Answer Script

Question No. 01

Write the time complexity of the following code segments with proper explanation.

```
void fun(int l,int r)
{
        int mid = (l+r)/2;
        for(int i = l ; i <= r ; i++)
        {
                cout<<i<<endl;
        }
        if(l<r){
                fun(l,mid);
                fun(mid+1,r);
        }
}

int main()
{
        int n;
        cin>>n;
        fun(0,n-1);
}
```

```
for(int i = 1 ; i <= n/2 ; i++)
{
        for(int j = 1 ; j <= n ; j = j + i)
        {
                cout<<i<<" "<<j<<endl;
        }
}
```

Answer No. 01

First portion is a recursive function,

Here first loop runs n=r-l+1;
As it is a recursive call, we can write a general formula, T(n)=aT(n/b)+ +f(n), which is derived from the master theorem.
Where a is the number of recursive sub problems, each of size is n/b. And f(n) is the complexity of outside of recursive call.

From the fun() we get that a=2 and b=2.
So, T(n)=2T(n/2)+O(n).
We know that a<b^c for T(n)=O(n^c), where c is the growth rate of the non recursive part which is 1.
a=b^c for T(n)=O(n^c*logn)
a>b^c for T(n)=O(n^logn)
As a==b log 2_(2)=1=c.
So the time complexity of the fun() is O(n^1*logn)->O(nlogn).
In the main function the time complexity is O(1).

The outer loop runs n/2 times
As i=1,2,3,.......,n/2
Inner loop runs n/i times
Now,
n/1+n/2+n/3+............+n/(n/2)
=n(1+½+⅓+.........+n/2)
The second part is a harmonic series so the summation of it logn.
=nlogn
So, the time complexity will be O(nlogn).

Question No. 02

Suppose you are implementing a linked-list where you want to maintain a floating point number and a character in each node. Each node will contain a next pointer and also a next_to_next pointer that will keep track of the node that is next to the next node. What will the node class look ?

```
class Node{
        // write your variables
};
```

Answer No. 02

```
class Node{
public:
float point;
char ch;
Node *nxt;
Node *nxt_to_nxt;

Node(float point, char ch)
{
```

```
this->point=point;
this->ch=ch;
this->nxt=NULL;
this->nxt_to_nxt=NULL;
}
};
```

Question No. 03

Write the main difference between linear and non-linear data structures. Compare between Stack, Queue and Deque. Are stack, queue, deque linear or non-linear data structure? What about a tree?

Answer No. 03

Linear data structure	Non-linear data structure
1. It looks like a straight line	1. It looks like a reverse of original tree
2. There are 2 sequences ,previous and next.	2. There are 2 or more sequences like parent, left, right etc.
3. Examples- stack, queue, deque, singly and doubly linked list	3. Examples– binary tree, binary search tree.

Stack	Queue	Deque
1. Works in LIFO method	1. Works in FIFO method	1. Works in both LIFO and FIFO method
2. Binary representation cannot implement	2. Binary representation can implement	2. Binary representation can implement
3. Postfix expression can implement	3. Postfix cannot implement.	3. Postfix expression can implement.
4. Last value is possible to print	4. First value possible to print.	4. All operation can do in it.

Stack, queue, deque are linear data structures as they store value in one straight line sequence.

Tree is a non-linear data structure as it has many branches and one branch has another 2 or more branches.

Question No. 04

Between singly linked list and doubly linked list which is better for implementing Stack and Queue? What about Deque?

Answer No. 04

Between 2 linked lists, a singly linked list is better for implementing stack.
In stack data structure, it works in the LIFO method, it means last in first out. If we add a value in the stack it will store in the last position and when we want any pop operation it deletes the last value of it. In a singly list there is a pointer called head which is pointed only at the last of the linked list. So it is easier to implement stack in a singly linked list and it takes less memory to build up. Again a singly linked list is preferable for implementing Stack.

Now for the queue, a doubly linked list is a better choice. In queue data structure, it works in FIFO method,it means first in first out. If we add a value in the queue it will store in the last position and when we want to print or pop operation ,it returns only the first value of it. It assures a more efficient implementation when removing values from the front of the queue. In a doubly linked list , each node maintains a pointer to both next and previous node, which is very helpful to delete the elements from the front in a small time. So, a doubly linked list is preferable for implementing Queue.

For deque ,it is a more powerful data structure as it can do all operations like stack and queue. We need to maintain the head and the tail part to implement this. For that we have to use a doubly linked list to implement it. Deque can delete both the front and last value in it. Doubly linked list allows efficient insertion and deletion at both ends of the list in a small time and makes it perfect. So, a doubly linked list is preferable for implementing Deque(double ended queue).

