Section - A

(1) Determine Θ bounds of the following relation

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

Master's Theorem

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

$a \geq 1$ = No. of subproblems

$b > 1$, $\frac{n}{b}$ = size of the subproblem.

Consider $f(n) = \Theta(n^K \log^P n)$.

We will find 2 values

(1) $\log_b a$

(2) K

Case-1 If $\log_b a > K$ $\Rightarrow$ T.C $= \Theta(n^{\log_b a})$.

Case 2 If $\log_b a = K$ then

(a) If $P > -1$ $\Rightarrow$ $\Theta(n^K \log^{P+1} n)$

(b) If $P = -1$ $\Rightarrow$ $\Theta(n^K \log\log n)$

(c) If $P < -1$ $\Rightarrow$ $\Theta(n^K)$

Case-3 If $\log_b a < K$ then

(a) If $P \geq 0$ then $\Theta(n^K \log^P n)$

(b) If $P < 0$ then $\Theta(n^K)$.

$\therefore$ $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$. ← Question

$a = 2 \quad b = 2$

(1) $\log_b a = \log_2 2 = 1$

$n^K \log^p n = n \log n$.

$\Rightarrow \quad K = 1 \quad p = 1$

$\therefore \log_b a = \log_2 2 = 1 = K \Rightarrow$ case 2

And also $p > -1$

$T.C = \Theta(n^K \log^{p+1} n) = \Theta(n^1 \log^{1+1} n)$

$= \Theta(n \log^2 n)$.

Ans = ~~$\Theta(n \log^2 n)$~~ $\Theta(n \log^2 n)$.

Set A

(2) Evaluate the prefix expression

+ * 2 3 - 5 1

. Scan from right to left

$\Rightarrow$ + * 2 3 [5-1]

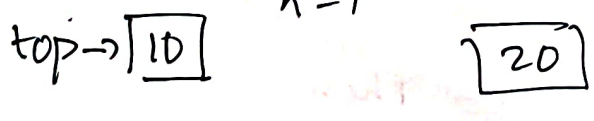
$\Rightarrow$ + * 2 3 [4]

$\Rightarrow$ + [2*3] [4]

$\Rightarrow$ + [6] [4] = 10.

Set A

(3)




```
def push(self, value):
    new_node = Node(value)   <- Assume people may not write this
    > new_node.next = self.top
Ptr assign > self.top = new_node
    > self.height += 1
```

DSA - Ans Key

Sect. A
Q.4 Formula

- $T(n)$ = Catalan No = $\frac{2nCn}{(n+1)}$ = max Binary Tree
- $T(n)$ = catalan no for labelled = $n! \times \frac{2nCn}{(n+1)}$
- $T(n) = \sum_{i=1}^{n} T(i-1) \times T(n-i)$ recursive
- $(h+1) \leq$ Nodes $\leq 2^{h+1} - 1$
  ← If height is given
- $\log_2(n+1) - 1 \leq$ Height $\leq (n-1)$
  ← If nodes are given

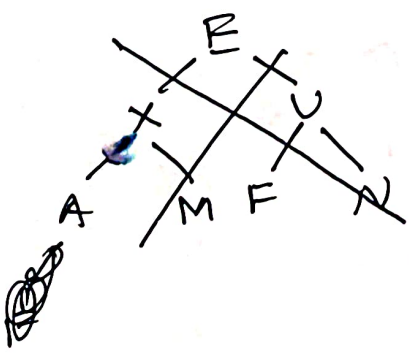
$n = 15$

$h_{min} = \log_2(15+1) - 1 = \log_2 16 - 1 = 4 - 1 = 3$

$h_{max} = 15 - 1 = 14$

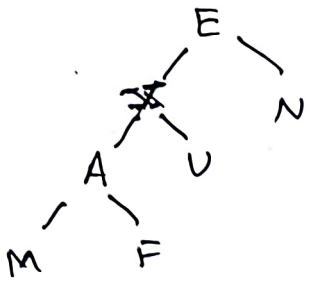
$3 \leq$ Height ≤ 14

Sect A
Q5



Preorder : EXAMFUN
~~Inorder~~

Q.5)

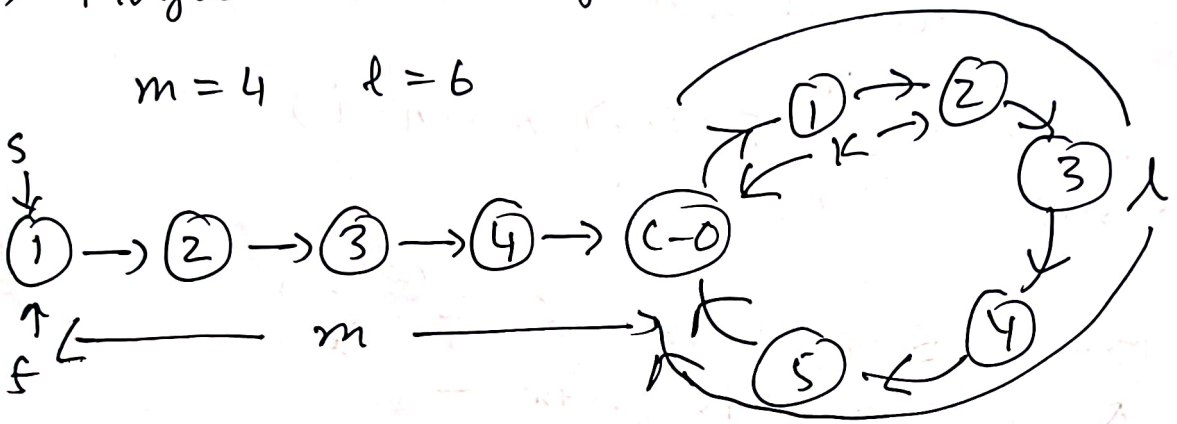


Preorder : EXAMFUN

Inorder: MAFXUEN

Section :- B

(1) Floyd Detection cycle with 10 nodes



s = slow pointer => moves by 1

f = fast pointer => moves by 2.

