**Name – Smriti**

**Roll no. – 20bcs105**

**Assignment – 8**

**Subject – CS-201**

Q1. Write a C/C++ program to implement following variations of Quicksort

a. First element is the pivot element

b. Middle element is the pivot element

c. Select any random element to be the pivot element

For a given array display the number of iterations for each of above variation of QuickSort.

```
#include <iostream>
#include<cstdlib>
#include<time.h>

using namespace std;

void display(int arr[],int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << " ";
    }
}
void swap(int arr[], int i, int j)
{
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

int partition_last(int arr[], int l, int r)
{
```

```
        int i = l + 1;
        int piv = arr[l];
        for (int j = l + 1; j <= r; j++)
        {
            if (arr[j] < piv)
            {
                swap(arr,i,j);
                i += 1;
            }
        }
        swap(arr,l,i-1);
        return i - 1;
    }

    int partition_first(int arr[], int l, int r)
    {
        int i = l + 1;
        int piv = arr[l];
        for (int j = l + 1; j <= r; j++)
        {
            if (arr[j] < piv)
            {
                swap(arr,i,j);
                i += 1;
            }
        }
        swap(arr,l,i-1);
        return i - 1;
    }

    int partitionR(int arr[], int low, int high)
    {
        int pivot = arr[high];

        int i = (low - 1);

        for (int j = low; j <= high - 1; j++)
        {
            if (arr[j] <= pivot) {
```

```
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

int partition_random(int arr[], int low, int high)
{
    srand(time(NULL));
    int random = low + rand() % (high - low);
    swap(arr[random], arr[high]);

    return partitionR(arr, low, high);
}

void quickSort_last(int arr[], int l, int r)
{
    if (l < r)
    {
        int pi = partition_last(arr, l, r);
        quickSort_last(arr, l, pi - 1);
        quickSort_last(arr, pi + 1, r);
    }
}

void quickSort_first(int arr[], int l, int r)
{
    if (l < r)
    {
        int pi = partition_first(arr, l, r);
        quickSort_first(arr, l, pi - 1);
        quickSort_first(arr, pi + 1, r);
    }
}

void quickSort_random(int arr[], int l, int r)
```

```
{
    if(l<r)
    {
        int pi=partition_random(arr,l,r);
        quickSort_random(arr,l,pi-1);
        quickSort_random(arr,pi+1,r);
    }
}
int main()
{
    cout << "Enter number of elements :\t";
    int size;
    cin >> size;
    int arr[size];
    cout << "Enter elements in array" << endl;
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
    }
    cout<<"\nTaking last element as pivot!\n";
    quickSort_last(arr, 0, size - 1);
    display(arr,size);
    cout<<"\nTaking first element as pivot!\n";
    quickSort_first(arr,0,size-1);
    display(arr,size);
    cout<<"\nTaking random element as pivot!\n";
    quickSort_random(arr,0,size-1);
    display(arr,size);


    return 0;
}
```

```
PS H:\4th sem\CS201\lab-ass8> cd "h:\4th sem\CS201\lab-ass8\" ; if ($?) { g++ 1.cpp -o 1 } ; if ($?) { .\1 }
Enter number of elements :      5
Enter elements in array
4
2
1
5
3

Taking last element as pivot!
1 2 3 4 5
Taking first element as pivot!
1 2 3 4 5
Taking random element as pivot!
1 2 3 4 5
```

Q2. Write a C/C++ program

a. To construct a binary search tree of integers.

b. To traverse the tree using all the methods, i.e., inorder, preorder, and postorder.

