Date:

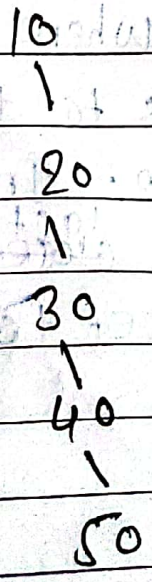
Page:

2079.

1) Why do we need to balance the binary search tree? Justify with example. Create an AVL tree From the data 24, 12, 8, 15, 35, 30, 57, 40, 45, 78.

→ Balancing a binary search tree (BST) is important to ensure efficient search, insertion, and deletion operations. An unbalanced BST can lead to skewed structures where the tree degenerates into a linked list, significantly degrading the performance of these operations from their optimal logarithmic time complexity. Balancing the tree helps maintain a more even distribution of nodes, resulting in better overall performance.

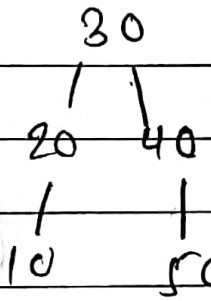
Let's illustrate this with an example. Consider an unbalanced binary search tree.



Date:
Page:

In this unbalanced tree, each new node has been inserted as the right child of its parent, leading to a linear structure, searching for a value in this tree would take O(n) time in the worst case. Which defeats the purpose of using a binary search tree.

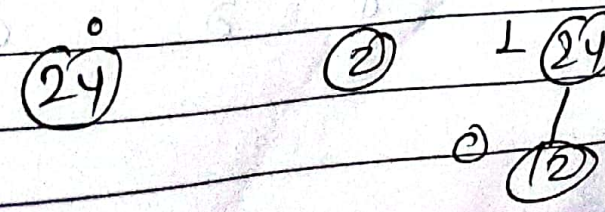
Now, let's compare it with a balanced binary tree:



In this balanced tree; the distribution of nodes on both side of each subtree is more even, creating a more efficient structure. Searching for a value of this tree would take O(log n) time in the worst case, which is the ideal behavior of a binary search tree.

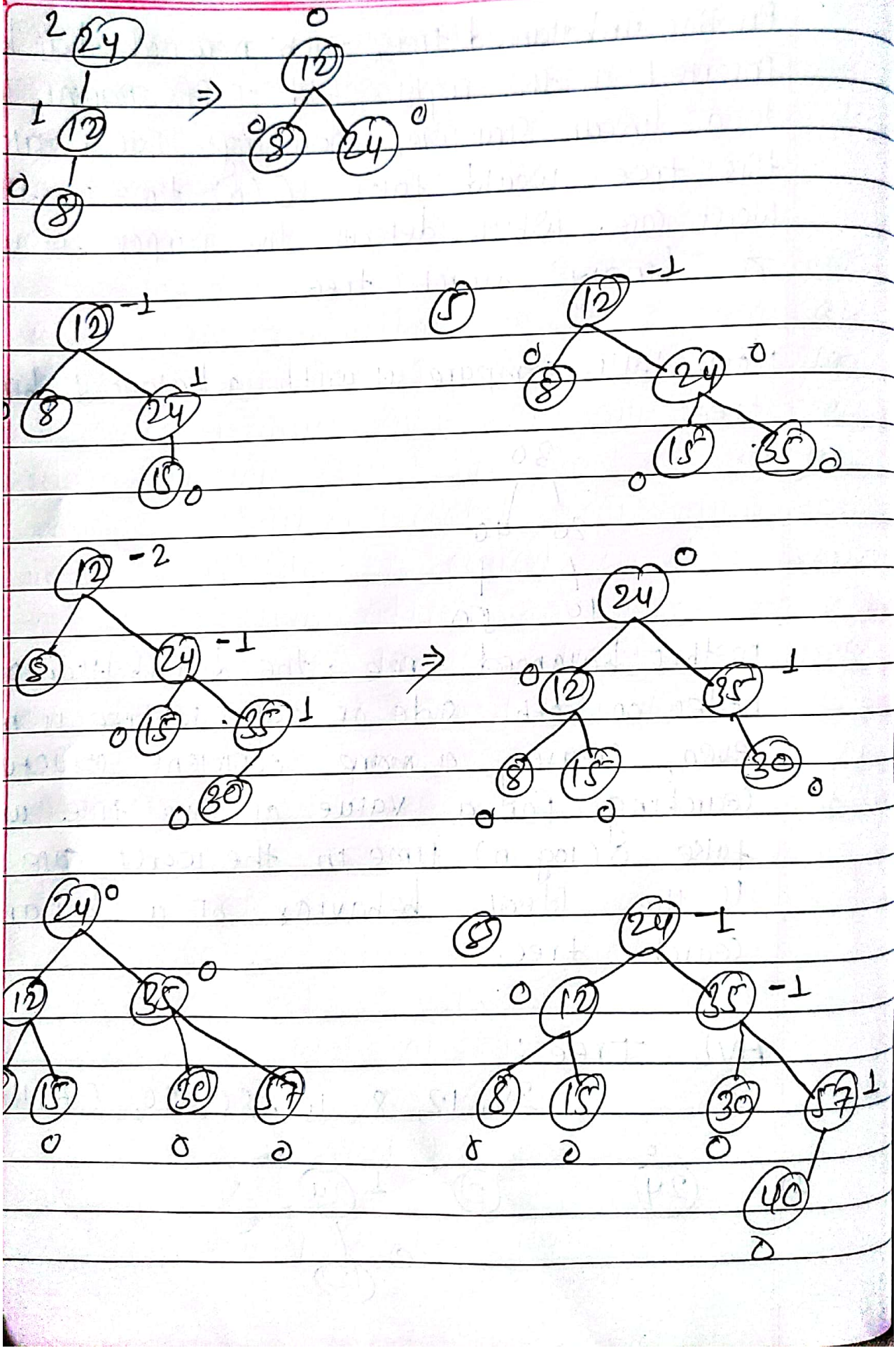
AVL Tree:

24, 12, 8, 15, 35, 30, 57, 40, 45, 78

1) 