. If slow ptr meets fast pointer, then the linked list has a cycle.

s	f	s_d	f_d	s%l	f%l	(f_d - s_d) abs_gap.
1	1	0	0	—	—	0
2	3	1	2	—	—	1
3	C-0	2	4	—	0	2
4	2	3	6	—	2	3
C-0	4	4	8	0	4	4
1	C-0	5	10	1	0	5
2	2	6	12	2	2	6

Presence of cycle – slow & fast pointer meets.

(i) abs_gap % l = 0

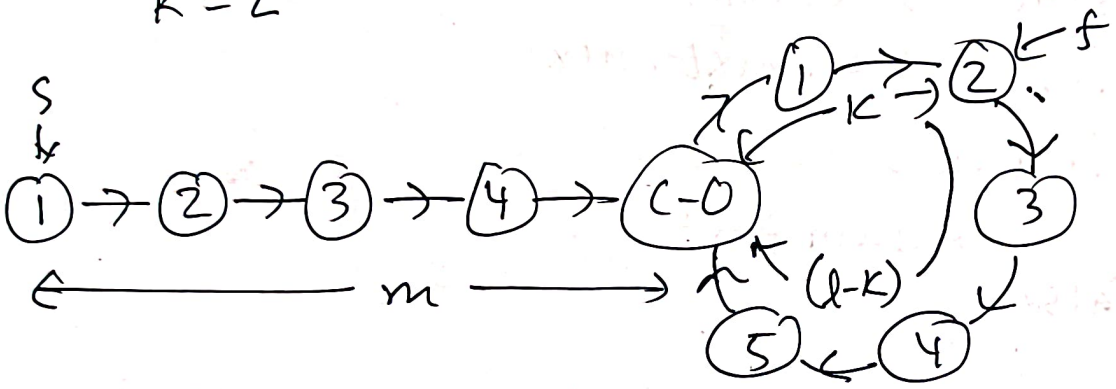
↓ ↓

6 % 6 = 0

(ii) s % l = f % l = 2

The position where they meet = K from 1st node of the cycle

K = 2



Now move s = s.next, f = f.next → they should meet at C-0 → starting point of the cycle

slow

$m + c_1 \times l + K$

$c_1 \in Z$

fast

$m + c_2 \times l + K$

$c_2 \in N$

$\therefore m + c_2 \times l + K = 2(m + c_1 l + K)$

$\Rightarrow (c_2 - 2c_1) \times l - K = m$

$\boxed{(int) \times l - K = m}$

m → l-K, 2l-K, 3l-K, ⋮

codes

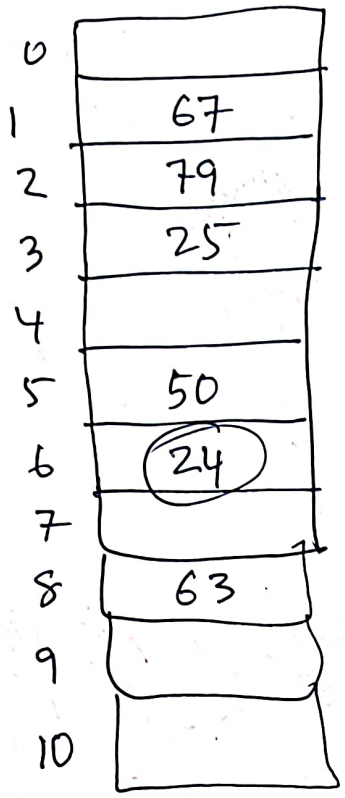
(1) has_loop
(2) detect_first_node_cycle

```
def has_loop (self):
    if self.head is None
        return False
    slow = self.head
    fast = self.head

    while fast is not None and fast.next is not None:
        slow = slow.next
        fast = fast.next.next
        if (slow == fast)
            return fast
    return False

def detect_first_node_cycle (self, meeting_point):
    slow = self.head
    fast = meeting_point

    while slow != fast:
        slow = slow.next
        fast = fast.next
    return (slow)
```

→ This will be (C-O) in our case

Section B

Q. 2.(a)



Keys = 63, 50, 25, 79, 67, 24

$h_1(K) = K \% 11$

$h_2(K) = 1 + K \% 7$

$h_1(K,i) = h_1(K) + i \cdot h_2(K)$.

$h_1(24) = 2 \Rightarrow$ collision

$h_2(24) = 1 + 24 \% 7 = 1 + 3 = 4$.

$\therefore h(24,1) = 2 + 1 \cdot 4 = 6$.

(b) Primary Clustering → Linear Probing

collision occurs → algo looks for next slot → Creates long contiguous blocks of occupied slot

Search + Insertion ↑ time

ex:- Keys = 12, 22, 32, 42

Size (H.T) = 10 ↑ $h(K) = K \% 10$

Primary cluster.

Secondary Clustering → Quadratic Probing.

$h(K,i) = h(K) + i^2$.