Question No. 05

Convert the infix expression to postfix expression using a stack. You need to show all the steps.

a*b+c*d+e

Answer No. 05

In expression, operands are written before the operators to execute the operation more perfectly for a computer is called postfix expression.
Because the postfix expression works faster than the infix expression. There is no concern which operator's precedence is first . Besides, in postfix expressions we do not need to use parenthesis. For doing the operation properly on the computer we need it.

The postfix expression of **a*b+c*d+e** is **ab*cd*+e+**. I will take the infix expression as input.
Procedure:

1. Declaration of string and input the infix expression as string .
   string s;cin>>s;
2. Create a stack of characters.
   stack<char>S;
3. Declare a blank string.
   string r="";
4. Then I start a loop from 0 to before the string size of s and make a condition if the character is a to z. If it is true then input it in the S blank string.
   for(int i=0;i<s.size();i++)
   {if(s[i]>= 'a' && s[i]<= 'z')
   {r+=s[i];}
5. Then I create a function name precedence which will return 0 if the character is + or -. Other it will return 1; int precedence(char r){if(r== '+' || r== '-'){return 0;}return 1;}
6. If it is not then I have to check if it is '+' or '-' sign. If this comes then I can push the sign in the stack.
   else{while(S.size() && precedence(S.top())>=precedence(s[i]))
   r+=S.top();
   S.pop();}
   S.push(s[i]);)}}
   Again if '*' or '/' sign come then first of all I push it in the stack, later if '+' or '-' sign come then I add the *,/ characters in r string and delete the top character of the stack and also add the +,- sign in the string until the stack size becomes 0.
7. After that if there is any value in the stack then lastly I add it to the string and pop the stack.
8. At last I print the r string.

Implement:

1. s= "a*b+c*d+e", r= "",S={}.
2. s= "a*b+c*d+e",r= "a",S={}.
3. s= "a*b+c*d+e",r= "a", S={'*'}.[as s[1]= '*']
4. s= "a*b+c*d+e",r= "ab",S={'*'}.
5. s= "a*b+c*d+e",r= "ab*", S={'+'}. [as s[3]= '+'][s[3]= '+' & the precedence of + & * is *]
6. s= "a*b+c*d+e",r= "ab*c",S={'+'}.
7. s= "a*b+c*d+e",r= "ab*cd", S={'+'}

8. s= “a*b+c*d+e”,r= “ab*cd”,S={‘+’, ‘*’}
9. s= “a*b+c*d+e”,r= “ab*cd*”, S={‘+’}
10. s= “a*b+c*d+e”, r= “ab*cd*+”, S={}
11. s= “a*b+c*d+e”, r= “ab*cd*+”, S={‘+’}
12. s= “a*b+c*d+e”, r= “ab*cd*+e”, S={‘+’}

As the loop is over then we have to check if there is any value in the stack.

1. r= “ab*cd*+e”, S={‘+’}
2. r= “ab*cd*+e+” ,S={}

Then if it prints, it will show in console—> ab*cd*+e+.

Question No. 06

Compare the memory usage of Array, Singly Linked-list and Doubly Linked-list with necessary explanation.

Answer No. 06

Array ,singly and doubly linked are linear data structures but the memory usage varies them in different sections.

Array	**Singly linked list**	**Doubly linked list**
1. It requires only one field.	1. It requires 2 fields to store the value	1. It requires 3 fields to store the value.
2. It only takes the value.	2. It takes the value but it stores it in a node which points to the next node.	2. It takes the value but it stores it in a node which points to the previous and the next node.
3. It takes memory from ram serially.	3. There is no fixed place of storing the value in ram, it can store either serially or far away.	3. It is also the same as a singly linked list.
4. It takes 4 bytes for storing integer value	4. It takes 4*2=8 bytes for integer value.	4. It takes 4*3=12 bytes for integer value.
5. It can go to the previous and next values by changing the index.	5. It can travel to the next node only.	5. It can travel to both the previous and next node.

At last it can say that an array has a fixed size but a linked list can allocate size dynamically and a singly linked list takes less memory than doubly linked list.

Question No. 07

Suppose you are implementing a stack in a scenario where numbers are added in sorted order so that the stack is always sorted. Sometimes you need to quickly search if a value exists in the stack or not. Array or Linked-list which implementation for stack will you prefer in this scenario? Give necessary explanations.