Date: _____

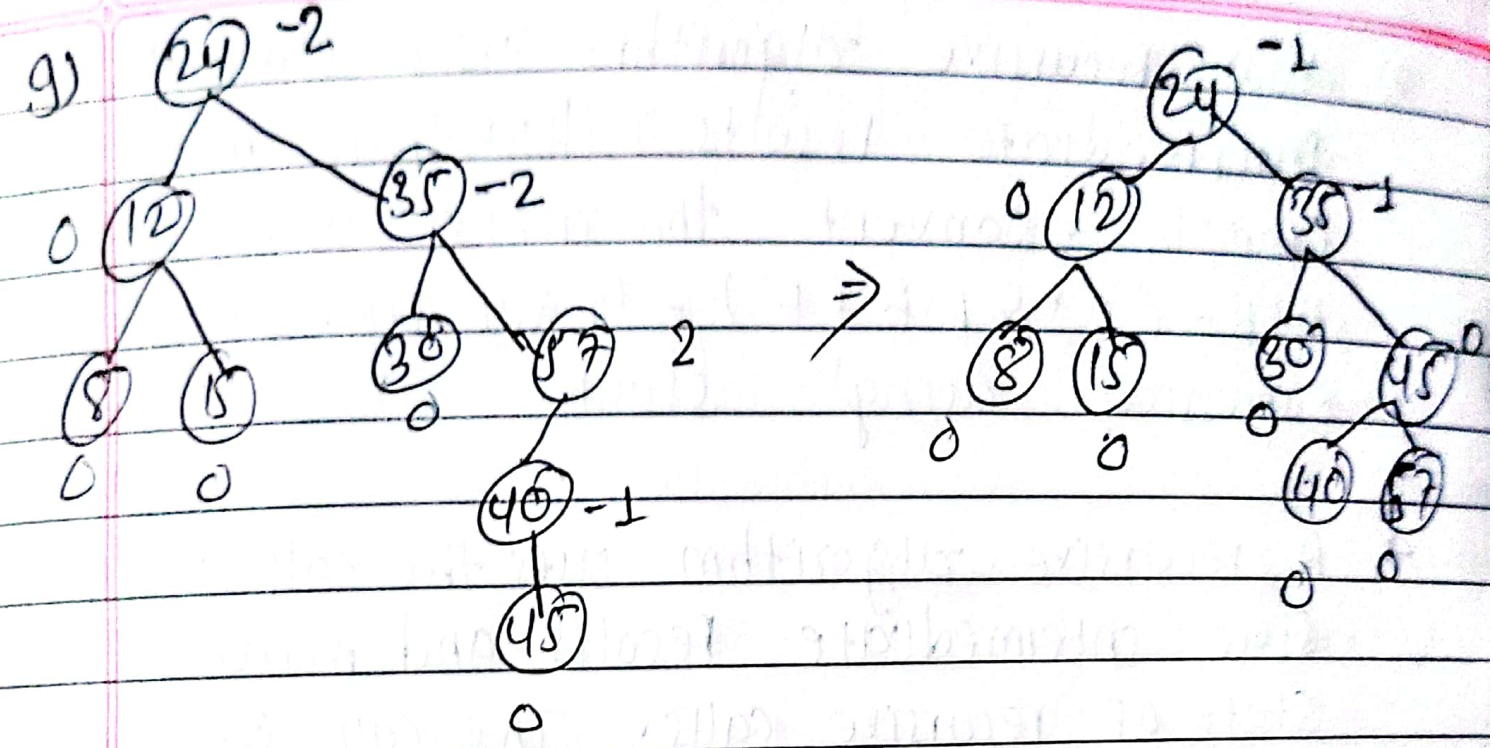
Page: _____


24
12
8
⇒
12
8
24
12
8
24
15
5
12
8
24
15
35
12
8
24
15
35
30
⇒
24
12
35
8
15
30
24
12
35
15
30
57
5
24
12
35
8
15
30
57
40

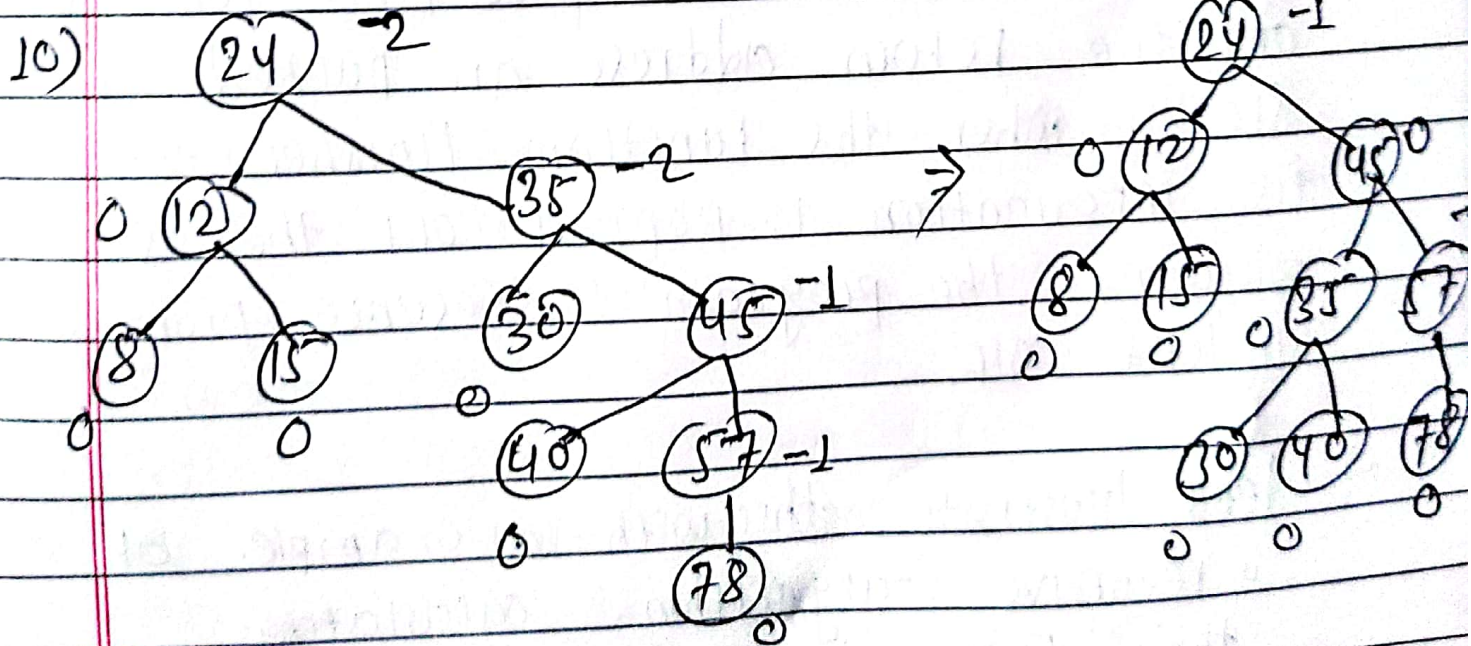
Date:

Page:

9)



10)



Date:

Page:

2) How recursive algorithm uses stack to store intermediate results? Illustrate with an example. convert the infix expression A+B-(C*D/E+F)-G*H into postfix expression using stack.

→ A recursive algorithm uses the call stack to store intermediate results and manage the state of recursive calls. The call stack is a data structure that keeps track of function calls and their associated local variables. When a function is called, its local variables and the return address are pushed onto the stack. When the function finishes executing, its information is popped off the stack, allowing the program to resume from where it left off.

→ Let's illustrate this with a example of a recursive algorithm. calculating the Factorial of a number

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Date:
Page:

result = Factorial (5)
print (result).

In this example the Factorial function is defined to calculate the Factorial of a given positive integer n. The base case is when n equals 0, in which case the function returns 1. Otherwise, the function returns n multiplied by the Factorial of n-1.

→ Now, show how the recursive calls and the call stack work when calculating Factorial (5):

1) Factorial (5) is called. The call stack store the function call Factorial (5).
2) Inside the function: n is not 0, so the function enters the else block. The function calls Factorial (4).
3) Factorial (4) is called. The call stack stores both Factorial (5) and Factorial (4).
4) Inside the function. n is not 0. so the function enters the else block. The function calls Factorial (3).
5) The pattern continues until Factorial (0) is called. Factorial (0) returns 1 (base case) The call stack now contains: Factorial. (5) Factorial. (4), Factorial (3) Factorial. (2), Factorial (1) Factorial (0).

Date:
Page:

6) Now, the result are propagated back: Factorial (1) returns 1*1 = 1. Factorial (2) returns 2*1 =2 Factorial (3) returns 3*2=6. Factorial (4) returns 4*6 = 24, Factorial (5) returns 5*24 =120.

7) The call stack is gradually unwound as the results are calculated. Factorial (0) is popped off the stack Factorial (1) is popped off the stack... & so on. until only Factorial (5) remains on the stack.

8) Finally, the call to Factorial (5) completes and the result 120, is returned.

→ This example demonstrate how the call stack is used to store intermediate results and manage the flow of recursive function calls

Infix to postfix solution:

A+B - (C*D / E+F) - G*H

Scanned symbol	operator stack	o/p postfix string
A		A
+	+	
B	+	AB

Date: _____

Page: _____

-	-	AB+
C	- C.	AB+
C	- C	AB+C
*	-(+*	AB+C
D	-C-*	AB+CD.
/	-C/	AB+CD*
E	-C/	AB+CD*E
+	-C+	AB+CD*E/
F	-C+	AB+CD*E/F
)	-	AB+CD*E/F+
-	-	AB+CD*E/F+-
G	-	AB+CD*E/F+-G
*	-*	AB+CD*E/F+-G
H	-*	AB+CD*E/F+-GH
		AB+CD*E/F+-GH*

Date:

Page:

Section B

4) Sort the number 82, 73, 12, 39, 26, 88, 2, 9, 60, 41 using shell sort.

-) Given: n=1

Iteration 1: n/2 = 5

82, 73 12 39 26 88 2 9 60 41

82 2 12 39 26 88 73 9 60 41

82 2 9 39 26 88 73 12 60 41

82 2 9 39 26 88 73 12 60 41

82 2 9 39 26 88 73 12 60 41

Iteration 2: (n/2)/2 = 2

9 2 82 39 26 88 73 12 60 41

9 2 82 39 26 88 73 12 60 41

9 2 26 39 82 88 73 12 60 41

9 2 26 39 82 88 73 12 60 41

9 2 26 39 73 88 82 12 60 41

9 2 26 12 73 39 82 88 60 41

9 2 26 12 60 39 73 88 82 41

9 2 26 12 60 39 73 41 82 88

Date:
Page:

Iteration 3 : ((n/2)/2)/2 = 1

2 9 26 12 60 39 73 41 82 88
2 9 26 12 60 39 73 41 82 88
2 9 12 26 60 39 73 41 82 88
2 9 12 26 60 39 73 41 82 88
2 9 12 26 39 60 73 41 82 88
2 9 12 26 39 60 73 41 82 88
2 9 12 26 39 41 60 73 82 88
2 9 12 26 39 41 60 73 82 88
2 9 12 26 39 41 60 73 82 88

5) Why do we need asymptotic notation? Describe about Big Oh notation with its curve.

