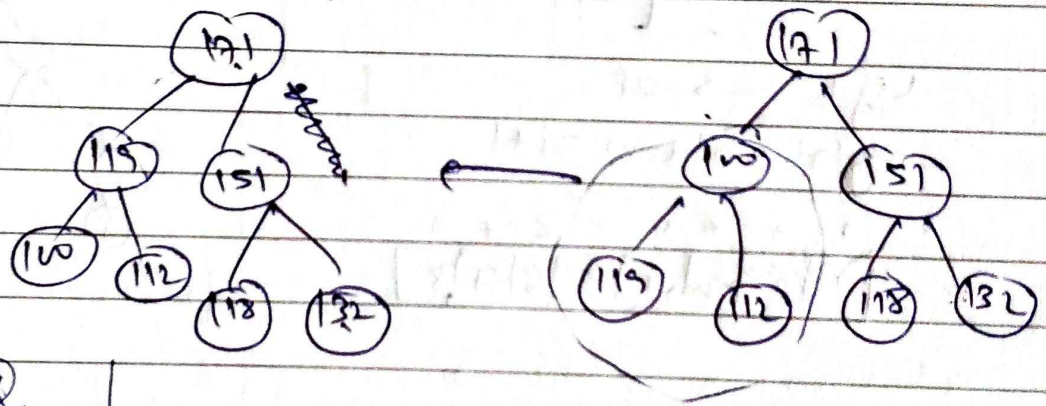
Q. 100 119 118 171 112 151 132.

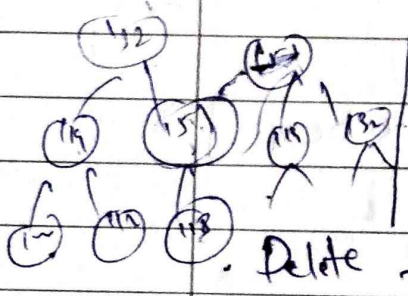
Create max heap

Root is max.









171 119 151 100 112 118 132

171

. Delete the root, replace with last node & again make max. heap. ie. 171, 132 are swapped.

we will get ascending order.

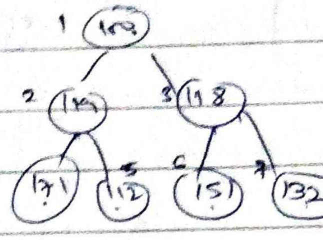
100 119 118 171 112 151 132   / /

IEEE student branch

for(i=7; i≤2; i--)

Heapsort (A,n)
Call Heapify (A,n) — O(n)
for i ← n to 2 by -1 — nlogn
  Swap A(i) ↔ A(1)
  Call adjust (A,1,i-1) — main — logn

heapsort complexity → I≠ nlogn



Heapify (A,n)
  for i ← ⌊n/2⌋ to 1 → $O(\frac{n}{2}) \to O(n)$
    Call Adjust

n=7
i = 7 to 2
$O(\frac{n}{2})^*$

Adjust (A,i,n)
  j ← 2*i, item ← A(i)
  while j ≤ n
  { if j<n ? & A(j) < A(j+1) (if no right child)
      then j ← j+1
    if item ≥ A(j) then exit
    else A[⌊j/2⌋] ← A(j)
      j ← 2*j
  }
A[⌊j/2⌋] ← item

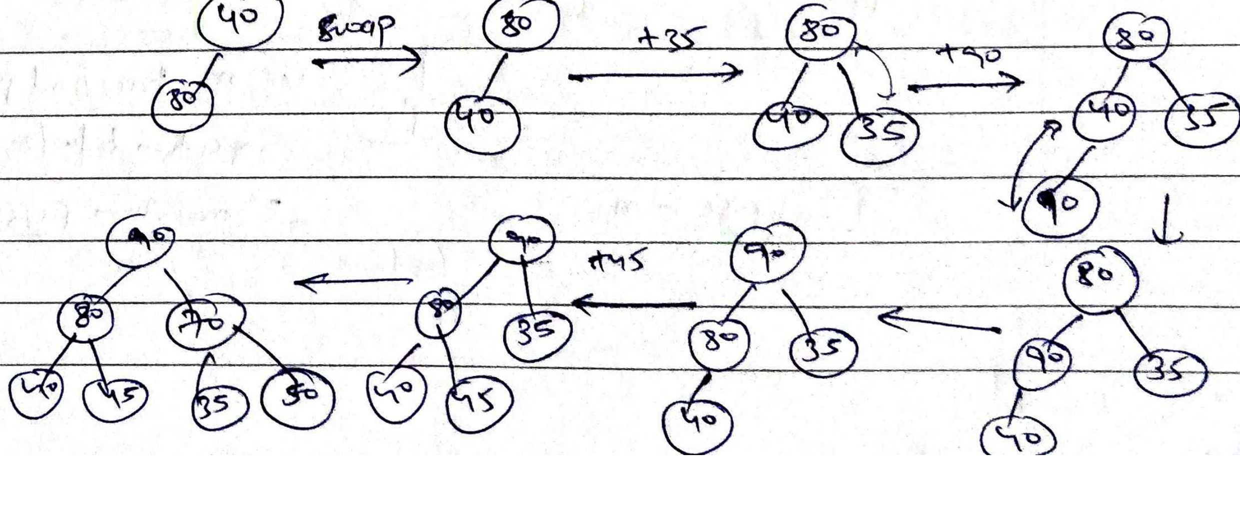
tree like structure logn

like binary search tree

* Insert an element into a heap-

40, 80, 35, 90, 45, 50, 70.

If heap given completely empty.

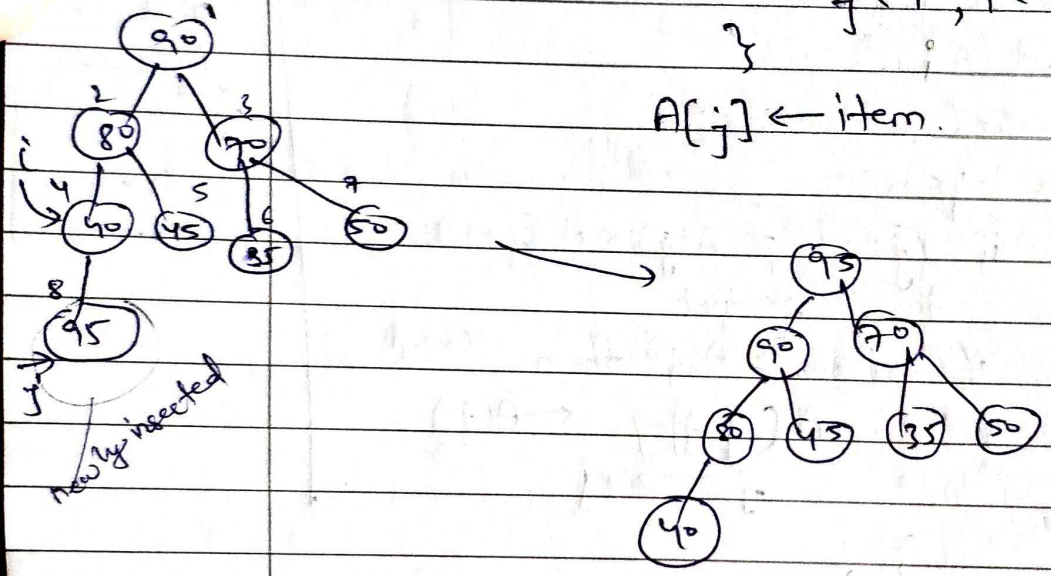


If heap is not empty → then insert from very left of root.

```
Insert (A,n)
    j ← n,  i ← (⌊n/2⌋)   item ← A(n)
    while i ≥ 1 & A[i] < item
        { A[j] ← A[i]
          j ← i , i ← ⌊i/2⌋
        }
    A[j] ← item.
```

i = parent of last element

i = 4

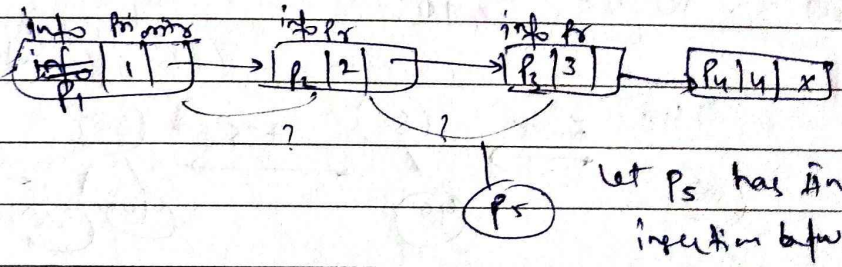


Compleaity = $\log n$.

* How to implement max priority queue, min P.Q.