Answer No. 07

In a scenario where numbers are added in sorted order, an array will be more efficient to implement a stack. Here the reasons of it:

1. **Random Access:** array is helpful for random access, it means that it can traverse all the elements at a time . but in the linked list, it is difficult . In array searching is faster than searching in a linked list.
2. **Sorted order:** As the numbers are sorted so it is easier to find any value by doing binary search. It can be implemented in an array easily. Binary search takes logn time to do this work. But in the linked list it will take O(n) for searching. So. array is better for searching.
3. **Memory allocate:** in array ,it does not need to store any kind of pointer . On the other hand, a linked list has to maintain a pointer to build up the list. So it takes more memory than an array.

When searching for a middle value, it is more easy in arrays because of binary search. In the linked list we can maintain the middle point by some variables but it will take more memory and time will consume more.

So, array implementation is more preferable for this scenario because of some important features which makes searching faster, memory efficiency and it is better than linked list.

Question No. 08

Suppose you are maintaining a head and tail for a singly linked-list. What will be time complexity of

a. Inserting a value at the beginning
b. Inserting a value at the end
c. Deleting a value at the beginning
d. Deleting a value at the end
e. Inserting a value at the mid point

f. Deleting a value at the mid point

Answer No. 08

a)Ans:O(1). We have to just create a node and make it head and attach to the next node.
b)Ans:O(1). As we are maintaining the tail so create a node ,attach to the tail, make the new node tail.
c)Ans:O(1). Point the head and make the head to the next node and delete the point.
d)Ans:O(n). Though we are maintaining the tail, we can not go to the previous node of the tail. For that we have to traverse n-1 times to find the previous node of the tail.
e)Ans:O(n). We have to find the middle point by running a loop and inserting it.
f)Ans:O(n). We have to find the middle point by running a loop and removing it.

Question No. 09

Consider the following binary tree in **Fig 1** (node 20 is the root) and answer the given questions.

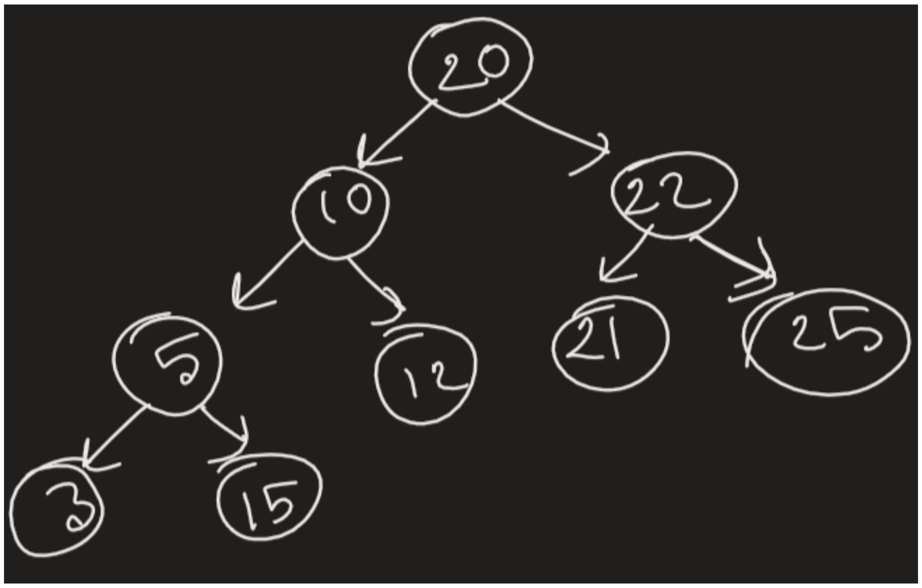
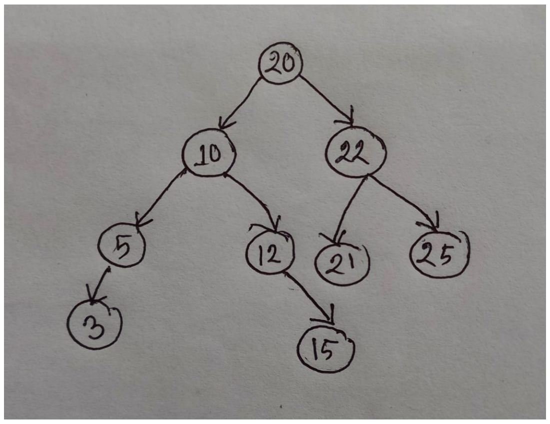


Fig: 1

a. Is the tree a Perfect binary tree? Why or why not?
b. Is the tree a Complete binary tree? Why or why not?
c. Is the tree a Binary search tree? Why or why not?
d. Write down the BFS, inorder, preorder and postorder traversal of the tree.