→ Asymptotic notation is mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

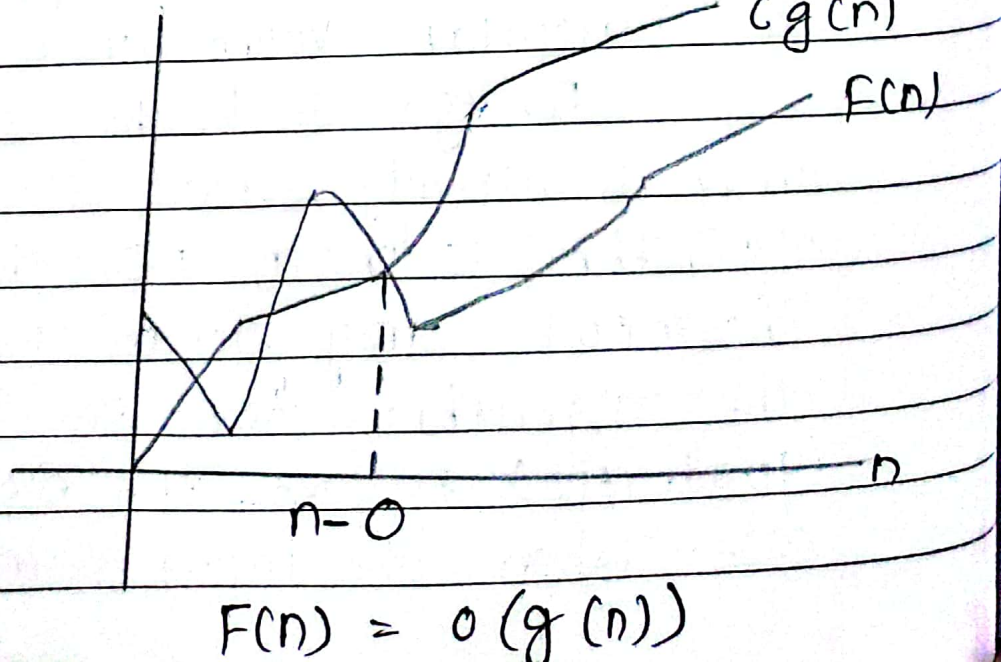
For eg: In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear, i.e the best case, but when the input array is in reverse condition. the algorithm takes the maximum ~~the~~ time (quadratic) to sort the elements i.e the worst case. When the input array is neither sorted

Date:
Page:

nor in reverse order, then it take average time. These durations are denoted using asymptotic notations

Big-O notation:

It represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm. It is the most widely used notation for Asymptotic analysis. It specifies the upper bound of a function. It determines the maximum time required by an algorithm or the worst case time complexity. It returns the highest possible output value (big-O) For a given input. It is defined as the condition that allows an algorithm to complete statement execution in the longest amount of time possible



$F(n) = O(g(n))$

Date:
Page:

IF F(n) describes the running time of an algorithm, F(n) is O(g(n)) if there exist a positive constant C and n0 such that, $0 \leq F(n) \leq cg(n)$ for all $n \geq n0$.

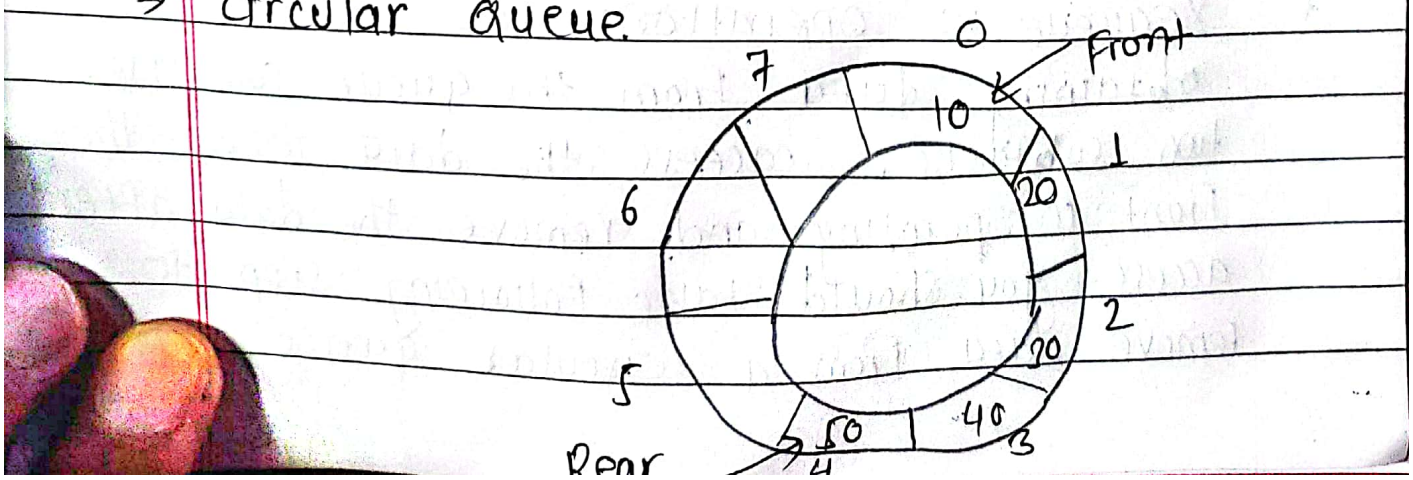
It returns the highest possible output value (big-O) for a given input.

The execution time serves as an upper bound on the algorithm's time complexity.

6) Define Queue. Explain about enqueue and dequeue operation in circular queue.

→ A Queue is define as a linear data structure that is open at both ends and the operations are performed in first in first out (FIFO) order.

→ Circular queue.



Date:
Page:

Enqueue Operation:

→ we should follow the following steps to insert (enqueue) a data element int a circular queue.

Step 1: Check if the queue is full (Rear + 1 % MaxSize = Front)

Step 2: if the queue is full, there will be an overflow error.

Step 3: check if the queue is empty, and set both Front and Rear to 0

Step 4: if Rear = MaxSize - 1 & Front ≠ 0 (rear pointer is at the end of the queue and front is not at 0th index), then set Rear = 0.

Step 5: Otherwise, set Rear = (Rear + 1) % maxsize

Step 6: insert the element into the queue (Queue [Rear] = x)

Step 7: Exit.

* Dequeue () operation:

obtaining data from the queue comprises two subtasks: access the data where the Front is pointing and remove the data after access. You should take following step to remove data from a circular queue.

Date:
Page:

Step1: check if the queue is empty (Front = -1 & Rear = -1)
Step2: If the queue is empty, underflow error.
Step3: set element = queue [Front].
Step4: If there is only one element in a queue. set both front and Rear to -1 (if Front = Rear, set Front = Rear = -1)
Step5: And if Front = MaxSize - 1 set Front = 0
Step6: otherwise, set Front = Front + 1
Step7: Exit.

7) Write a program to implement binary search.