c. To display the elements in the tree.

```cpp
#include<iostream>
#include<queue>
#include<bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    Node* left;
    Node* right;
};

class BST
{
    public:
    Node* root;
    BST()
    {
        root=NULL;
    }
```

```
    Node* insert(Node* temp,int x);
    void display();
    void inorder(Node* temp);
    void preorder(Node* temp);
    void postorder(Node* temp);
};


Node* BST::insert(Node* temp,int x)
{
    if(temp==NULL)
    {
        Node* node=new Node;
        node->data=x;
        node->left=NULL;
        node->right=NULL;
        return node;
    }
    if(x>temp->data)
    {
        temp->right=insert(temp->right,x);
    }
    else
    {
        temp->left=insert(temp->left,x);
    }
    return temp;

}

void BST::display()
{
    if(root==NULL)
    {
        cout<<"\nBinary Search Tree is empty!!!\n";
        return;
    }
    queue<Node*> q;
    q.push(root);
```

```
    while(!q.empty())
    {
        Node* temp=q.front();
        q.pop();
        cout<<temp->data<<"\t";
        if(temp->left)
        q.push(temp->left);
        if(temp->right)
        q.push(temp->right);
    }
}

void BST::inorder(Node* temp)
{
    if(temp==NULL)
    return;
    inorder(temp->left);
    cout<<temp->data<<"\t";
    inorder(temp->right);

}

void BST::preorder(Node* temp)
{
    if(temp==NULL)
    return;
    cout<<temp->data<<"\t";
    preorder(temp->left);
    preorder(temp->right);
}

void BST::postorder(Node* temp)
{
    if(temp==NULL)
    return;
    postorder(temp->left);
    postorder(temp->right);
    cout<<temp->data<<"\t";
}
```

```
int main()
{
    int choice;
    BST obj;
    do
    {
        cout<<"\nMENU:\n\n1. Insert\n2. Display Level
Wise\n3. Inorder\n4. Preorder\n5. Postorder\n6.
Exit\n\nEnter your choice....\t";
        cin>>choice;
        switch(choice)
        {
            case 1: int x;
                    cout<<"\nEnter the number your want to
insert:\t";
                    cin>>x;
                    obj.root=obj.insert(obj.root,x);
                    break;
            case 2: obj.display();
                    break;
            case 3: obj.inorder(obj.root);
                    break;
            case 4: obj.preorder(obj.root);
                    break;
            case 5: obj.postorder(obj.root);
                    break;

            case 6: break;


            default: cout<<"\nWrong choice Entered!!!\n";
                    break;

        }
    }while(choice!=6);
}
```

```
1. Insert
2. Display Level Wise
3. Inorder
4. Preorder
5. Postorder
6. Exit

Enter your choice....   2
8       3       10      1       6       14      4       7       13
MENU:

1. Insert
2. Display Level Wise
3. Inorder
4. Preorder
5. Postorder
6. Exit

Enter your choice....   3
1       3       4       6       7       8       10      13      14
MENU:

1. Insert
2. Display Level Wise
3. Inorder
4. Preorder
5. Postorder
6. Exit

Enter your choice....   4
8       3       1       6       4       7       10      14      13
MENU:

1. Insert
2. Display Level Wise
3. Inorder
4. Preorder
5. Postorder
6. Exit
```

```
Enter your choice....   5
1       4       7       6       3       13      14      10      8
MENU:

1. Insert
2. Display Level Wise
3. Inorder
4. Preorder
5. Postorder
6. Exit

Enter your choice....
```

Q 3. Write a simple recursive function to count the number of nodes in a Tree. Write a countNode() function that prints the number of nodes in a tree and takes the address of root node as an argument

```
int Tree::countNode(Node* root)
{

    if(root==NULL)
    return 0;
    int count=0;
    for(int i=0;i<temp->child.size();i++)
    {
        count= 1+countNodes(root->child[i]);
    }
    return count;
}
```

Q 4. Write a simple recursive function to determine the type of node (leaf, root, non-leaf node). Input a value from user, search the node with the value in a binary search tree and do the following:

a. If the node is a leaf, then print it

b. If the node is the root, then print the entire tree

c. If the node is an internal node (non-leaf) node then print its immediate children

```
void BST:: check(Node* root,Node* temp)     //takes required
node's address
{
    if(temp==root)
    {
        inorder(root);
    }
    else if(temp->left==NULL&&temp->right==NULL)
    cout<<temp->data<<endl;
    else
```

```
    {
        cout<<"\nImmidiate children are:\n";
        if(temp->left==NULL)
        cout<<"NULL &";
        else
        cout<<temp->left->data<<" & ";
        if(temp->right==NULL)
        cout<<" NULL";
        else
        cout<<temp->right->data;
    }
}

Node* BST::search(Node* temp,int x)     // to search element
and return it's node
{

    if (temp == NULL || temp->data == x)
       return temp;


    if (x>temp->data)
       return search(temp->right, x);

    return search(temp->left, x);

}
```