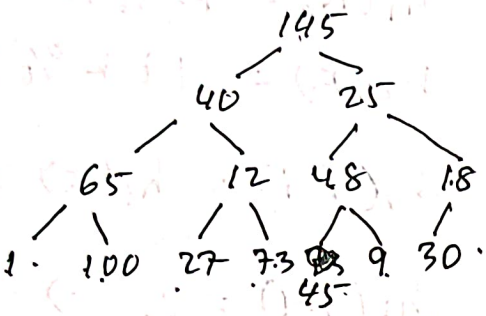
- While it pushes the Keys distance apart, it still follows the same probe sequence

ex:- Keys = 12, 22, 32, 42 can create secondary clustering as well.

3. (a) Construct a min Heap by applying Heapify procedure

145, 40, 25, 65, 12, 48, 18, 1, 100, 27, 73, 45, 9, 30

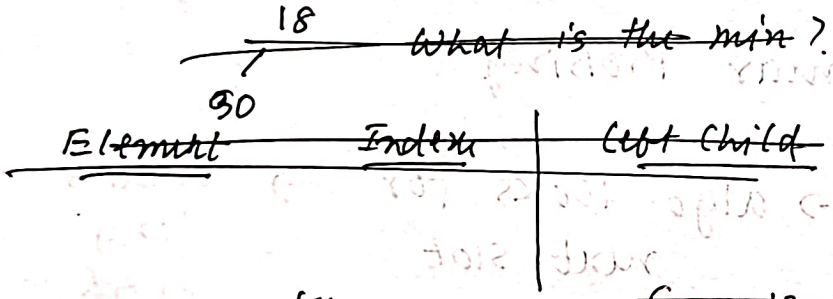
using bottom-up Heapify

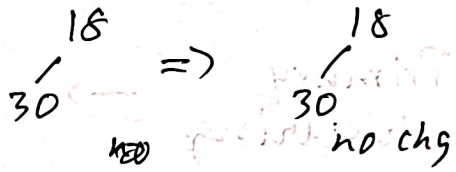


Heap => It is a complete B.T

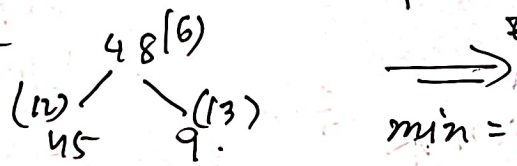
145	40	25	65	12	48	18	1	100	27	73	45	9	30
1	2	3	4	5	6	7	8	9	10	11	12	13	14

(1)





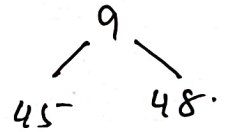
(2) Max 48 (6)



~~6 = 12, 13.~~

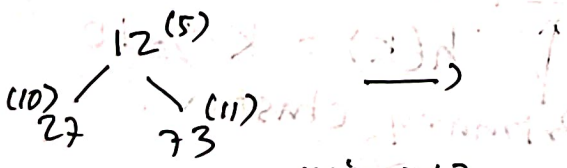
min = 9.

=>

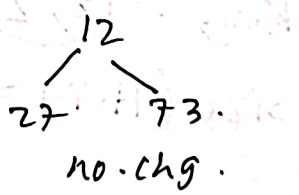


swap (48, 9)

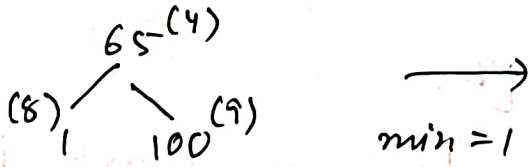
(3)



min = 12

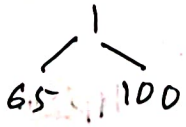


(4)

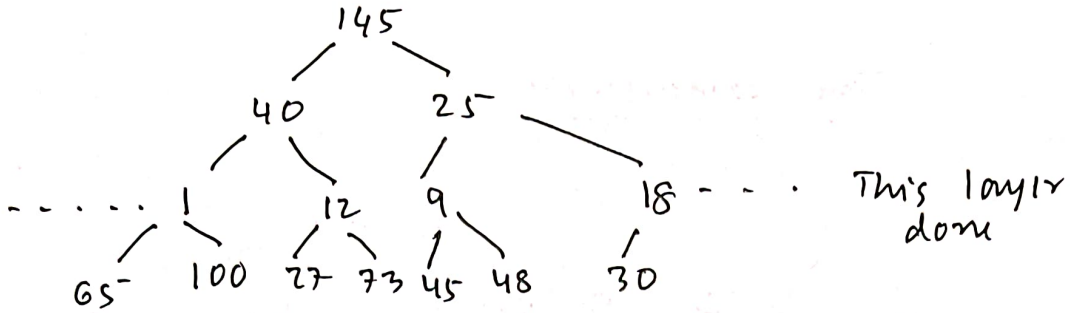


min = 1

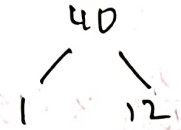
swap (65, 1)



Intermediate

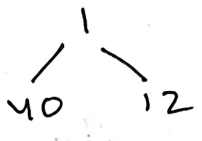


(1)

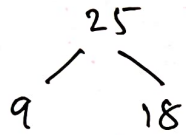


$\Rightarrow$ min = 1

swap (40, 1)