→
```
#include <stdio.h>
Int main ()
  {
Int c, first, last, middle, n, search, array[100]
printf ("Enter number of elements \n");
    scanf ("%d", &n);
printf ("Enter %d integers \n", n);
  For (c= 0; c<n; c++)
    scanf ("%d", &array[c]);
printf ("enter value to find \n");
scanf ("%d", &search);
```

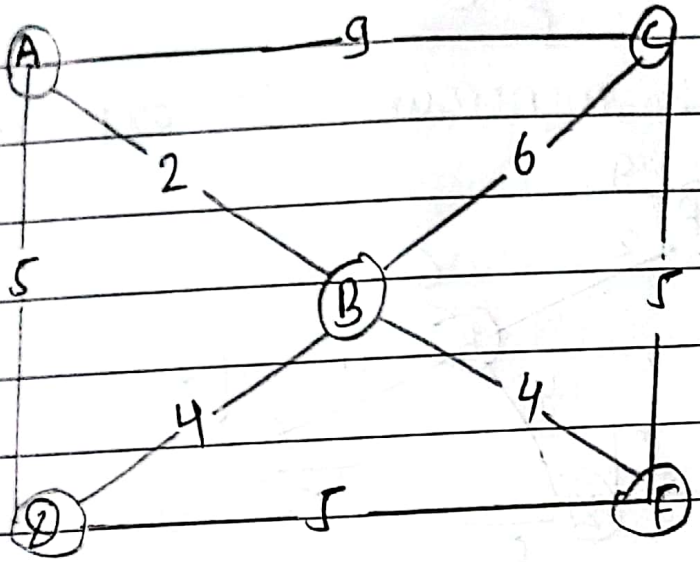
Date: _____

Page: _____

```
first = 0;
last = n-1;
middle = (first + last) /2;
while (first <= last)
        {
if (array[middle] < search)
  first = middle + 1;
else if (array[middle] == search)
    {
printf ("%d found at location %d \n", search,
                                middle +1);
  break;
    }
    else {
  last = middle -1;
middle = (first + last) /2;
  }
if (first > last)
printf ("Not found ! %d isn't present
in the list .\n", search);
        return 0;
        }
```

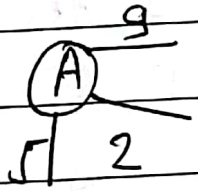
Date:
Page:

8) Find the MST OF Following graph using prim's algorithm.

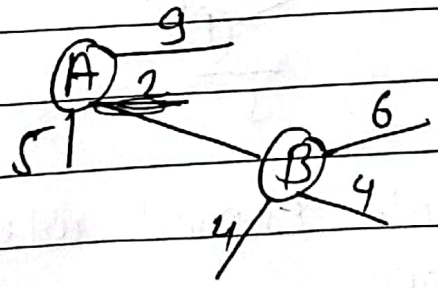


Soln Slecting A as initial vertex.

Step1:



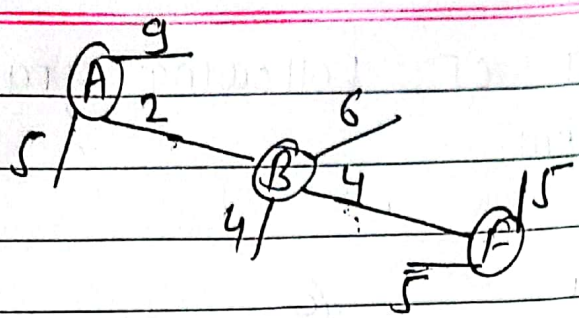
Step 2: 2 is minimum, so, expand 2



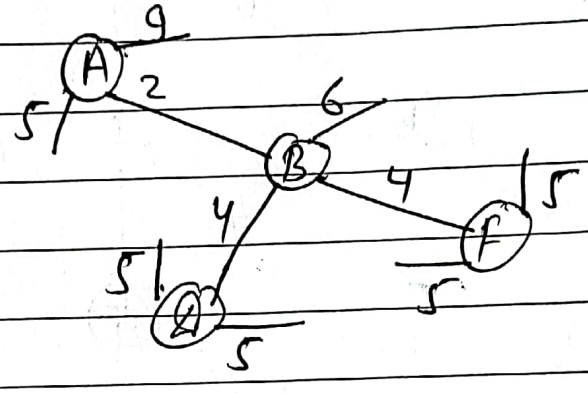
Step 3: 4 is minimum, so, expand either of. then

Date: _____

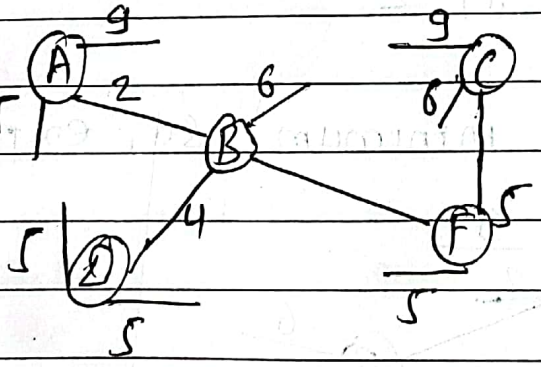
Page: _____



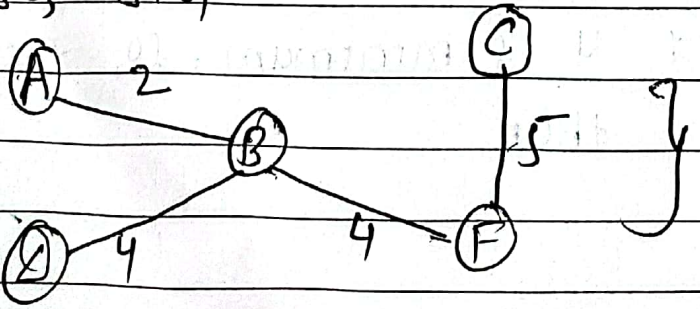
Step 4: U is minimum so, expand



Step 5: 5 is minimum among many 5's the only one that doesn't from loop is F to C. so expand.



Now, we can't expand without creating loop so, stop.