Answer No. 09

a)Ans:
No, the tree is not a perfect binary tree. Because the tree has 4 levels and 2^4-1=15 nodes should be there. But there are 9 nodes in 4 levels. Again last level is not fully filled, so it is not a binary tree.
b)Ans:
Yes, the tree is a complete binary tree. Because the tree is fully filled except the last level. In a complete binary tree the last level may be filled or may not be filled. So, it is a complete binary tree.
c)Ans:
No, the tree is not a binary search tree. Because 15 no. id is kept in the wrong position. It will be a binary search tree if 15 no. id is kept in the right child of 12 no. id. Again, 10<15, so, 15 has to be kept right of 10 to make it a binary search tree.
The right figure is:



d)Ans:
BFS–20,10,22,5,12,21,25,3,15.
Inorder–3,5,15,10,12,20,21,22,25.
Preorder–20,10,5,3,15,12,22,21,25.
postorder–3,15,5,12,10,21,25,22,20.

Question No. 10

Write the steps to insert **70** in the following binary search tree in **Fig 2** (node 50 is the root).

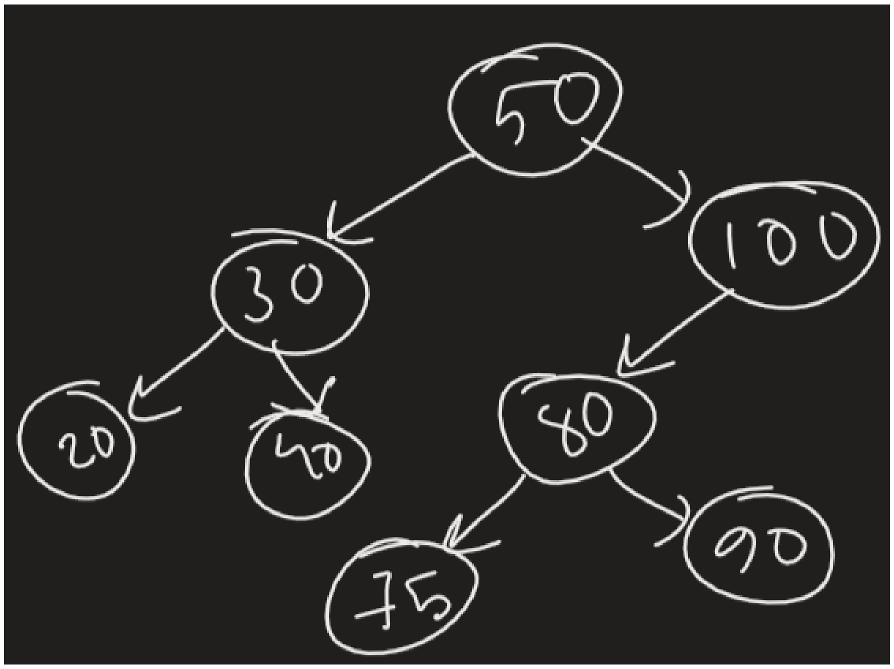


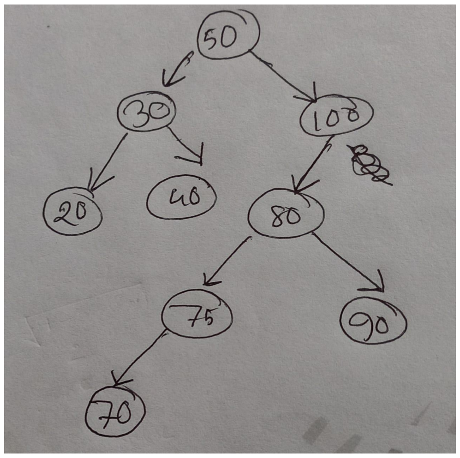
Fig: 2

Answer No. 10

Steps:

1. 70 is greater than root 50. So, 70 will go to the right side of 50.
2. Then 70 is less than 100. So, 70 will go to the left side of 100.
3. After that 70 is less than 80. So, 70 will go to the left side of 80.
4. Now 70 is less than 75. So, 70 will go to the left side of 75.
5. As 75 has no left child, so 70 will sit as a left child of 75.

The figure will be:


50
30
100
20
40
80
75
90
70