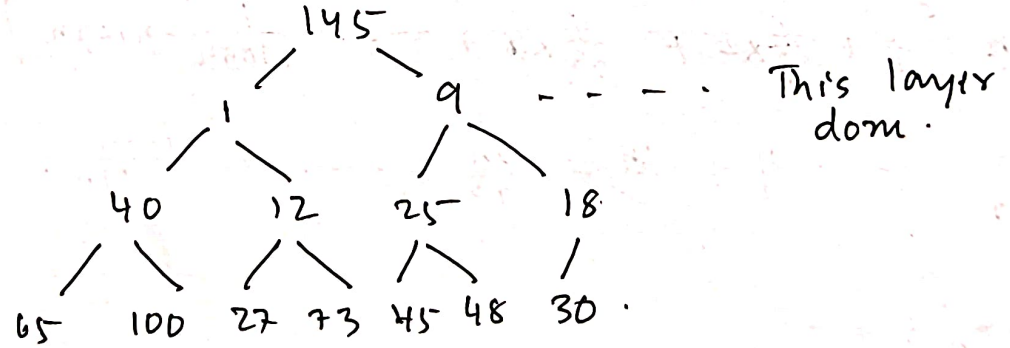
(2)

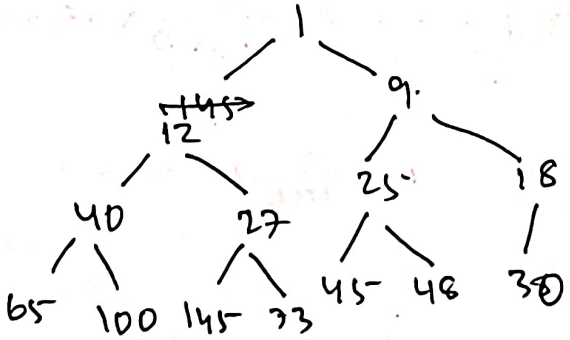


$\Rightarrow$ min = 9

swap (25, 9)

Intermediate



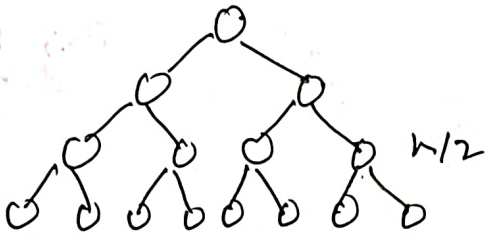


Final : 1, 12, 9, 40, 27, 25, 18, 65, 100, 145, 73, 45, 48, 30.

→ (P.T.O)

Final : $\frac{1}{1}, \frac{12}{2}, \frac{9}{3}, \frac{40}{4}, \frac{27}{5}, \frac{25}{6}, \frac{18}{7}, \frac{65}{8}, \frac{100}{9}, \frac{145}{10}, \frac{73}{11}, \frac{45}{12}, \frac{48}{13}, \frac{30}{14}$.

Time complexity



Total swap

$$S = \frac{n}{2^0} \times 0 + \frac{n}{2^1} \times 1 + \frac{n}{2^2} \times 2 + \cdots + \frac{n}{2^{\log n}} \times \log n.$$

$\frac{n}{2^0}$ ↗ no. of elements at that level

0 ↑ no. of swaps.

$$S = \frac{n}{2} + \frac{n}{2^2} \times 2 + \cdots + \frac{n}{2^{\log n}} \times \log n.$$

$$S = n\left[\frac{1}{2} + \frac{1}{2^2} \times 2 + \frac{1}{2^3} \times \cdots + \frac{1}{2^{\log n}} \times \log n\right]$$

$$\Rightarrow S = \frac{n}{2} + \frac{2n}{2^2} + \frac{3n}{2^3} + \cdots + \frac{\log n}{2^{\log n}}$$

$$\Rightarrow \frac{1}{2} S = \cdots$$

$$S = \frac{n}{2^1} \times 1 + \frac{n}{2^2} \times 2 + \frac{n}{2^3} \times 3 + \cdots + \frac{n}{2^{(\log n - 1)}} \times (\log n - 1) + \frac{n}{2^{\log n}} \times \log n \quad -(1)$$

$$\Rightarrow \frac{1}{2} S = \frac{n}{2^2} \times 1 + \frac{n}{2^3} \times 2 + \frac{n}{2^4} \times 3 + \cdots + \frac{n}{2^{\log n}} (\log n - 1) + \frac{n \log n}{2 \cdot 2^{\log n}} \quad -(2)$$

(1) - (2)

$$\Rightarrow \frac{1}{2} S = \left[\frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^{\log n}}\right] - \frac{n \log n}{2^{\log n} \times 2}$$

$$\Rightarrow \frac{1}{2} S = \frac{n}{2}\left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{\log n - 1}}\right) - \frac{n \log n}{2 \times 2^{\log n}}.$$

$$\Rightarrow \frac{1}{2}S = \frac{n}{2}\underbrace{\left(1+\frac{1}{2}+\frac{1}{2^2}+\cdots+\frac{1}{2^{(\log n-1)}}\right)}_{x} - \frac{n\log n}{2.2^{\log n}}.$$

$$x \Rightarrow \frac{a(1-r^n)}{1-r} = \frac{1\left(1-\left(\frac{1}{2}\right)^{\log n}\right)}{1-\frac{1}{2}} = 2\left[1-\left(\frac{1}{2}\right)^{\log n}\right]$$

$a = 1 \quad r = \frac{1}{2}$

$n = \log n$

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

Note $2^{\log n} = n$

$$= 2\left(1-\frac{1}{n}\right).$$

$$\Rightarrow \frac{1}{2}S = \frac{n}{2} \times 2\left(1-\frac{1}{n}\right) - \frac{n\log n}{2n}$$

$$\Rightarrow S = 2n\left(1-\frac{1}{n}\right) - \log n = 2n - 2 - \log n.$$

$\therefore \quad S = O(n).$

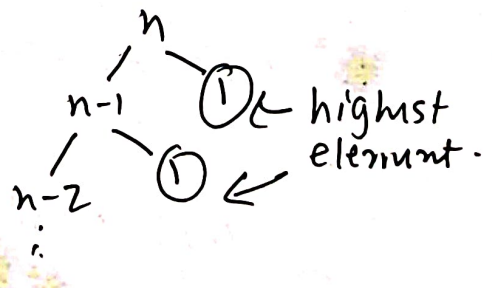
Total Swap = Time complexity = $O(n)$.