Cost = 2+4+4+5 = 15

Date
Page

9) Assume you have to store the data {0, 1, 2, 4, 5, 7} into a hash table of size 5, with hash function, $h(x) = x \% 5$, Apply linear probing and double hashing as collision resolution techniques.

→ Here, Given data : {0, 1, 2, 4, 5, 7}
hash function $H(x) = x \% 5$
performing hashing as
For 0, $H(0) = 0 \% 5 = 0$ i.e key of 0 = 0
for 1, $H(1) = 1 \% 5 = 1$ i.e key of 1 = 1
for 2, $H(2) = 2 \% 5 = 2$ i.e key of 2 = 2
for 4, $H(4) = 4 \% 5 = 4$ i.e key of 4 = 4
For 5, $H(5) = 5 \% 5 = 0$ i.e collision.

collision occurs, so using linear probing to solve it.
Acc. to linear probing.
$H'(x) = (H(x) + i) \% 5$; i → probe no.
For $i = 1$,
$H'(5) = (H(5) + 1) \% 5$
$= (0 + 1) \% 5$
$= 1$; again collision occurs.

For $i = 2$
$H'(5) = (H(5) + 2) \% 5$

Date: .....

Page: .....

= 2 ; again collision occurs.

For i = 3

$H'(5) = (H(5) + 3) \% 5$

$= 3$

i.e key of 5 = 3

For 7, $H(7) = 7 \% 5 = 2$

Collision occurs so,

$H'(7) = (H(7) + 1) \% 5$

$= 3$

Collision so,

$H'(7) = (H(7) + 2) \% 5$

$= 4$

Collision occurs on 2 on repeatedly

∴ No solution is found for 7

i.e no key for 7

value	key
0	0
1	1
2	2
5	3
4	4

Let us try solving collision using double hashing we define hash function as:

$H_1(x) = x \% 5$

$H_2(x) = 3 - (x \% 3)$

H(21) = (H1(21) + 1' H2(21)) %.5

similarly as above, keys for 0, 1, 2, 4 are 0, 1, 2, 4 simultaneously. However collision occurs for 5 & 7.

for 5,

H1(5) = 5 %.5 = 0

H2(5) = 3 - (5 %.3) = 1

H(5) = (H1(5) + 1 + H2(5)) %.5

= 1

Again,

H(5) = (H1(5) + 2 x H2(5)) %.5

= 2

Again H(5) = (H1(5) + 3 x H2(5)) %.5

= 3

∴ key for 5 = 3

For 7

H1(7) = 7 %.5 = 2

H2(7) = 3 - (7 %.5) = 2

H(7) = (H1(7) + 1 x H2(7) %.5

= 4

Again, H(7) = (H1(7) + 2 x H2(7)) %.5

= 1

Again H(7) = (H1(7) + 3 x H2(7)) %.5

= 3 & soon

∴ key for 7 is not found

(same Fig.)

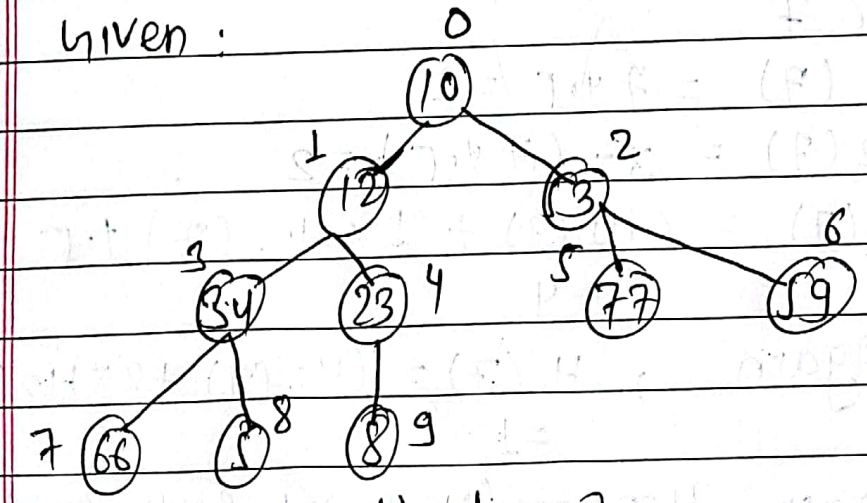
Date:

Page:

10) In which case the position of pivot element in quick sort always either in the last or the first position? create a max heap from the numbers {10, 12, 53, 34, 23, 77, 59, 66, 51, 8}

→ In quick sort, worst case occurs when the pivot element is either greatest or smallest element suppose, if the pivot element is always the last element of the array, the worst case would occore when the given array is sorted already in ascending or descending order. The worst case time complexity of quicksort is $O(n)^2$

Given:



We know

$i = n/2 - 1$ to 0

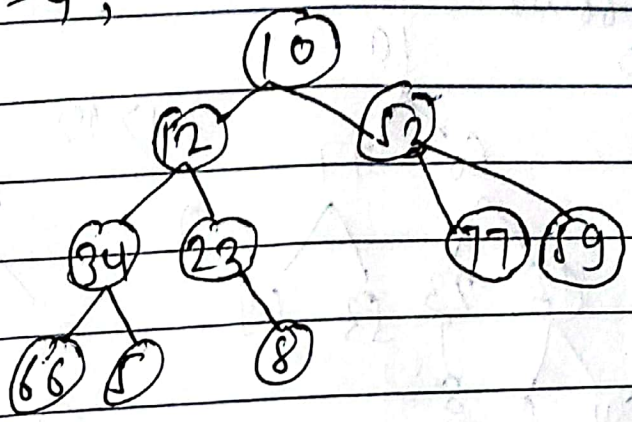
$= 10/2 - 1$ to 0

$= 4$ to 0

Date
Page

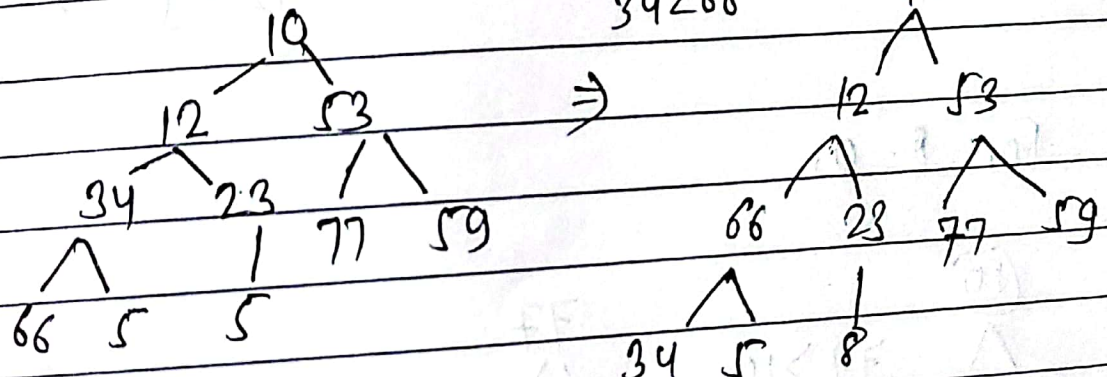
4, 3, 2, 1, 0.