If CPU is busy then next processes will go to waiting queue → FIFO

Implement using Linked list.



disadvantage → space

worst case - n

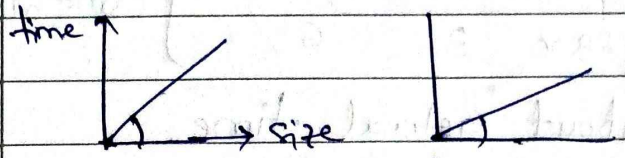
Let P_5 has 3rd priority then insertion b/w (2,3)

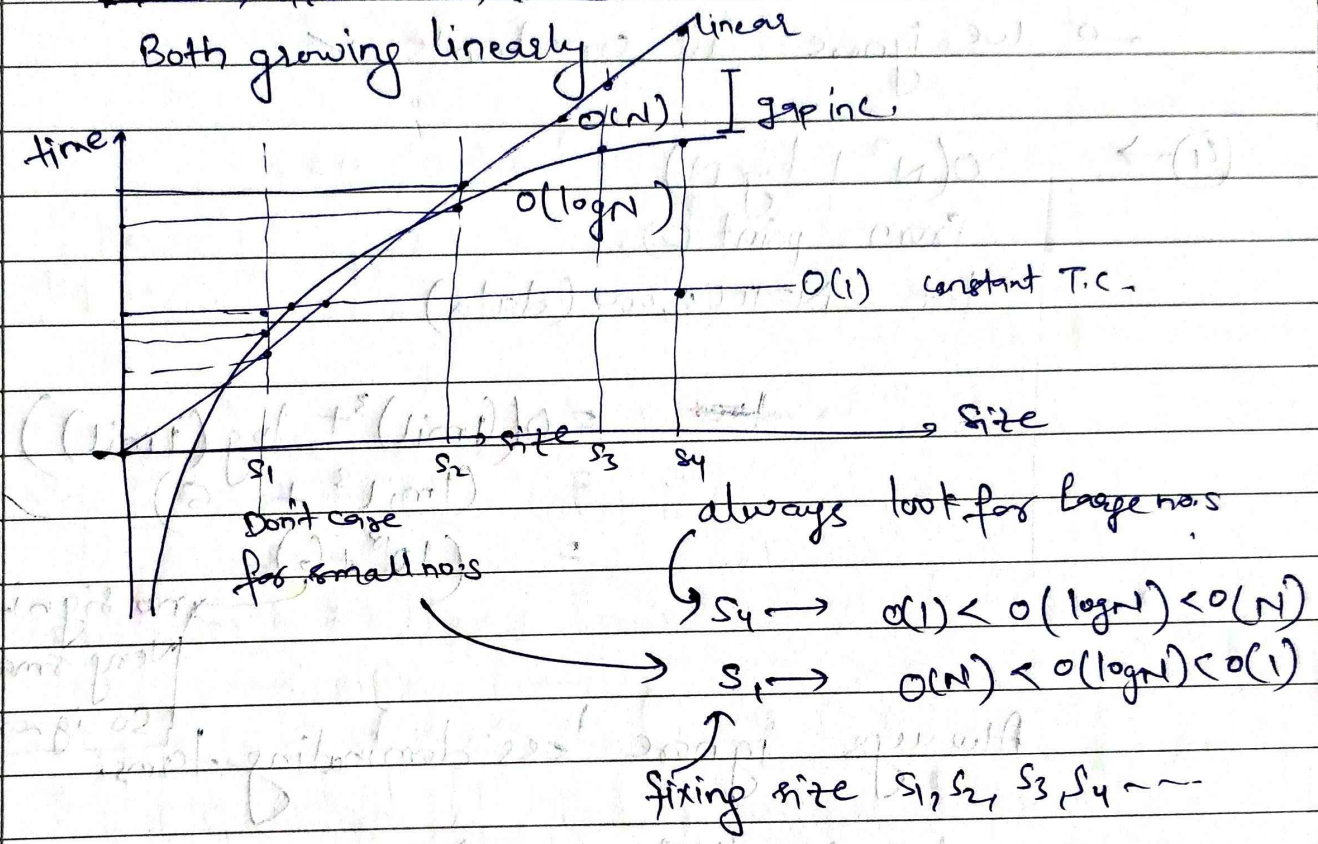
∴ maintain FCFS.

Time Complexity ≠ Time Taken

(vs)

Time Complexity tells us how the time grows as input grows.



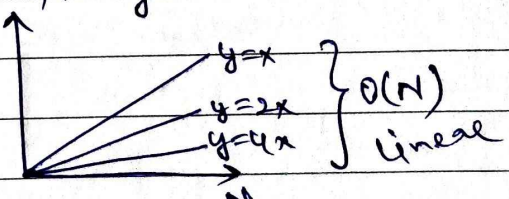


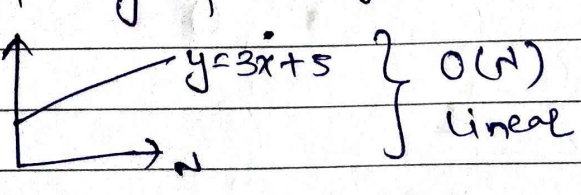
Q. What do we consider when thinking about complexity?

① → Always look for worst case complexity. (Ex: 10 users or 1 million users using same website, so optimize the code for worst case).

② → Always look at complexity for large / ∞ data.

③ →





Rules -

__/__/__

- Even tho values of actual time is different, they are all growing Linearly.

i.e.

	x=1	x=2	
$y=x$	1	2	actual time
$y=2x$	2	4	all growing linearly.
$y=3x$	3	6	

- We don't care about actual time.
- We ignore all constants. ✓

④ → $O(N^3 + \log N)$

From point ②.

let $N = 1000,000$ (data)

$= O((1mil)^3 + \log(1mil))$

$= (1mil^3 + 6)$

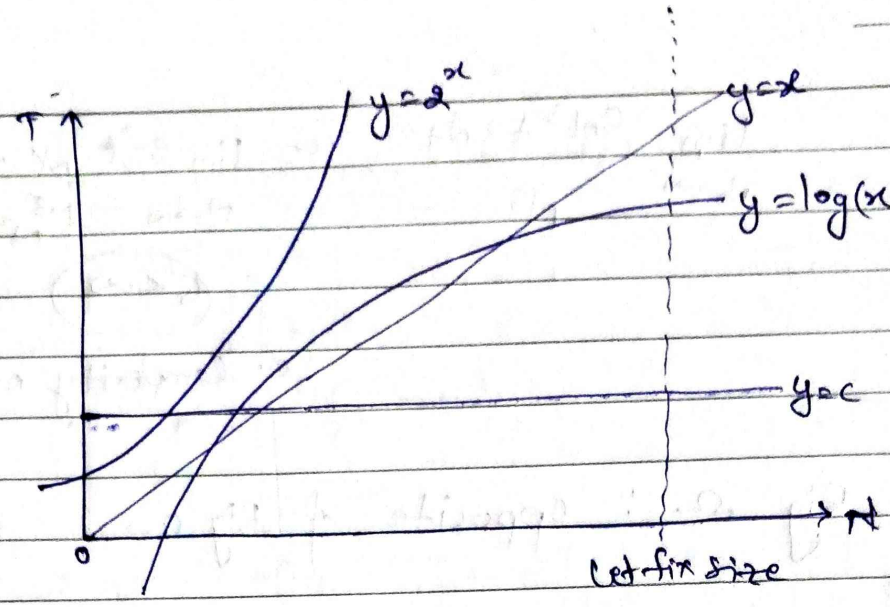
$= (10^{18} + 6)$ → no significance. Very small. So ignore it

Always ignore less dominating terms.

Q. $O(3N^3 + 4N^2 + 5N + 6)$

$= O(N^3 + N^2 + N)$ — from ③

$= O(N^3)$ — from ④



$O(1) < O(\log N) < O(N) < O(2^N)$ (fiboracci) $< O(n!)$

$O(N \log N)$ — Radert complexity

n^2 n^3 ..... n^k

* Big-O Notation

word definition -

Ex: $O(N^3)$ ⇒ upper bound.

i.e. complexity should not exceed $O(N^3)$.

Problem could be solve in $O(1)$ | $O(\log N)$ ... $O(N \log N)$ | $O(N^2)$ but can't be solve in $O(N^4)$.

Maths

$$O(f(N)) = O(g(N))$$

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} < \infty$$ i.e. finite value

Ex: $O(N^3) = O(6N^3 + 3N + 5)$

$g(N) = f(N)$

$$\lim_{N\to\infty} \frac{6N^3+3N+5}{N^3} = \lim_{N\to\infty} 6 + \frac{3N^2}{N^3 \cdot N^2} + \frac{5}{N^3}$$

$$= 6 < \infty$$

i.e. Complexity never exceed 6.

* Big Ω : opposite of Big O.

words:

Ex: $\Omega(N^3)$ ⇒ lower bound.
atleast N^3 time complexity required.
Can also solve in N^4, N^5 ... 2^N, $N!$ etc.

maths: $\lim_{x\to\infty} \frac{f(x)}{g(x)} > 0$

* Q. What if an algo has lb & upper as (N^2)

Ans: $\Theta(N^2) \Rightarrow O(N^2)$ & $\Omega(N^2)$ Combining both.

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

* Little O notation-
This is also giving u-b.
words: Loose u-b.

Big O	Little O
$f = O(g)$	$f = o(g)$
$f \le g$	$f < g$ · f is strictly slower than g.
growth of $f \le g$.	

Maths. $\lim_{n\to\infty} \frac{f(N)}{g(N)} = 0$

$f = N^2$, $g = N^3$

$\lim_{N\to\infty} \frac{N^2}{N^3} = 0$

* little omega – (ω) $f = \omega(g)$

Ω ... ω

loosely ~ B

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

$f = \omega(g)$ $f > g$ strictly greater

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

SPACE COMPLEXITY

Original space + Extra space

(given) (We create)

(space used by input) | (Auxiliary space)

Q.
```
for (i=1; i<=N;) {
  for (j=1; j<K; j++)
    { statements; } t time
}
i = i+1
}
```

Inner loop = $O(k)$

Ans: $O(kt$ * times outer loop is running)

i = i + k

x = 0, 1, 2, 3, ...

i = 1, 1+k, 1+2k, 1+3k ... (1+xk)

1+xk ≤ N

xk ≤ N-1

$x \leq \frac{N-1}{k}$ ~~Last value~~

Times the outer loop is running.

Complexity = $O\left(k * \frac{N-1}{k}\right)$

$= O(t*(N-1))$ ignore

$= O(tN)$ → const.

$= O(N)$.

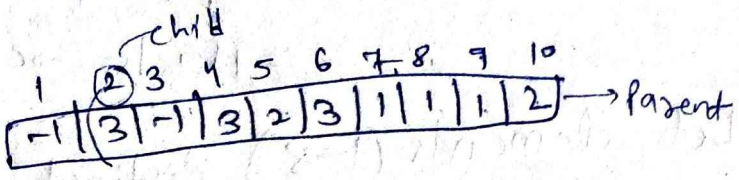
Max & Min Heap, priority queue Implementation.
Helpful in - (KRUSHKAL'S ALGO) Job Sequencing with deadline.

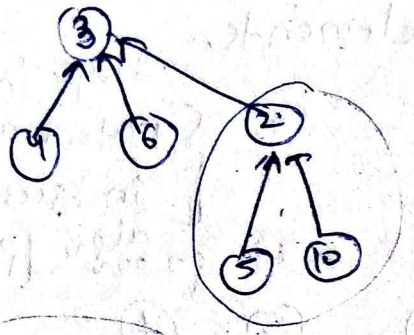
Union of sets:-

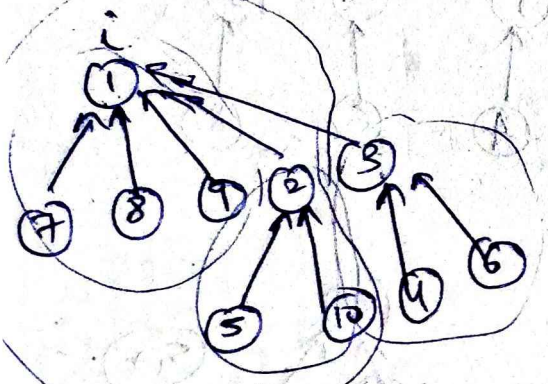
$S_1 = \{1,2,3\}$

$S_2 = \{2,3,4\}$

$S_1 \cup S_2 = \{1,2,3,4\}$

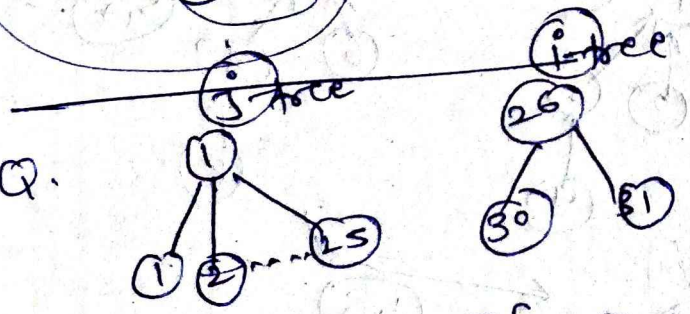






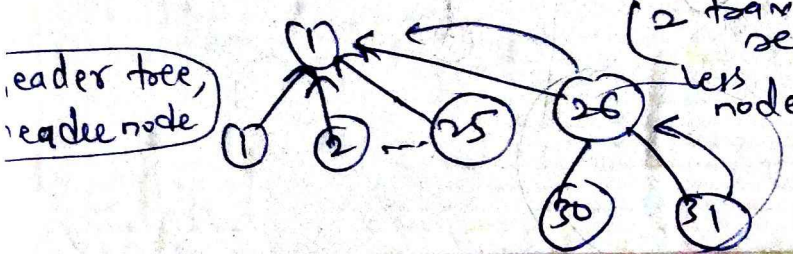

```
union(i,j)
    P[i] ← j        ——— O(1)
find(i)             ——— logn
while(P[i] ≥ 0)
    {do i ← P[i]}
return i;
```

Any set can be a parent.



Weighing Rule: If a no. of node in tree 'i' is less than no. of node in tree 'j' Then make 'j' as parent of 'i'

weigh(i,j)



ADA

16/8/22

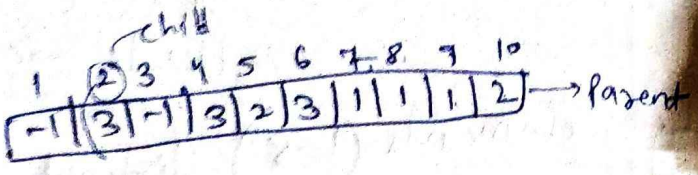
lax & Min Heap, priority queue Implementation.

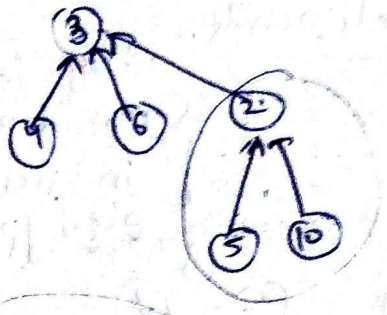
Union of sets:- → (KRUSHKAL'S ALGO) Helpful in -
Job Sequencing with deadline.

$S_1 = \{1,2,3\}$

$S_2 = \{2,3,4\}$

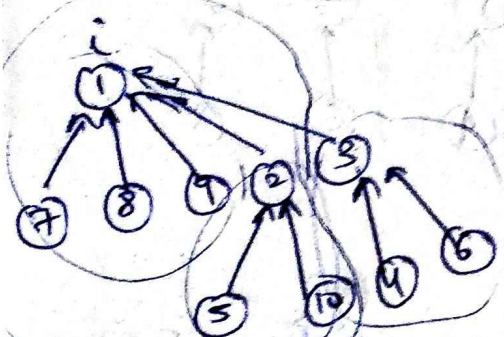
$S_1 \cup S_2 = \{1,2,3,4\}$

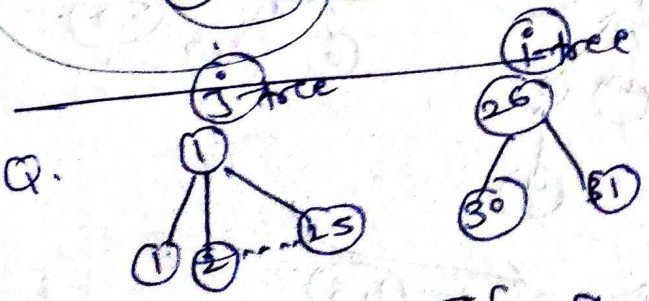





```
Union(i,j)
    P[i] ← j        ——— o(1)
Find(i)             ——— logn
while( P[i] ≥ 0)
      {do i ← P[i] }
return i;
```

Any set can be a parent.

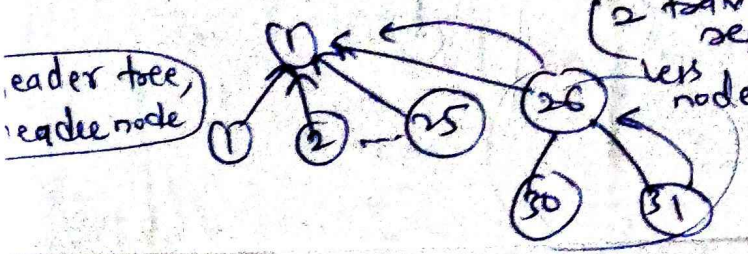




Q.

Weighing Rule: If a no. of node in tree 'i' less than no. of node in tree 'j' Then make 'j' as parent of 'i'

weigh(i,j)



Header node for storing info of multiple nodes.

```
weigh(i,j)
  x ← P[i] + P[j]
  If P[i] > P[j]      (means i<j)
      P[i] ← j
      P[j] ← x
  else
      P[j] ← i
      P[i] ← x
```

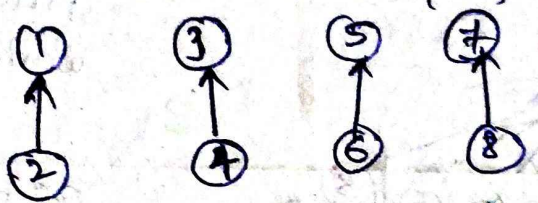
Q. Let elements (1–8) distinct elements.

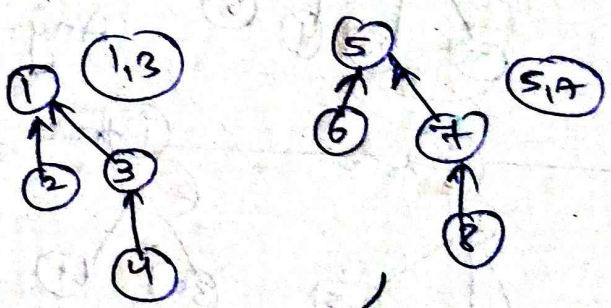
{1}, {2}, ... {8}

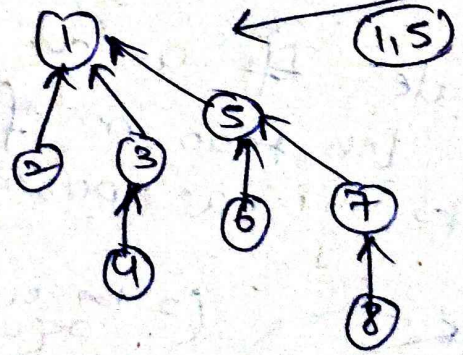
Helps in –
{ Detect a cyc
in Kruskal
algo Prim

union(1,2)
union(3,4)
union(5,6)
union(7,8)
union(1,3)
union(5,7)
union(1,5)

{1,2} {3,4} {5,6} {7,8}







* Divide & Conquer :-
Binary Search, Merge Sort.