4.(a) Q.S algorithm applied on n distinct elements.

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

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

$$T(n) = T(n-1) + \underbrace{cn}_{\text{work done to break the tree.}}$$



Master's theorem

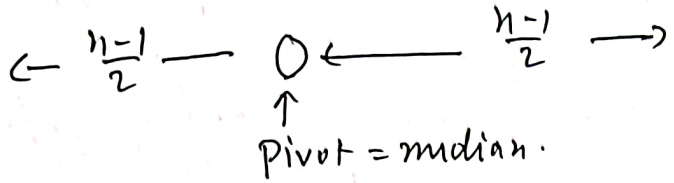
$$T(n) = a\,T(n-b) + f(n).$$

$a < 1$	$a = 1$	$a > 1$
$T(n) = O(f(n))$	$T(n) = O(n \times f(n))$	$O\left(a^{\frac{n}{b}} \cdot f(n)\right)$

Here $a = 1$

$T(n) = O(n \times O(n)) = O(c \cdot n^2) = O(n^2)$.

4.(b) Suppose the pivot is always chosen as the median.



R.R

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

$\Theta(n)$ → scan the elements

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

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$ → work done = Divide the problem + combine the result

$a \geq 1 \quad b > 1$

$f(n) = n$ → scan n elements.

Consider $f(n) = \Theta(n^K \log^P n)$.

we will find 2 values

(i) $\log_b a$

(ii) K

case 1 If $\log_b a > K$ $\quad \Theta(n^{\log_b a})$

case 2 If $\log_b a = K$ $\quad P > -1$ $\quad \Theta(n^K \log^{P+1} n)$.

$a = 2 \quad b = 2$

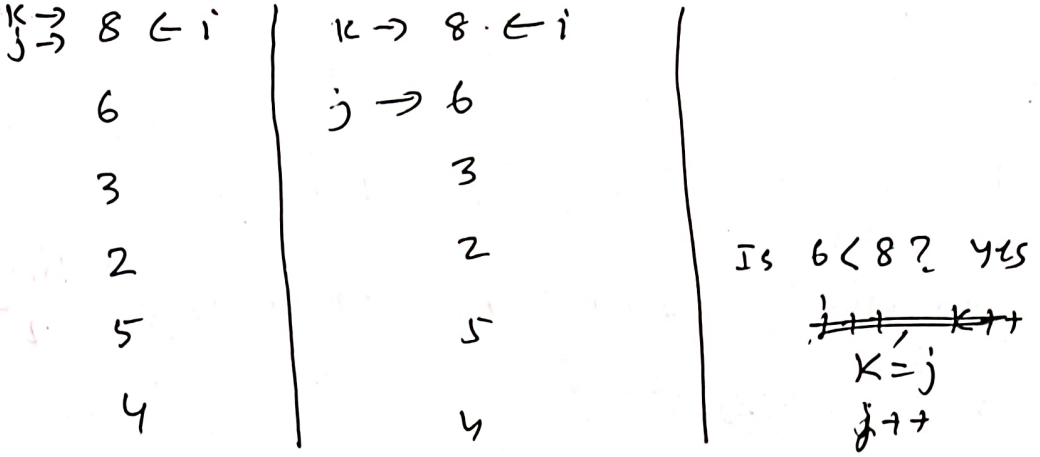
$\therefore \log_2 2 = 1$

$n^K \log^P n = n.$

$K = 1 \quad P = 0$

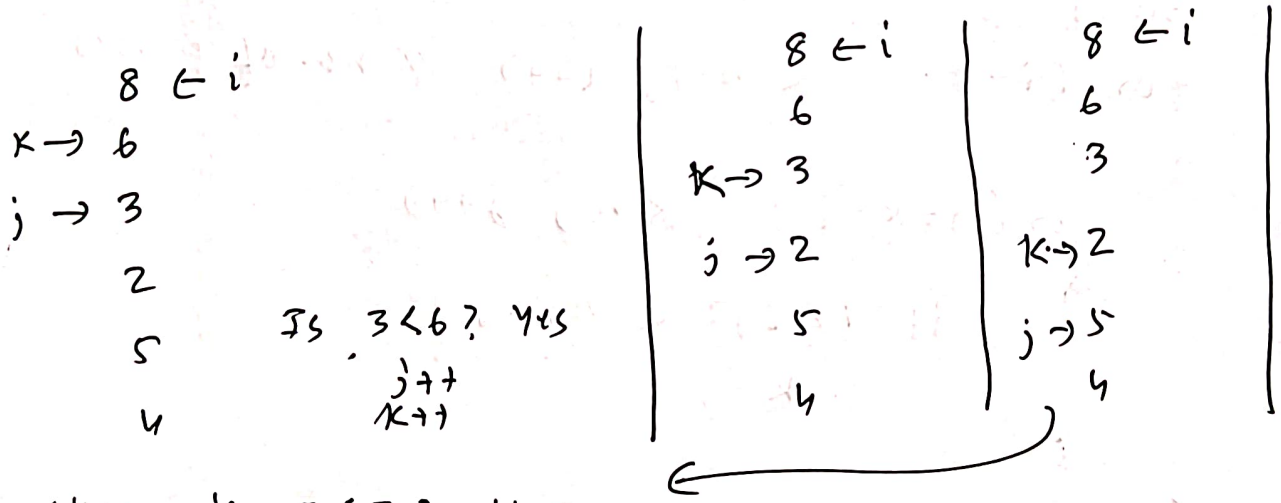
$$T(n) = \Theta(n^1 \log^1 n) = \Theta(n \log n).$$

5. (a)

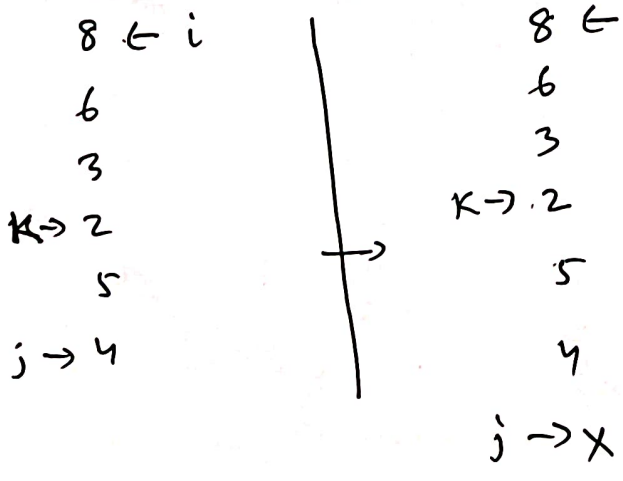


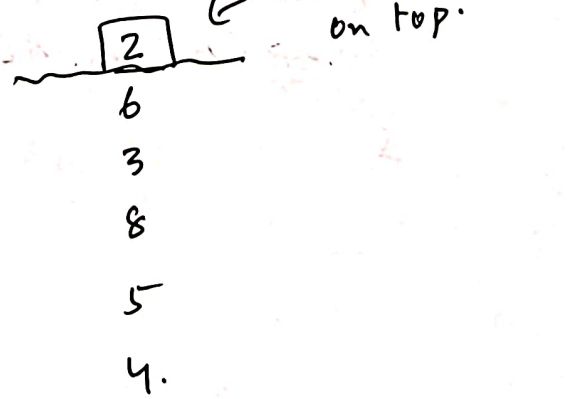
(1) k+1p i fixed
(2) use j & k to find the min.
(3) swap (i, k)

If A[j] < A[k] k=j
else j++ until array ends.



Now is 5 < 2 ? No
just j++





Pass ends here.

swap (A[i], A[k])

passes :	1	2	3	4	5
Comparisons :	5	4	3	2	1
swaps :	1	1	1	1	1

Total comparison = $1+2+3+\cdots+(n-1) = \frac{n(n-1)}{2} = O(n^2)$

swaps = $(n-1) = O(n)$

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

5. (b)

Algo

```
void selection-sort ( int A[], int n)
{
    int i
    for (i=0;  i < n-1; i++)  // no. of passes
    {
        for (j=k=i ;  j < n ; j++)
        {
            if (A[j] < A[K])
                K=j
        }
        # now K will be on the min element
        swap (A[i], A[K]).
    }
}
```

6. (a) AVL Tree with keys = 10, 20, 30, 25, 28, 27, 5

(1) 10 (0) <Insert 10>

(2) 10 (-1), 20 (0) <Insert 20>

(3) 10 (-2) ← Imbalanced, 20 (-1), 30 (0) <Insert 30> → RR Rotation → 20 (0), 10 (0), 30 (0)

(4) Insert 25

20 (-1), 10 (0), 30 (1), 25 (0)

Balanced.

(5) Insert 28

20 (-2), 10 (0), 30 (2), 25 (-1), 28 (0) → LR Rotation → 20 (-1), 10 (0), 28 (0), 25 (0), 30 (0)

balanced.

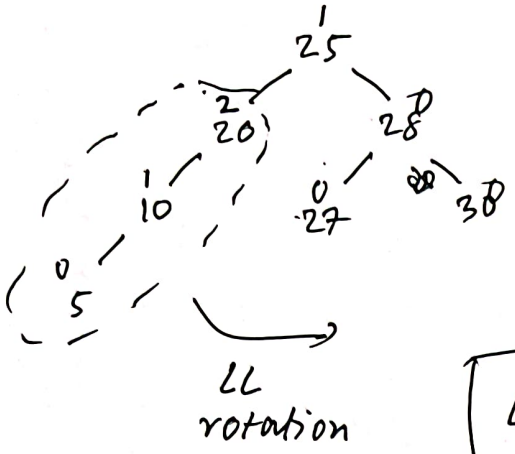
(6) Insert 27

20 (-2) ← imbalance., 10 (0), 28 (1), 25 (-1), 30 (0), 27 (0)

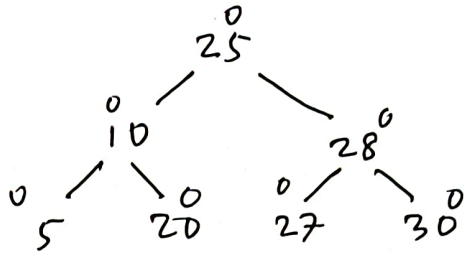
→ 25 (0), 20 (1), 28 (0), 10 (0), 27 (0), 30 (0)

balanced.

(7) Insert 5



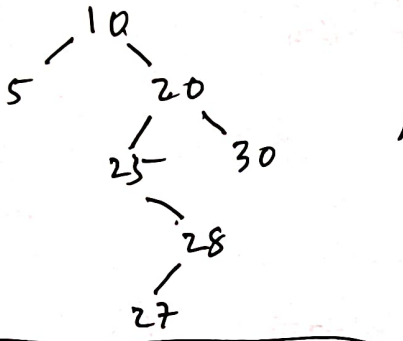
LL rotation



(balanced)

$[O(n)]_{search} = \log n$ $h = 2$

6.(b) Without AVL Tree the Tree would be



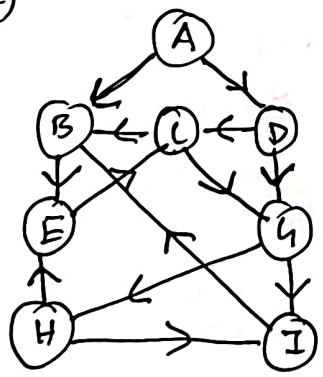
$h = 4$

$[O(n)]_{search} = n$

---

(7) BFS

a & b

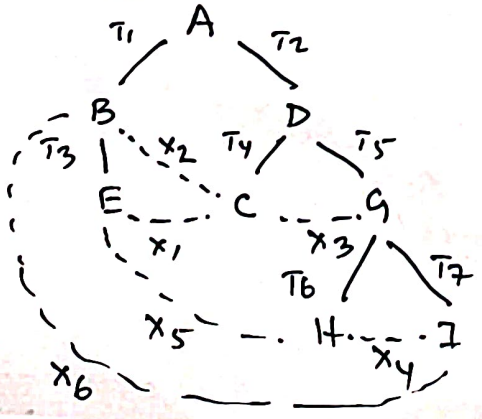


visit	neighbour	Queue
A	B, D	B, D.
B	E	D, E
D	C, G	E, C, G.
E	C,	C, G.
C	B, G	G.
G	H, I	H, I.
H	I, E	I
I	B	

BFS path = A; B, D, E, C, G, H, I

Same as Level order.

BFS Spanning Tree

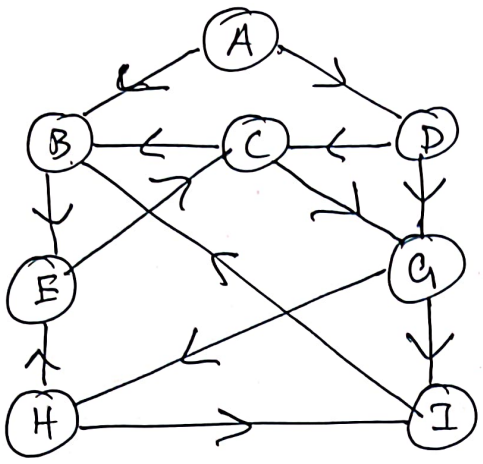


Tree Edges = (A, B), (A, D), (B, E), (D, C) (D, G), (G, H), (G, I).

Cross Edges = (E, C), (B, C), (C, G), (H, I) (B, I).

(8) (a) & (b)

DFS Traversal



Tree

(A, B), (A, D), (B, E), (E, C)
(C, G), (G, I), (G, H)

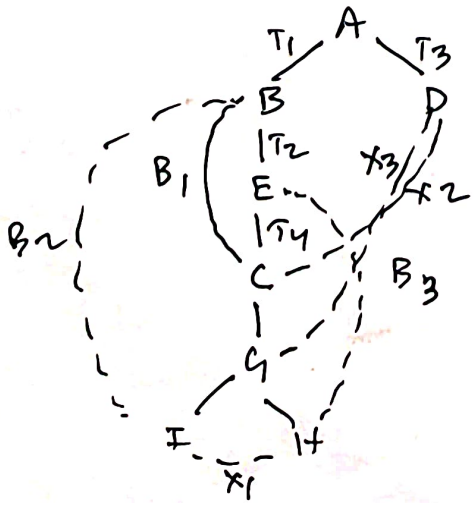
Back

(C, B), (I, B), (H, E)

Cross

(H, I), (D, C), (D, G).

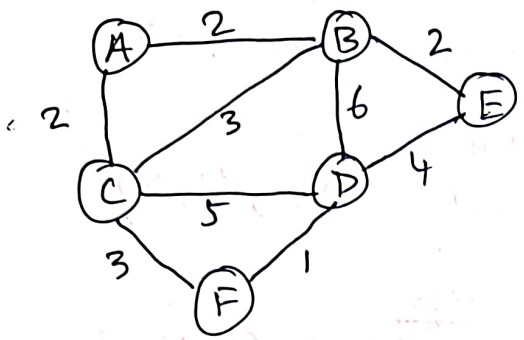
visit	Neighbour	stack
A	✓B, D	A
B	✓E	A, B
E	✓C	A, B, E
C	B, ✓G.	A, B, E, C
G	✓I, H	A, B, E, C, G.
I	B ✗	A, B, E, C
G	I ✗, ✓H	A, B, E, G, G.
H	I ✗, E ✗	A, B, E, C
C	B ✗, G ✗	A, B, E
E	C ✗	A, B.
B	E ✗	A
A	B ✗, ✓D	A
D	C ✗, G ✗	
A	✓	



DFS path
= A, B, E, C, G, I, H, D

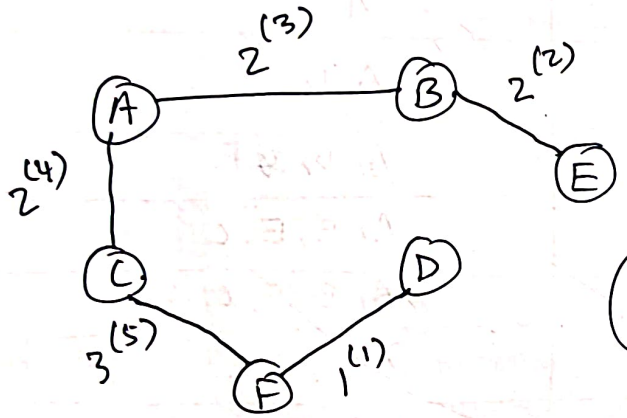
pre-order Traversal path.

(9) Original Graph



Kruskal

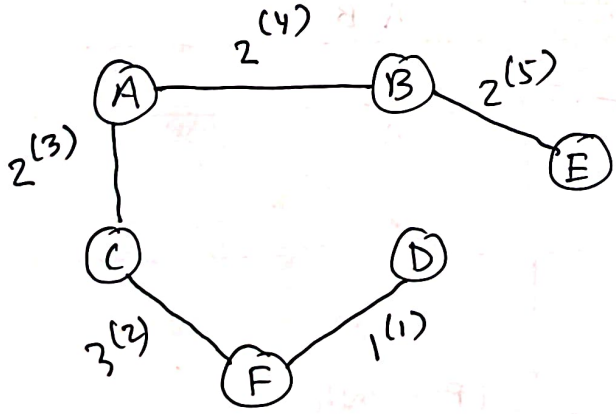
Focus 1 = min cost Focus 2 = should be connected



$\left(\begin{matrix}\text{min}\\ \text{cost}\end{matrix}\right)_{\text{Kruskal}} = 2\times3 + 3 + 1 = 10.$

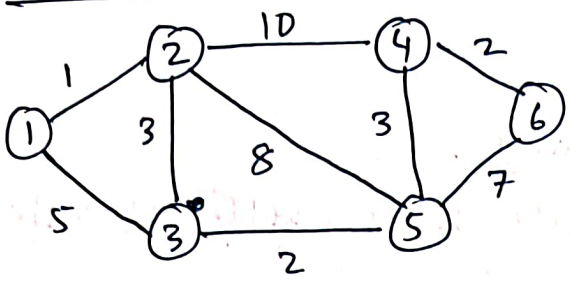
Prim

Focus 1 = Connected Focus 2 = min cost



$\left(\begin{matrix}\text{min}\\ \text{cost}\end{matrix}\right)_{\text{prim}} = 10$

Dijkstra



Problem

From source = 1
Find the min distances to all nodes.

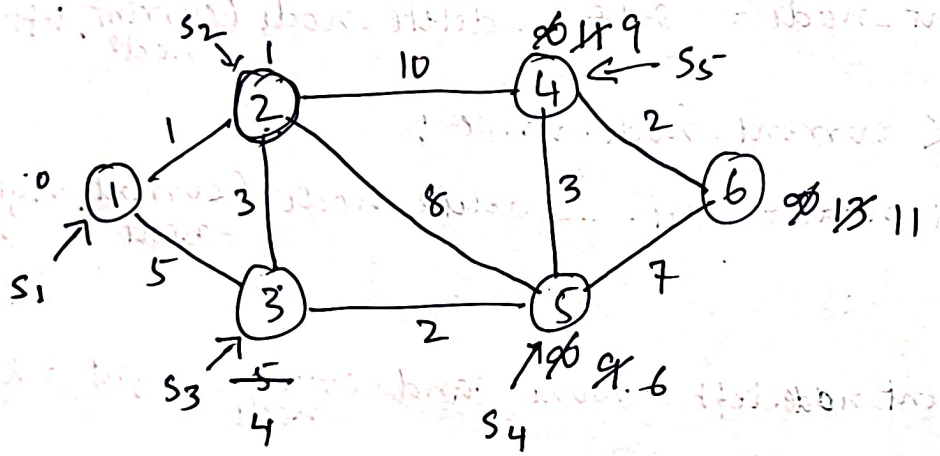
constraint - Path should exist
objective - min distance from source.

Relaxation Logic

$$d[v] + c(u,v) < d[v]$$

$$d[v] = d[u] + c(u,v).$$

* Choose global min always + Not selected node.



1 - 2	=	1
1 - 3	=	4
1 - 4	=	9
1 - 5	=	6
1 - 6	=	11

min distances.

Selected	2	3	4	5	6
			∞	∞	∞
1	(1)	5			
2	(1)		(4)	11	9
3	(1)		(4)	11	(6)
5	(1)		(4)	(9)	(6)
4	(1)		(4)	(9)	(6)

					13
					(11)

min distances.

11. Delete node in B.S.T

Idea

```
def delete_node(self, value):
    return self.__delete_node(self.root, value)

def __delete_node(self, current_node, value)
    if current_node is None:
        return None
```

Search

```
    elif value > current_node.value:
        current_node = self.__delete_node(current_node.left, value)

    elif value < current_node.value:
        current_node = self.__delete_node(current_node.right, value)
```

Value found →

```
    else:
        if current_node.left == None and curr_node.right == None
            return None
        elif curr_node.left == None and curr_node.right != None
            curr_node = curr_node.right
        elif curr_node.left != None and curr_node.right == None
            curr_node = curr_node.left
```

Both left + right child exist →

```
    else:
        sub_tree_min = self.min_value(curr_node.right)
        curr_node.value = sub_tree_min
        curr_node.right = self.__delete_node
            (curr_node.right, sub_tree_min)
```

```
def min_value(self, curr_node):
    while curr_node.left is not None:
        curr_node = curr_node.left
    return curr_node.value
```