For i = 4,



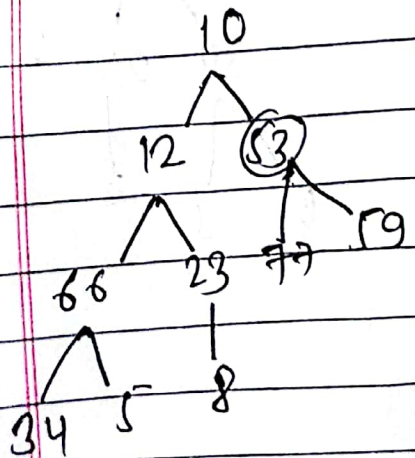
23 > 8 so no need to swap

For i = 3

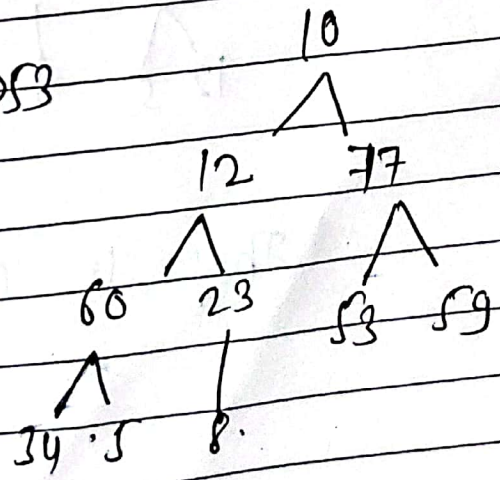


34 < 66

For i = 2,



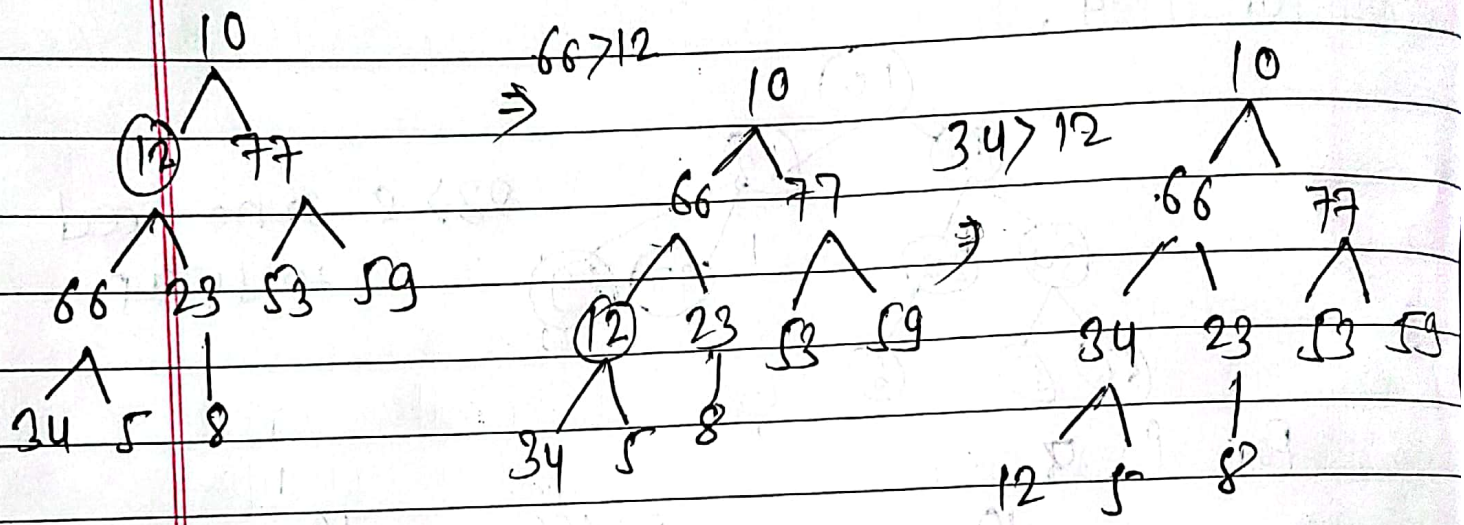
77 > 53



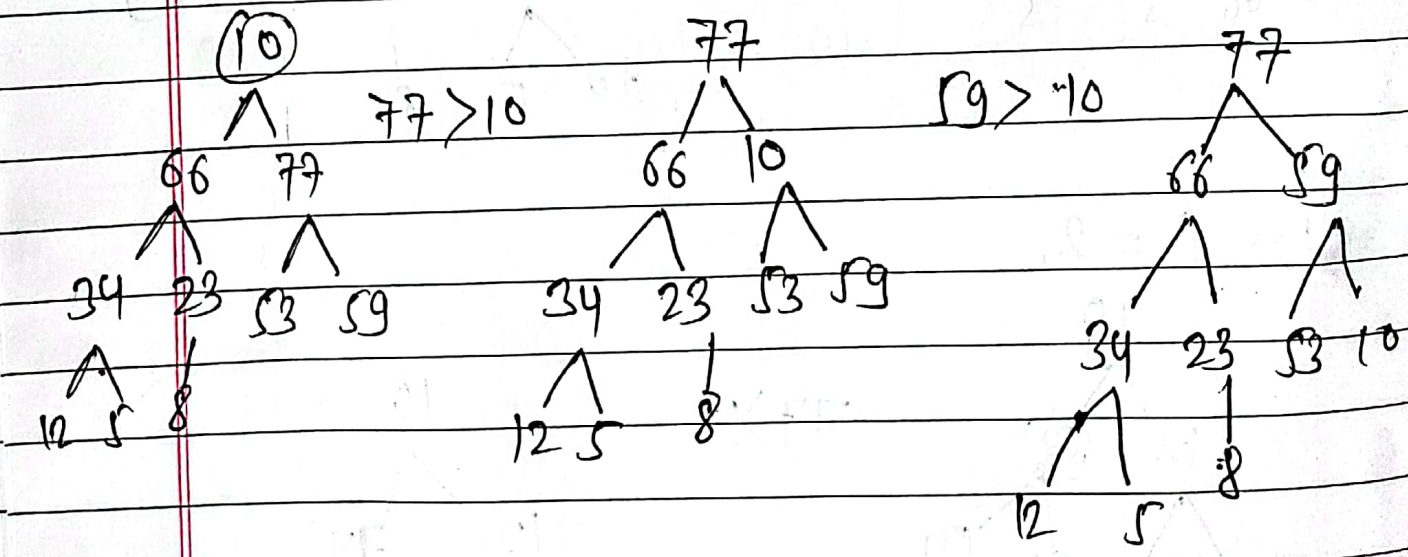
Date: .....

Page: .....

For i = 1



For i = 0,



This is max heap.

Date:
Page:

11) Evaluate the postfix expression 574-*8/4+ using stack.

Given:

→ 574-*8/4+

character scanned	A	B	value	operand stack
5		1		5
7				5, 7
4				5, 7, 4
-	7	4	3	5, 3
*	5	3	15	15
8				15, 8
/	15	8	1.875	1.875
4				1.875, 4
+	1.875	4	5.875	5.875

Date:

Page:

12) Write short note on:

a) Priority queue.

b) Breadth first traversal of a graph.

i)-> Priority Queue.

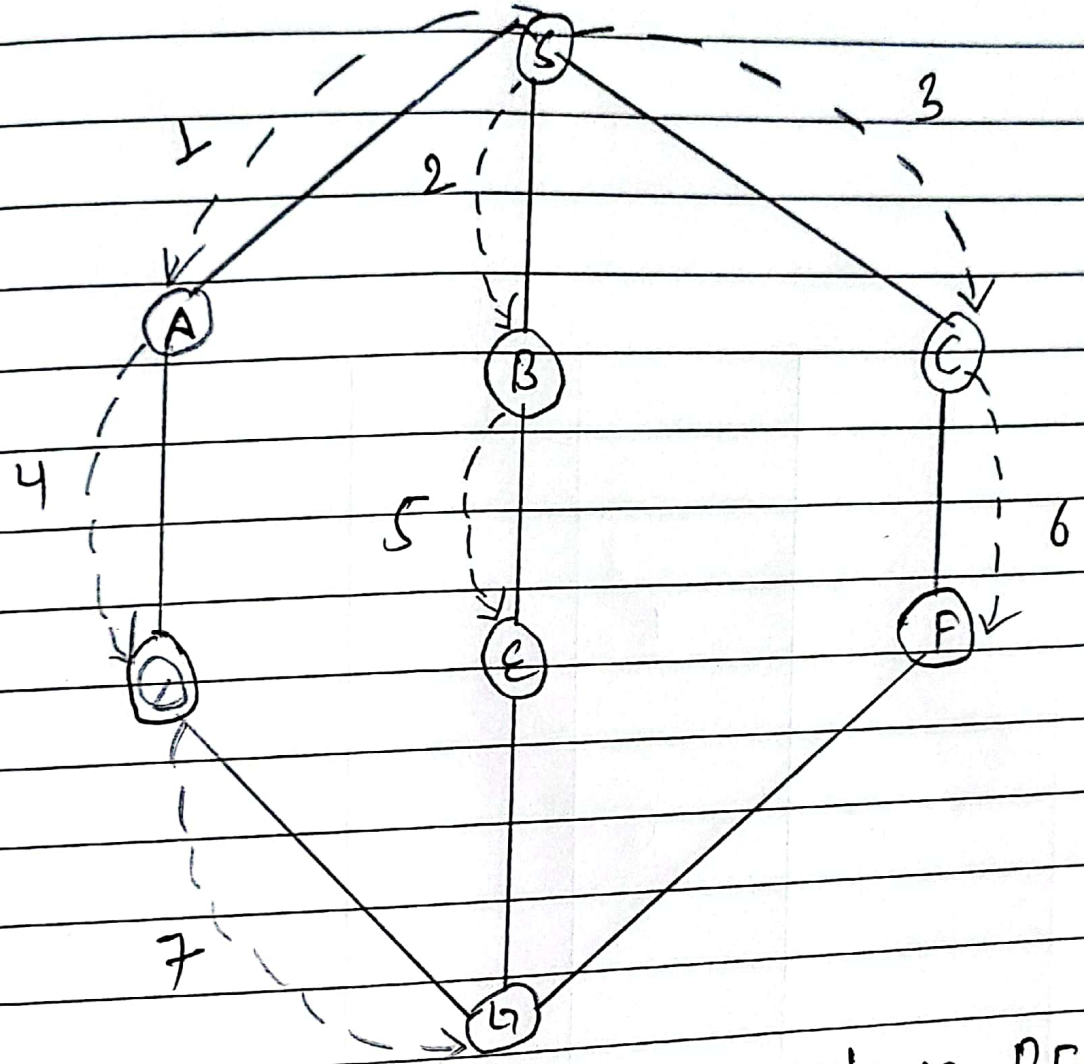
priority queue is an abstract data type that performs operations on data elements per their priority. The hospital emergency queue is an ideal real-life example of a priority queue.

ii) Breadth First Traversal of a graph.

-> Breadth First search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start to search, when a dead end occurs in any iteration.

Date: _____

Page: _____



As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs