SOFTWARE CPC 3

MAX MARKS: 45

Section A
Basics of programming and OOP
(20 marks)

1) Out of the following pairs of traversal details, which are enough to uniquely determine the binary tree? (Choose all that are valid) (2)
    a) Inorder and preorder
    b) Inorder and postorder
    c) Preorder and postorder
    d) Traversals can't be used to determine the tree
2) In a full binary tree of height h, what is the total number of nodes in each level and total number of nodes in the tree?
    (2)
3) Explain the scope resolution operator (::). How is it used to avoid ambiguity while accessing global and local variables with same name? Explain with example. (3)
4) What will the following function return if root of a BST is passed as argument? (2)

```
int f(node * root)
{
        if(root == NULL)
                return -1;
        if(root->left == NULL)
                return root->val;
        return f(root->left);
}
```

    a) Maximum element in the BST and -1 if empty.
    b) Minimum element in the BST and -1 if empty.
    c) Will always return -1.
    d) Will return sum of elements in the left subtree of the original tree.
5) Given a BST, what traversal scheme will you use to get the nodes sorted in descending order?
    (2)
6) Why is the argument to a copy constructor always passed by reference and not by value? What will go wrong if we pass the argument by value? (3)
7) What is the total number of NULL nodes in a full binary tree of height h? (2)
8) How can a heap be used to sort array? Briefly explain with a small sample array. (4)

Section B
Data structures and algorithms
(25 marks)

Note: Marks will be awarded for correct and efficient code. A well written code with a better time and space complexity will fetch you more marks.

Assume the definition of "node" as the following structure for Q1, Q2 and Q3.

```
struct node{int data;node *left;node * right;}
```

1) Write a function that takes as argument two binary trees and checks whether or not they're mirror symmetric. The root of both the trees is provided as arguments. Your function must return 1 if the trees are mirror symmetric and 0 otherwise. Function prototype: (5)

```
int mirror_check(node * root1, node * root2){
        //Definition here
}
```

Example

```
root1        root2
  5          |    5
 / \         |   / \
4   8        |  8   4
 / \         |  / \
13  4        | 4  13
```

Return value: 1 (trees are mirrors of each other)

2) Write a function that takes as argument a sorted array and constructs a height balanced BST from it. A height balanced tree is such that the difference in heights of its two subtrees is <= 1. Your function must return the root of the tree constructed. Function prototype is as follows:: (7)

```
node * create_BST(int arr[], int size){
        //Definition here
}
```

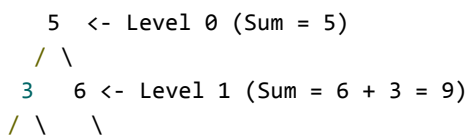
Example: A = [1, 3, 4, 5, 6, 7]

One of the possible output trees:

3) Write a function that takes as input the root to a binary tree and outputs the sum of each level. Function prototype is as follows: (7)

```
void level_sum(node * root){
        //Definition here
}
```

Example :



```
                        1   4   7 <- Level 2 (Sum = 7 + 4 + 1 = 12)
Output : 5 9 12
```

4) Tom loves candies. He has a shelf with multiple jars of candies in them. Tom can perform K moves and in each move, he can choose any one jar out of the N jars and have half (in case the value is odd, choose integer part of the half) of the candies from the jar. As he really loves candies, he asks you to help him optimize his moves in such a way that he gets to eat maximum number of candies.
Write a function that takes as input K, N and the array A[] where A[i] is the number of candies present in the ith jar. Function prototype: (6)

```
int max_candies(int a[], int n, int k)
{
        //Definition here
}
```

```
Example:
        N = 5
        A = 5 1 12 3 2
        K = 4
In this case, Tom can choose the jars in following way
        1st move: Choose jar 3 (12 candies). So he gets to have 12 / 2 = 6 candies
        and the new configuration becomes:
                5 1 6 3 2
        2nd move: He can again choose the same jar (with 6 candies now) and
        have 6/2 = 3 candies. The new configuration is:
                5 1 3 3 2
        3rd move: He can now choose the 1st jar (with 5 candies) and have 5/2 = 2
        candies. New configuration:
                3 1 3 3 2
        4th move: He can choose 1st, 3rd or 4th jar and have 3 / 2 = 1 candy.
        Assuming he chose 1st jar, the new configuration becomes:
                2 1 3 3 2
Maximum number of candies he can have = 6 + 3 + 2 + 1 = 12.Hence function must
return 12.
```

ALL THE BEST :)