Q. find max & min in an Array.

```
maxmin(A, n, max, min)
    max ← min ← A(1)
for i ← 2 to n; if(A(i) > max)
                    max ← A(i)
    else if (A(i) < min)
                    min ← A(i)
```

Comparisons
(n-1)
(n-1)
2(n-1)

(dec. order) worst case : 2(n-1)

MaxMin(i, j, fmax, fmin) fmax = final.max

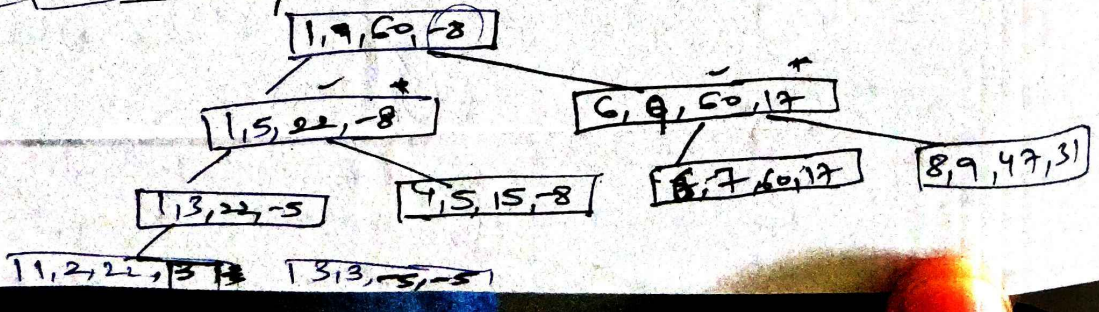
Case: i=j, max ← min ← A(i)

i=j-1 if A(i) < A(j)
    fmax ← A(j) &
    fmin ← A(i)
    else fmin ← A(j) &
    fmax ← A(i)

consecutive memory location.

else mid ← L(i+j)/2] four value.
Call maxmin(i, mid, Lmax, Lmin)
Call maxmin(mid+1, j, Rmax, Rmin)
fmax ← max(Lmax, Rmax)
fmin ← min(Lmin, Rmin)

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
22	13	-5	-8	15	60	17	31	47

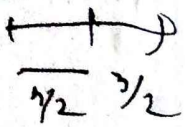


$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 2 \qquad n > 2$$

L R

$n = 2$

$n = 1$



Rewrance Relation of max min.

$n = 2^k$

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

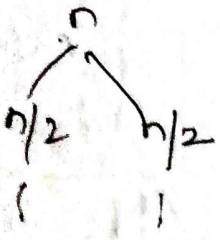
$$= 2\left(2T(n/4) + 2\right) + 2$$

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

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

$$= \frac{n}{2^k}(2) + n - 2$$

$$= \frac{n}{2} + n - 2 = \frac{3n}{2} - 2$$



$\frac{n}{2^k} = 2$

$n = 2^k \cdot 2$

$n = 2^{k+1}$

→ Comparisons are less for recursion.

→ which sol^n is slower?

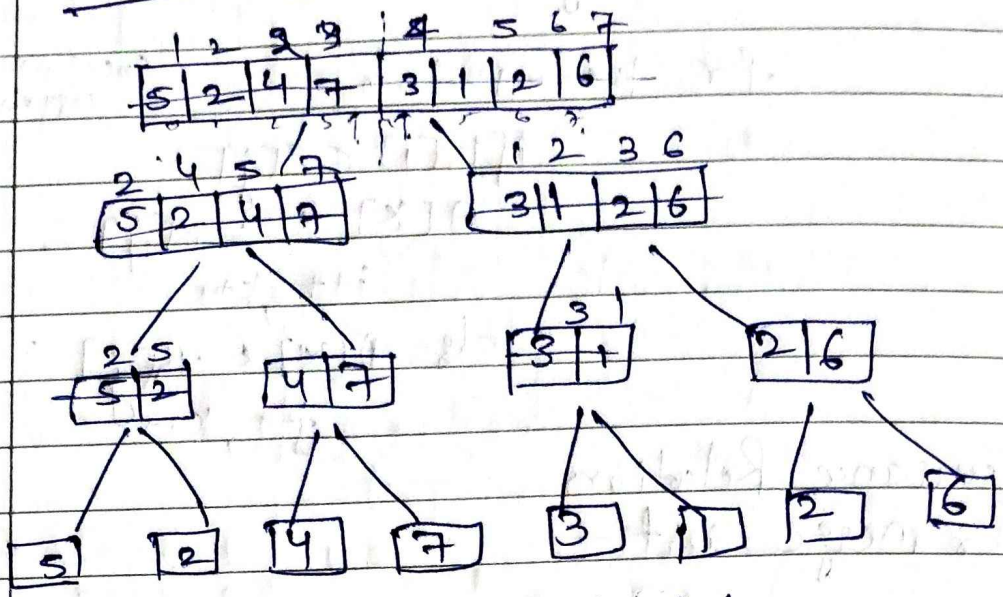
Recursive is slower ∵ maintain stack.

for low n recursive is better.

as high n recursion tree ↑. ⇒ take time. $2n-2$

22/08/22

MERGE SORT



Merge-Sort (A, p, r) (p: starting index, r: last index)

p=r single element

if (p < r)

then q ← ⌊(p+r)/2⌋ mid of q.

rec. calls Merge Sort (A, p, q)

rec. calls merge Sort (A, q+1, r)

Merge (A, p, q, r)

Mergesorting // In-place Algo

Merge (A, p, q, r)

$n_1 \leftarrow q-p+1$

$n_2 \leftarrow r-q$

3-0+1=4.

left side array side

p=1 in this ques.

O(n log n)

for i ← 1 to n_1

$L[i] \leftarrow A[p+i-1]$

for j ← 1 to n_2

$R[j] \leftarrow [q+j]$

$L[n_1+1] \leftarrow \infty, R[n_2+1] \leftarrow \infty$

// Adding // Sentinal

```
i ← 1, j ← 1
for k ← p to r                    // new array copy elements
    if L[i] ≤ R[j]
    { A[k] ← R[j] }
        i++ , k++
    else A[k] ← R[j]
    { j++ , k++ }
```

Recurrance Relation for merge sort –

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

(L) (R) cost for merging

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

Substitution: $2\left(2T\left(\frac{n}{4}\right)+\frac{cn}{2}\right)+cn$

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

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

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

$$= n + cn\log_2 n.$$

$$O(n\log_2 n)$$

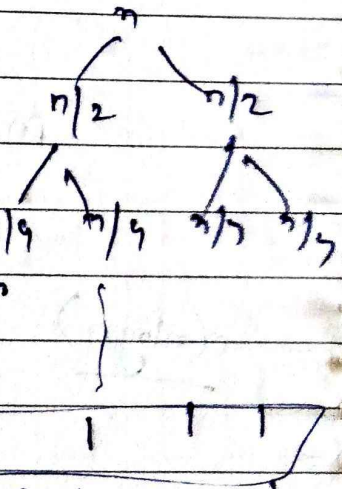
$\frac{n}{2^k}=1$

$n = 2^k$

$\log n = k\log 2$

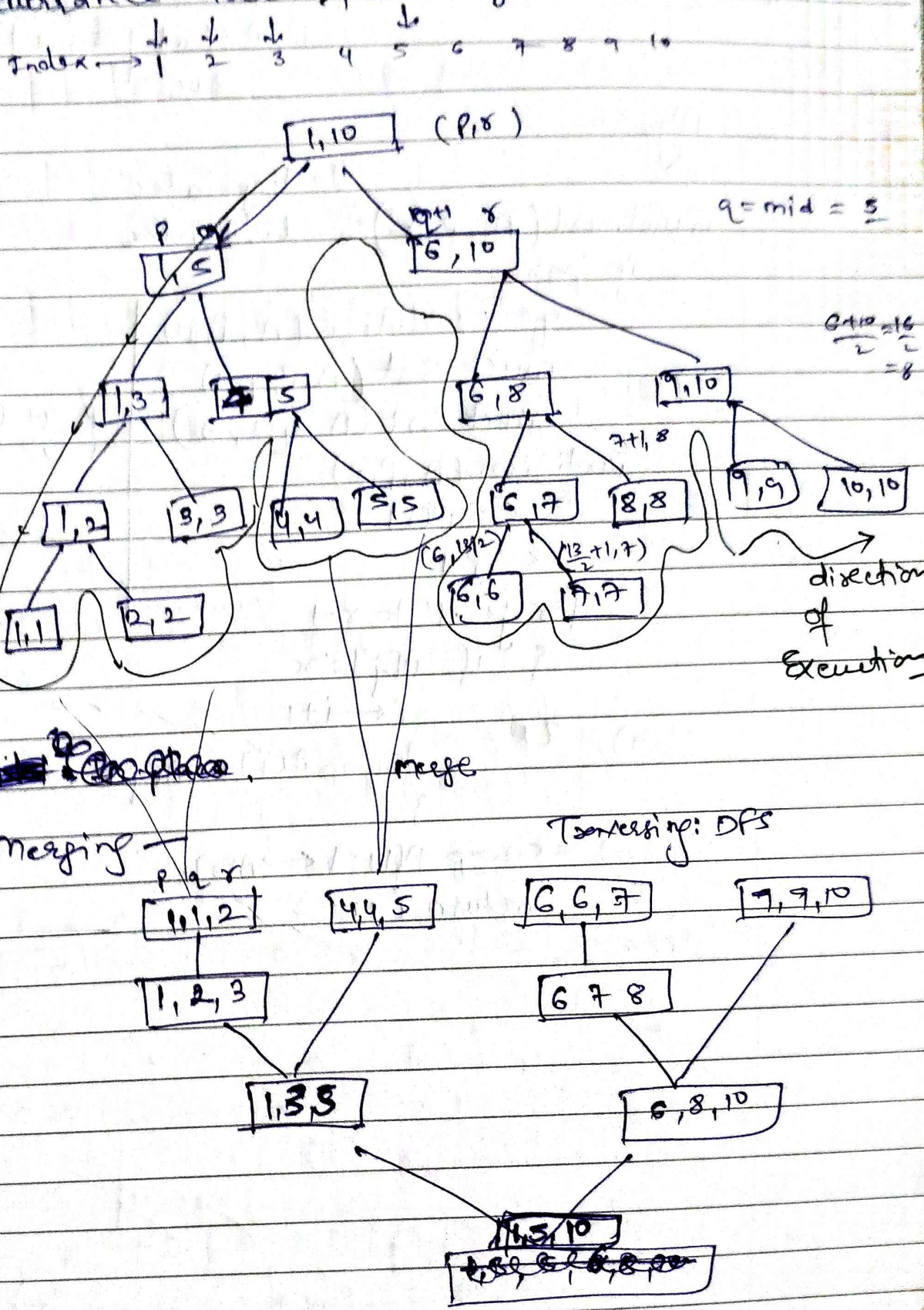
$k = \frac{\log n}{\log 2}$

$K = \log_2 n$



Merge sort → Not In place

Recurrance Tree for Merge Sort -



QUICK SORT : Choose pivot element.
at start | mid | end.

Algo :-

```
QuickSort(A, p, r)        // p: starting index, r: last index
  if p<r.
     q ← Partition(A, p, r)
     Quick Sort(A, p, q-1)
     Quick Sort(A, q+1, r)
Partition(A, p, r)
     x ← A[r]
     i ← p-1
     for j ← p to r-1.      // last is pivot
     { if A[j] ≤ x
           i ← i+1
           Swap A[i] ↔ A[j]
     }
     Swap A[i+1] ↔ A[r]
     return i+1;
```

q: जहाँ divide किया है.

p q-1 (q) q+1 r

(q) already in actual position

p-1 i p 1<4 r

2	8	7	1	3	5	6	4
j	j	j	j				x

x=4

i=p-1

2<4 ✓ true ⇒ i++, A(i), A(j) swapped.

2	1	3	8	7	5	6	4
							j

swap A[i+1] ↔ A(r)

pivot

2	1	3	4	7	5	6	8

Q.

-1 p 0 1 2 3 4 5 i r-1 r

65	70	75	80	85	60	55	50	45
j	j	j	j				j	x

65<45 x

70<45 x

75<45 x

!

50<45 x

//return i+1 = 1+1 = 0

p p r

45	70	75	80	85	60	55	50	65
i	j	j	j	j	j	j	j	x

0, 0-1

0+1, 8

0, -1

1, 8

(70, 60 swap)

60	75	80	85	70	55	50	65

2nd

(75, 55)

60	55	80	85	70	75	50	65

3rd swap

(80, 50)

60	55	50	85	70	75	80	65
		i				j	

//return i+1 = 3+1 = 4

Recurrence Relation :-

WORST CASE

$T(n) = \underbrace{T(0)}_{L} + \underbrace{T(n-1)}_{R} + cn$ (no element in left)

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

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

A.P.

$\frac{n(n-1)}{2} = O(n^2)$

Avg, Best : $n \log n$ Case

↓

Assume partitioning is balanced → some elements on left & some on right

$T(n) = 2T(\frac{n}{2}) + cn$ // Using master Theorem.

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

$f(n) = cn, \quad a = 2, \quad b = 2$

Now $n^{\log_b a} = n^{\log_2 2} = n^1 = (n)$

Case II :-

$f(n) = g(n)$

$cn = n$

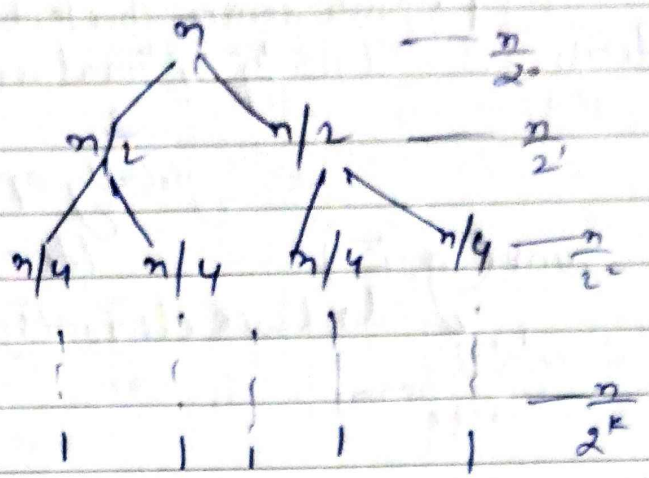
$n(c-1) = 0$

$\boxed{n=0}$ / $\boxed{c=1}$

$T(n) = \theta(n^{\log_b a} \cdot \log_2 n)$

$= \underline{\theta(n \cdot \log_2 n)}$

Recurrance Tree -



cost height

$n \times \log n = n \log n$

height of tree x cost of each level

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

$$n = 2^k \Rightarrow \boxed{\log n = k}$$

Q. Which algo has min no. of swaps if elements given in dec. order.
Convert to ascending order.

or. which algo is best for the elements given in dec. order.

GREEDY APPROACH

where we can apply -
- Knapsack problem
- Job sequencing with deadline
- optimal merge pattern
- Huffman code
- MST → Kruskal's, Prim's, Boruvka's
- Single source shortest path (Dijkstra's)

25/08/22

Sequence of decision => locally optimal sol^n

↓

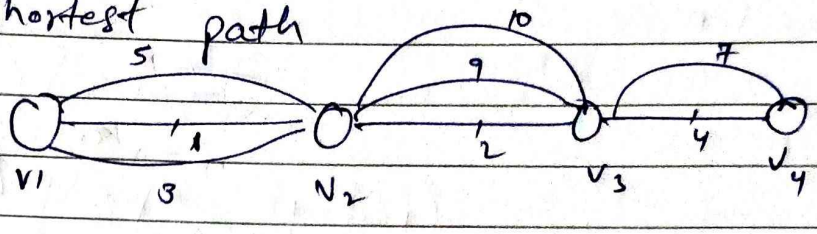
globally optimal sol^n

Ex: Largest sum(n, k) (desired result)

↳ choose elements

total size

Ex: Shortest path



Smallest path = 1+2+4

Ex: MST : Minimum Spanning Tree.
(Used greedy Approach).
Kruskal's algo, Prim's algo.

Q. Optimal storage on tape:

n programs on a tape of length L.
'i' length 'li th'

Program : I → i_1, i_2, i_3

$t_4 = l_1 + i_2 + i_3$

Time to retrieve $t_j = \sum_{k=1}^{j} l_{ik}$
j^{th} prog.

Expected mean Retrival time (MRT)

$= \frac{\sum_{j=1}^{n} t_j}{n}$

Optimally store n

minimize MRT

$$d(I) = \sum_{j=1}^{n} \sum_{k=1}^{j} l_{i_k}$$

Q. Single Tape Problem –
Ex: n=3 $(l_1, l_2, l_3) = (5, 10, 3)$ Length of program

	Retrival time	
1, 2, 3	5 + (5+10) + (5+10+3)	= 38
1, 3, 2	5 + (5+3) + (5+3+10)	= 31
2, 1, 3	10 + (10+5) + (10+5+3)	= 43
2, 3, 1	10 + (10+3) + (10+3+5)	= 41
✓ 3, 1, 2	3 + (3+5) + (3+5+10)	= 29 Ans.
3, 2, 1	3 + (3+10) + (3+10+5)	= 34

n! = 3! = 6

Brute force →

Greedy Approach : l_3 l_1 l_2
3 5 10

Q. 'n' programs, store on 'm' tapes.

13 programs on 3 tapes.

Length of 13 programs: 12, 5, 8, 32, 7, 5, 18, 26, 4, 3, 11, 10, 6.

Sort : 3, 4, 5, 5, 6, 7, 8, 10, 11, 12, 18, 26, 32

Index 1 2 3 → 13

Index 0 1 2 ✓ → 12 ✓ (used)

Using Round Robin Algo.

$T_0 \rightarrow$ 3[0] 5[3] 8[1] 12[2] 32[4]
$T_1 \rightarrow$ 4[1] 6[4] 10[3] 18[0]
$T_2 \rightarrow$ 5[2] 7[5] 11[8] 26[6]

$i \rightarrow T_{i \bmod m}$

m = no. of tapes = 3.
i = index.
$9 \% 3 = 0 \rightarrow T_0$

Algo -

```
store(n, m)
j ← 0
for (i = 0 to n-1)
{
    write("append prog. 'i' to tape 'j'")
    j = j mode m
}
```

merge sort
max. of O(n log n) & O(n)
Sorting for loop.
Ans: O(n log n)

1m Complexity

6 months internship
Ans.
↓
semiconductors.

26/8/22

0/1 Knapsack Problem:

1. +ve integers $P_1, P_2, P_3 \ldots P_n > 0$ (n items)
$w_1, w_2, w_3 \ldots w_n > 0$ weight of ith items and M = total capacity of Knapsack.

P_i Profit of ith item

2. Find $x_1, x_2, x_3 \ldots x_n$, $0 \le x_i \le 1$ such that

$\sum_{i=1}^{n} P_i x_i$ is maximized.

Subject to $\sum_{i=1}^{n} w_i x_i \le M$

$x_i = 0$ not to pick that thing

$x_i = 1$ Choosen as whole. Poora uthaliy

Q. $M = 20$, $(P_1\ P_2\ P_3) = (25, 24, 15)$

$(w_1\ w_2\ w_3) = (18, 15, 10)$

	$x_1\ x_2\ x_3$	$\sum w_i x_i$	$\sum P_i x_i$	
Let	$\frac{1}{2}, \frac{1}{3}, \frac{1}{4}$	$18\cdot\frac{1}{2} + \frac{15}{3} + \frac{10}{4}$	$\frac{25}{2} + \frac{24}{3} + \frac{15}{4}$	$\frac{2}{15}$
		$9 + 5 + 2.5$	24.25	$\frac{10}{15}$ 10
		$= 16.5$		
1) max Prof. 2) min weight 3) Ratio $\frac{\max P}{\min w}$ & put in dec. order.				
	$1, \frac{2}{15}, 0$	20.	28.2	$1+0+\frac{2}{15}$
	0/18 10/15, 10/10			$\frac{17}{15} = 1.1$
	$0, \frac{2}{3}, 1$	20	31	$1, \frac{2}{15}, 0$

$\frac{P_1}{w_1} = \frac{25}{18} = 1.38$ (3) | $\frac{P_2}{w_2} = 1.6$ (1) Selected as whole. | $\frac{P_3}{w_3} = 1.5$ (2)

25 24 15
18, 15, 10 m=20

5/10 (1.5)

stating as whole

0, 1, 1/2 20 31.5

Q. P_1 P_2 P_3 [24, 15, 25] → P_3 P_2 P_1
w_1 w_2 w_3 [15, 10, 5] 15 24 25
10 15 5

$x = [x_1 \; x_2 \; x_3]$ // 0 0 0
1 0.5 0

```
Greed knapsack (m,n){
  for i=1 to n
      x[i] ← 0
  U=m
  for i=1 to n
      if (w[i] > U) then break;
      else { x[i]=10 , V=U-w[i] }
  if (i≤n)
      { x[i] = U/w[i] }
```

m=20
30, 40, 50
0 0
3/2 1 0 0

Case.
(w_1, w_2, w_3) (15, 10, 18)
⇓
43
m=50
Picks whole.
$x = [1, 1, 1]$

Time Complexity:
= (nlogn) do sorting first

02/09/22

Job Sequencing with deadlines:

$n \rightarrow$ Jobs $\qquad d_i > 0 \qquad$ deadline of Job $i = d_i$

$\qquad\qquad\qquad\qquad p_i > 0 \qquad$ Profit of item i.

Each job take one unit of time to complete.

feasible solution is $J \subseteq n$.

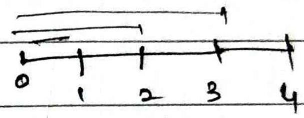
$$\max \sum_{i=1}^{n} p_i$$

Ex: $n = 4$ $\quad (P_1 P_2 P_3 P_4) = (100, 10, 15, 27)$

$\qquad\qquad (d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

All subsets of d.

Jobs	sequence	Profit Earned
(1,2)	(2,1)	110
(1,3)	1,3 or 3,1	115
(1,4)	4,1	127
(2,3)	2,3	25
(2,4)	not feasible	
(3,4)	(4,3)	42
(1)	1	100
(2)	2	10
(3)	3	15
(4)	4	27



Apply Greed - Arrange profit in desc. order.

Ex: (1,4) $\qquad$ (100, 27, 15, 10)

Q. $(P_1\ P_2\ P_3\ P_4\ P_5\ P_6) = (99, 67, 45, 34, 23, 10)$

$(d_1\ d_2\ d_3\ d_4\ d_5\ d_6) = (2, 3, 1, 4, 5, 3)$

I II III IV V VI

Max size of feasible subset is 5

Take an array of size 5. (index = deadline)

	1	2	3	4	5
deadline	2	3			
	I	II			

↓ shifting

1	2	3	4	5
III	I	II	IV	V

VI
3

Discard last job.
{∵ Can't shift no empty index left}

Q. $(P_1\ P_2\ P_3\ P_4\ P_5) = (20, 15, 10, 5, 1)$

$(d_1\ d_2\ d_3\ d_4\ d_5) = (2, 2, 1, 3, 3)$

I II III IV V

max size of possible subset.

ALGO SORTED SAN!

1	2	3
2	1	3
I	II	IV

discard (III) → deadline violated

discard (I) → no space left + deadline.

Not Easy to implement on array.

↳ Sol^n: $[\alpha-1, \alpha]$ (slot)

α = largest int r; $1 \le r \le d_i$

delaying ⇒ no shifting the process required.

Q. (I) (II) (III) (IV) (V) (VI)
99 67 45 34 23 10
deadline → 2 3 1 4 5 3

(III) (I) (II) (IV) (V) (VI)
(-1,0) (0,1) (1,2) (2,3) (3,4) (4,5)

discarded

occupied

occupied

move backward to check empty slot

4-1, 4	4-1, 4
(3,4)	2-1, 2
5-1, 5	3-1, 3
(4,5)	(2,3)
3-1, 3	1-1, 1
(2,3)	(0,1)

Q. I II III IV V
2, 2, 1, 3, 3 → deadline

II, V (-1,0), (0,1) (1,2) (2,3)
II I IV

discarded

occupied

(I) 2-1, 2
(1,2)

(II) 2-1, 2
(1,2)

occupied move to left.

(1,2)

(2,3) (0,1) (3,4) (4,5)

Disjoint set union.

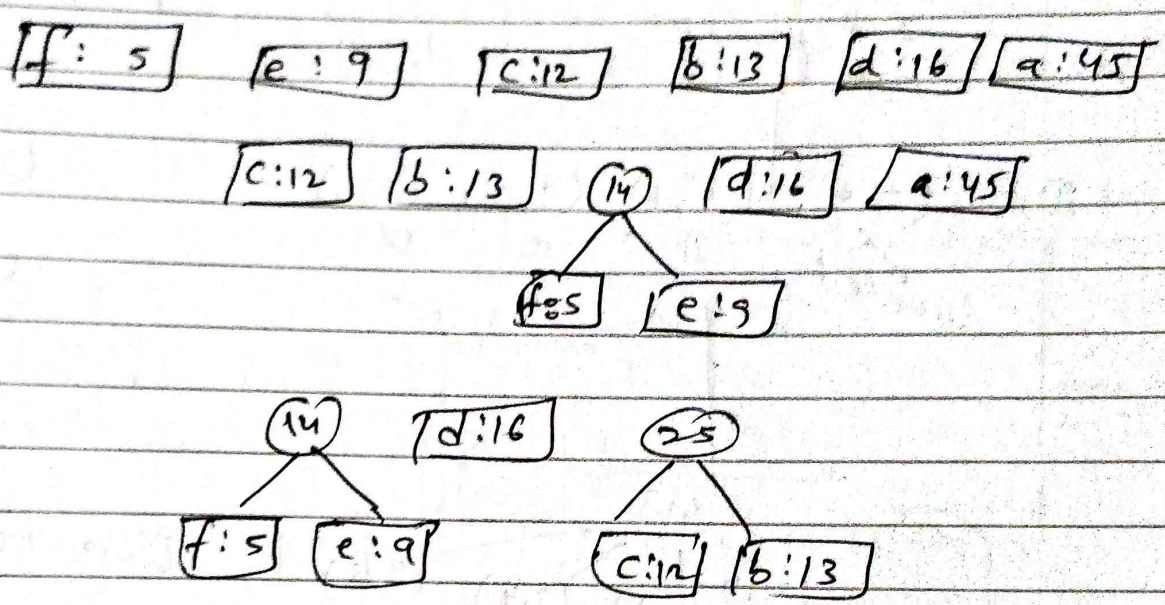
6/9/22

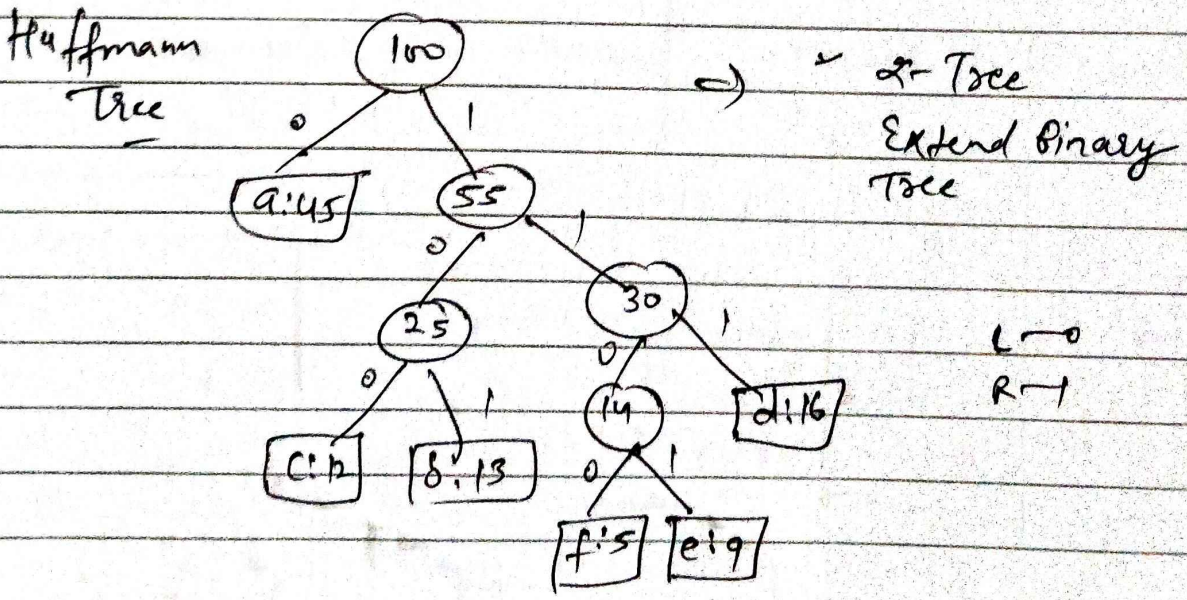
Huffman Coding: used for data compression.

	a	b	c	d	e	f	total	Repeating characters
thousand	45	13	12	16	9	5	100	

Prefix code: ~~no other code have prefix.~~
none of code is prefix of other.

Q.





→ 2-Tree
Extend Binary Tree

L → 0
R → 1

nodes having higher frequency => near to root. Ass: Extract-min

$b = 101$, $c = 100$, $d = 111$, $e = 1101$, $f = 1100$

* Bits required to encode whole tree –

$$B(T) = \sum_{c \in C} c.freq * d_T(c)$$

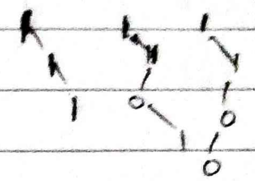
C = set of characters

{ 4.5 – f
{ 4.9 – e

depth frequency

Ex: def
111 1101 1100

starting from root
↓
reach leaf
↓



Implement : min priority queue

Algorithm:

Huffmann (C)

C = character set

n ← |C| // n=6

key = frequency

Q ← C

for i ← 1 to n−1

allocate a new node z.

left [z] ← x ← Extract_min (Q) // logn

right [z] ← y ← Extract_min (Q)

f(z) = f(x) + f(y)

Insert (Q, z) // inserted back to min priority Q.

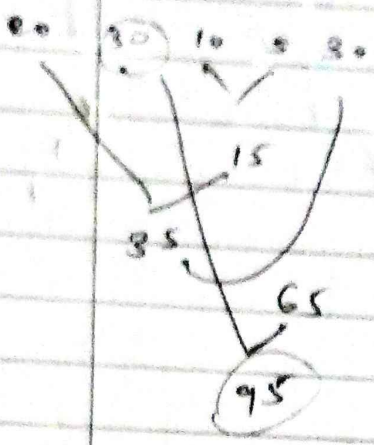
Return Extract_min (Q)

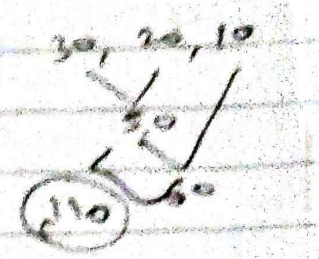
Complexity: nlogn.

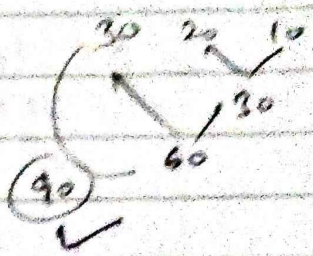
Optimal Merge Pattern: O(n+m)

x_1, x_2, x_3
30 20 10 records

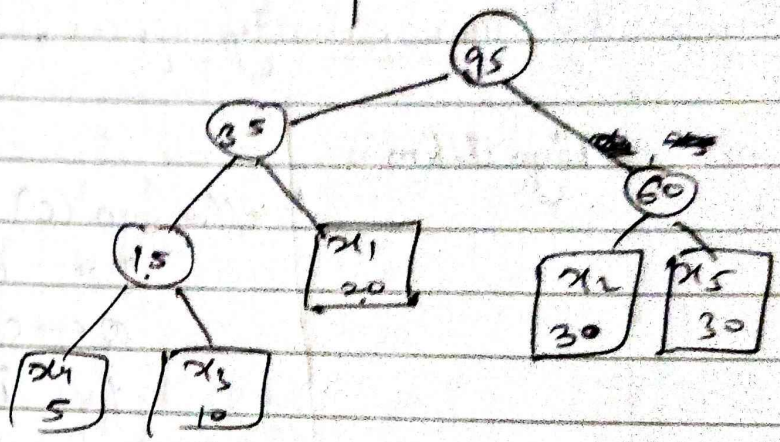
Q. $(x_1\ x_2\ x_3\ x_4\ x_5)$
20 30 10 5 30







125



$\sum_{i=1} d_i q_i$ = depth
$20\times2 + 30\times2 + 10\times3 + 5\times3 + 30\times2$
$= 205$

Total moves = $95+35+60+15$
$= 205$

8/9/22

* Minimum Spanning Tree:

Undirected connected graph G

Spanning Tree $\subseteq$ G ← subgraph

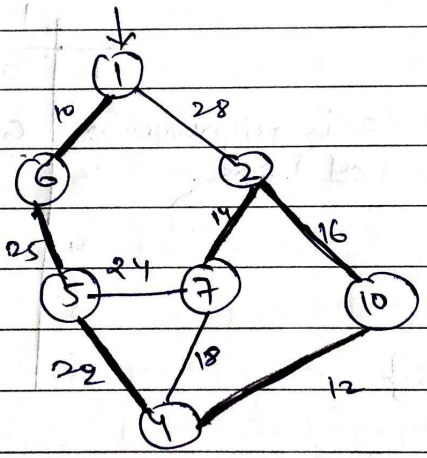
Tree

Contains all vertices of G.

- Tree ↓ Can't have cycle
- Graph can have cycle

* One can have more than one spanning trees.
Minimum Spanning Tree ⇒ least weighted spanning Tree.

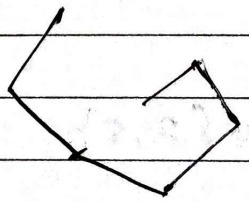
1) Prims Algo
2) ~~Dij~~ Kruskal's



PRIMS ALGO — choose min edge

after selecting 18 cycle is formed ⇒ ~~don't~~ select it.

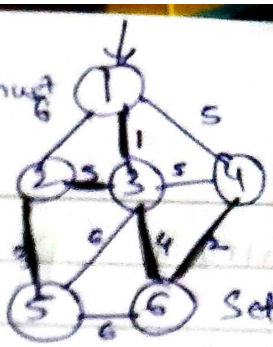
18, 24, 28 ⇒ form cycle so discard.



⇒ minimum ST.

Limits: where to backtrack, how to detect a cycle.

Equal weight → more than one MST
But final weight must be equal.



for edges
Set T = NULL
initial root node

Algo: Prims (G)

```
T = ϕ
U = {1}
while U ≠ V
  { let (u,v) (lowest cost) such that
    u∈U & v∈(V-U)
    T ← T ∪ {u,v}
    U ← U ∪ {v}
  }
```

G = E, V
E = edges {6,1,5...
V = vertices {1,2,3,4,5,6}

Imple": Adjacency matrix, Adjacency list.

	1	2	3	4	5	6
1	0	6	1	5	0	0
2	6	0	5	0	3	0
3						
4						
5						
6						

2 is neighbour of 1, 3, 5.

U = {1}
V - U = {2,3,4,5,6}

1 2
1 3 lowest
1 4
3 2
3 4 lowest
3 5
3 6

U = {1, 3, 6}

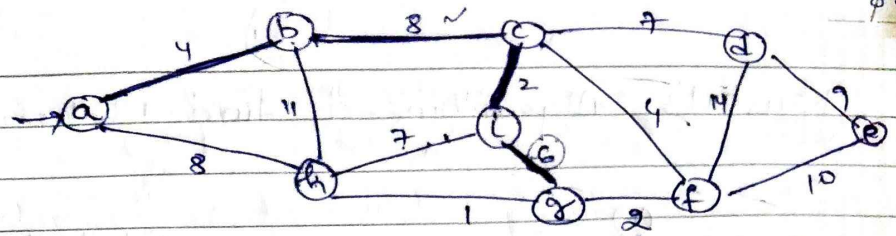
V-U = {2, 4, 5}

U = {1, 3, 4, 6}, V-U = {2, 5}

{1, 3, 4, 6, 2} {5}
{1, 3, 4, 6, 2, 5} { }

4, 80

4,



$u = \{a\}$ ← u

$v = \{a, b, c, d, e, f, g, h, i\}$

1) $v \in (v-u) = \{b, c, d, e, f, g, h, i\}$

lowest cost
a – b ✓
a – h

$T = \{a, b\}$

$u = \{a, b\}$ ↑u

2) $v - u = \{c, d, e, f, g, h, i\}$ ← v

lowest cost
b – c ✓
b – h

$T = \{a, b, c\}$

$u = \{a, b, c\}$ ↑u

3) $v - u = \{d, e, f, g, h, i\}$ ← v

c – d
c – f
c – i ✓

$T = \{a, b, c, i\}$

$u = \{a, b, c, i\}$ ↑u

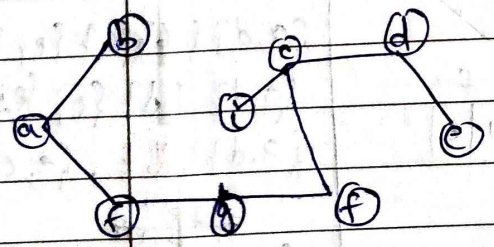
4) $v - u = \{d, e, f, g, h\}$

$T = \{a, b, c, i,$

$u = \{a, b\}$

a – h – 8
b – h – 11
b – c – 8

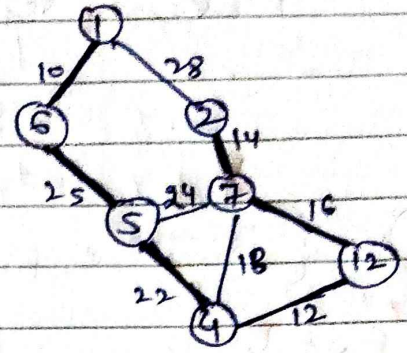
can choose any one of them
⇒ 2 MST.



⇒ 2nd

09/09/22

Kruskal's Algorithm to find MST -



Select 1st mini edge then 2nd mini edge ... + avoiding cycle

Kruskal (G, w)

T ← ϕ

for each vertex $v \in V[G]$

make-set(v)

Sort edges into increasing order by w.

for each $(u,x) \in E(G)$ taken in { increasing order

if find(u) ≠ find(v)

T ← T ∪ {u, v}

Union (find(u), find(v))

}

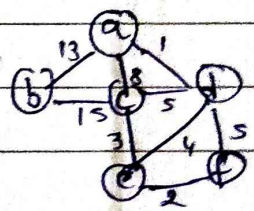
return T.

{ using Disjoint set & find algo

	a	b	c	d	e	f
P	-1	-1	-1	-1	-1	-1

Parent set

{a} {b} {c} ... {f}



{a,d} {b} {c} {e} {f}

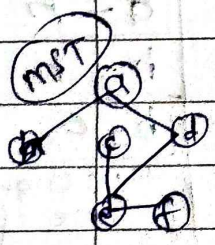
{a,d} {b} {c} {e,f}

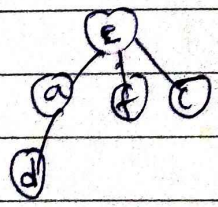
{a,d}, {b}, {e,c,f}

~~{a,d} {b}~~

{e,a,c,d,f} {b}

{e,a,c,d,f,b}





	a	b	c	d	e	f
P	e	-1	e	a	-1	e

Application of MST-

Networks ⇒ no duplicate packets

⇓

no Overhead

Broadcast

↓

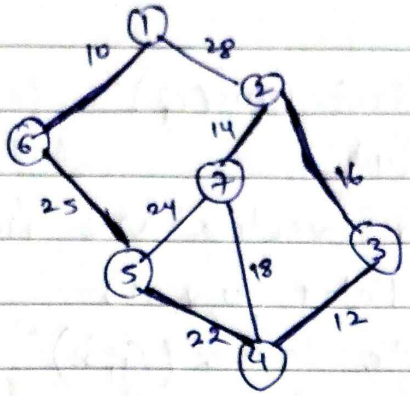
packet send to all roots

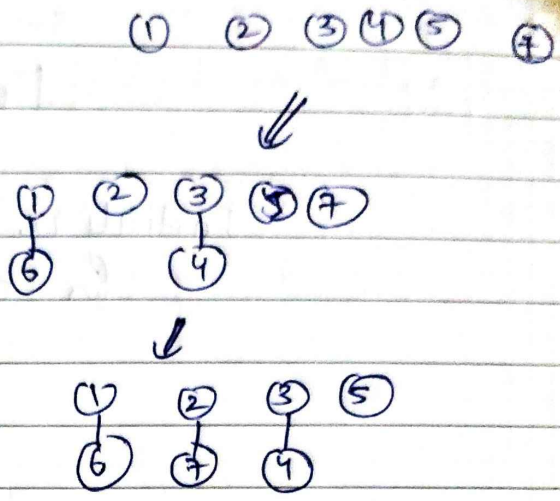
Multicast

↓

Selected roots

Ex:





12/09/22

MST: Dijkstra Algorithm / Single source shortest path

Initialize Single Source (G, S)

$S \leftarrow \phi$

Min priority Queue → $Q \leftarrow V[G]$

while $Q \neq \phi$

$u \leftarrow$ Extract-min(Q)

$S \leftarrow S \cup \{u\}$ // set

for each vertex $v \in Adj[u]$ (neighbours)

Relax (u, v, w)

Initialize Single source (G, S)

for each vertex $v \in V[G]$

$d[v] \leftarrow \infty$

$\pi[v] \leftarrow$ Nil (Predecessor)

$d[s] \leftarrow 0$

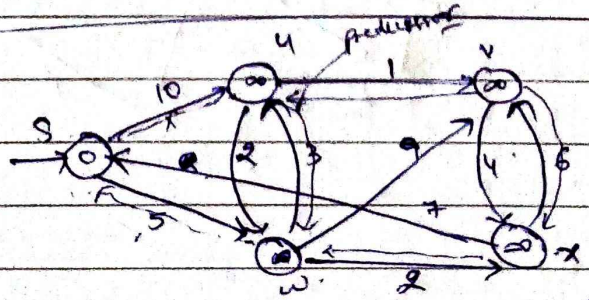
Relax (u, v, x)

if $d[v] > d[u] + w[u, v]$

$d[v] \leftarrow d[u] + w[u, v]$

$\pi[v] \leftarrow u$

Q.



final

d	0	8	9	5	7
π	0	w	u	s	w
	s	u	v	w	x

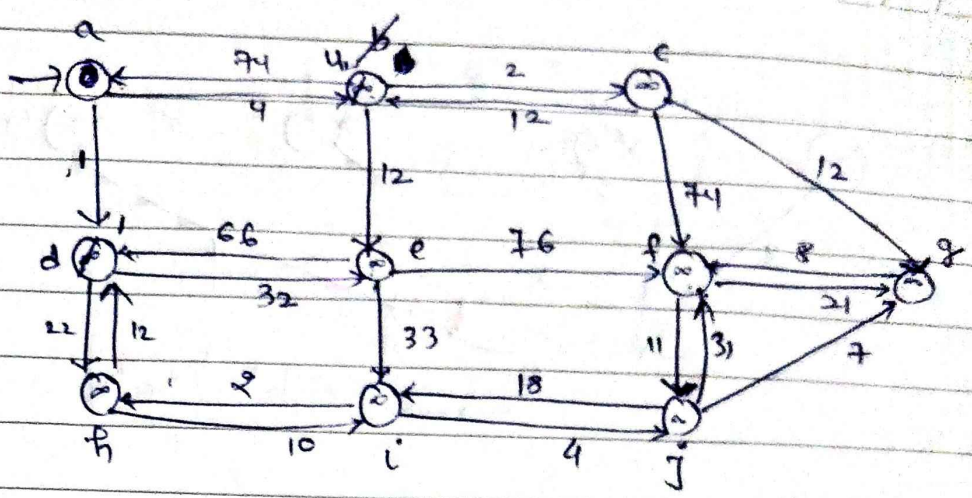
	s	u	v	w	x
d =	0	~~∞~~ 10	∞	~~∞~~ 5	∞
	s	u	v	w	x
π =	0	s	0	s	0

predecessor (previous node)

Dijkstra fail for -ve weight

Complexity = ?

Q.



	a	b	c	d	e	f	g	h	i	j
d =	0	∞	∞	∞	∞	∞	∞	∞	∞	∞

$d(u) + w(u,v) < d(v)$

$0 + 4 < \infty$

$4 < \infty$ ✓

$0 + 1 < \infty$

$1 < \infty$ ✓

$4 + 2 < \infty$

$4 + 24 < 0$ ✗

	a	b	c	d	e	f	g	h	i	j
π =	0	a	a	a	b	b	b	b	b	b

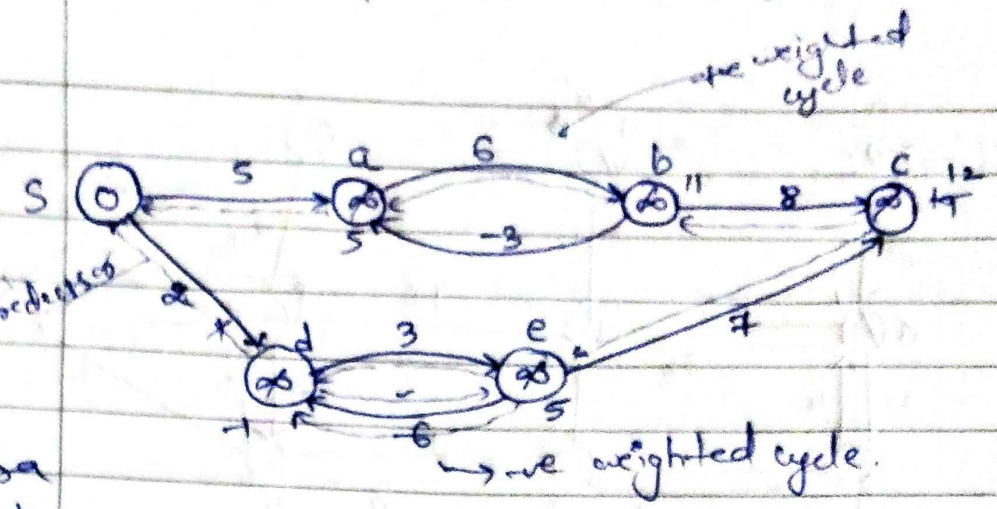
a b c

→ (0) (4) (2)

d (1) e (6) f (20) g (18)

h (23) i (33) j (37)

Exam: upto Dijkstra Algo.



→ -ve weighted cycle.

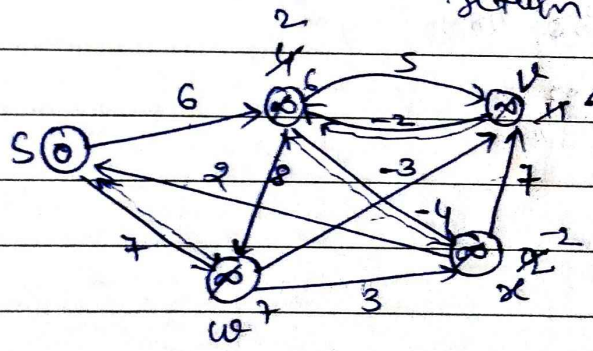
Dijkstra fails for -ve weight → Internet → -ve weighted cycle

dijkstra
min priority queue

Bellman-ford Algo (G, w, s) (graph, weight, source)

Initialize-Single Source (G, S)
for i ← 1 to |V[G]| - 1
{ for each edge (u,v) ∈ E[G]
relax(u,v,w)
}
for each edge (u,v) ∈ E[G]
if d[v] > d[u] + w[u,v]
return false
return true.

Q.



u v
~~u x~~
u w
v u
w ~~x~~
~~w x~~
x s
x v
s y
s u

Next: Floyed Warshell