Design and Analysis of Algorithms Assignment - 3

Name: Dhanraj Kore

Div: TY B

Roll No: 60

Batch : B-3

Majority Search

A majority element in an array A[] of size n is an element that appears more than n/2 times

Approach 1:

Run two loops and keep track of the maximum count for all different elements

CODE :

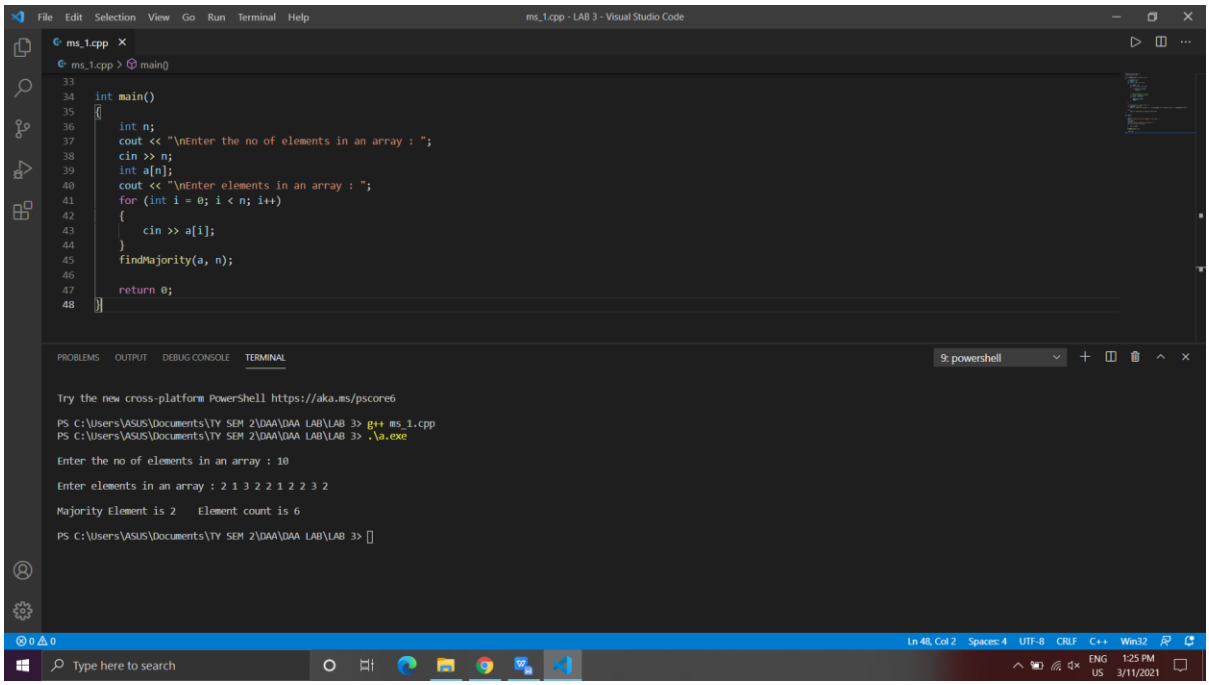
```
#include <bits/stdc++.h>

using namespace std;

void findMajority(int arr[], int n)

{

    int maxCount = 0;

    int index = -1;

    for (int i = 0; i < n; i++)

    {

        int count = 0;

        for (int j = 0; j < n; j++)

        {

            if (arr[i] == arr[j])

                count++;

        }
```

```
        // update maxCount if count of
        // current element is greater
        if (count > maxCount)
        {
            maxCount = count;
            index = i;
        }
    }
    // if maxCount is greater than n/2
    if (maxCount > n / 2)
        cout << "\nMajority Element is " << arr[index] << "\t Element count is "<<maxCount<<"\n\n";
    else
        cout << "\nNo Majority Element found!\n\n";
}
int main()
{
    int n;
    cout << "\nEnter the no of elements in an array : ";
    cin >> n;
    int a[n];
    cout << "\nEnter elements in an array : ";
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
    }
    findMajority(a, n);
    return 0;
}
```

*Time Complexity: O(n*n)*

Space Complexity : O(1)

O/P:



Approach 2: *Divide & Conquer Approach*

CODE :

```
#include <bits/stdc++.h>

using namespace std;

int max_num(vector<int> v, vector<int> arr, int n, int m)

{

    if ((m - n + 1) % 2 != 0)

    {

        int l = v[0];

        int count1 = 0;

        for (int i = 0; i < arr.size(); i++)

        {

            if (arr[i] == l)

                count1++;

        }

        if (count1 > (arr.size() - 1) / 2)
```

```
        {
            return (l);
        }
        v.erase(v.begin());
        m--;
    }
    if ((m - n + 1) == 0)
        return (-1);
    if ((m - n + 1) == 2 and v[m] == v[n])
    {
        return (v[m]);
    }
    else if ((m - n + 1) == 2 and v[m] != v[n])
        return (-1);
    else
    {
        int p, q;
        int k = (m - n) / 2;
        p = max_num(v, arr, n, k);
        q = max_num(v, arr, k + 1, m);
        if (p == -1 and q == -1)
            return (-1);
        else if (p == -1 and q != -1)
        {
            int count = 0;
            for (int i = n; i <= m; i++)
            {
                if (q == v[i])
                    count++;
            }
            if (count > (m - n + 1) / 2)
                return (q);
            else
                return (-1);
        }
        else if (p != -1 and q == -1)
```

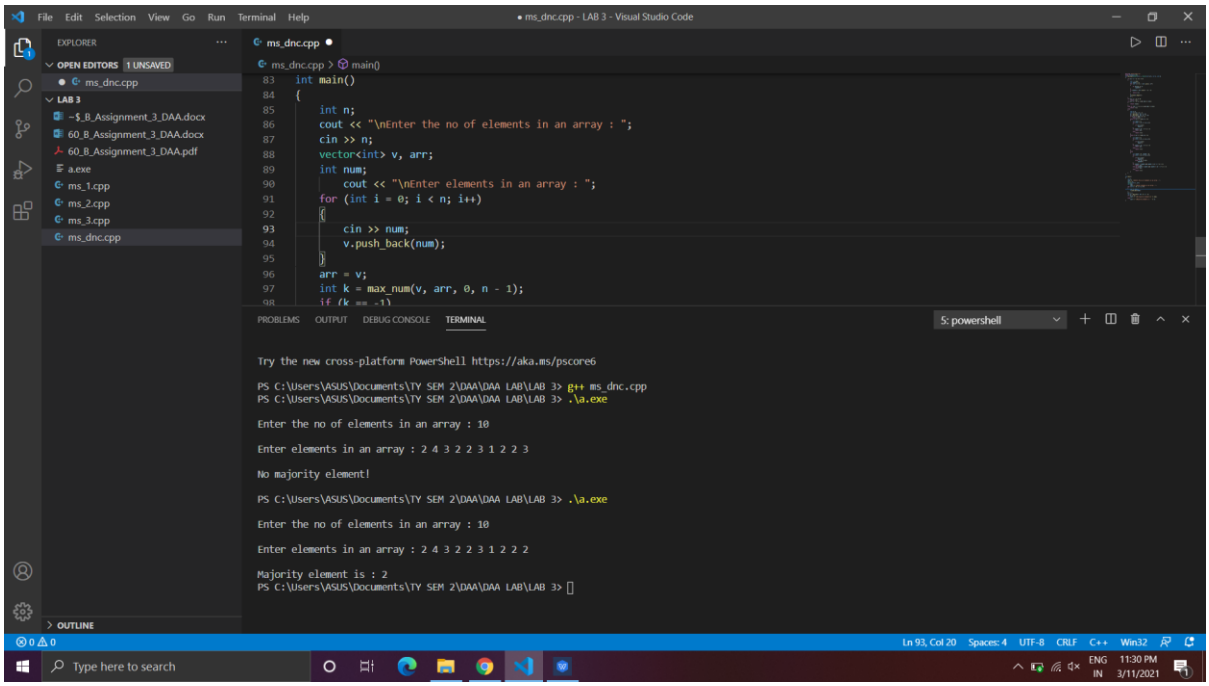
```
        {
            int count = 0;
            for (int i = n; i <= m; i++)
            {
                if (p == v[i])
                    count++;
            }
            if (count > (m - n + 1) / 2)
                return (p);
            else
                return (-1);
        }
        else
        {
            int count1 = 0, count2 = 0;
            for (int i = n; i <= m; i++)
            {
                if (p == v[i])
                    count1++;
                else if (q == v[i])
                    count2++;
            }
            if (count1 > count2 and count1 > (m - n + 1) / 2)
                return (p);
            else if (count2 > count1 and count2 > (m - n + 1) / 2)
                return (q);
            else
                return (-1);
        }
    }
}
int main()
{
    int n;
    cout << "\nEnter the no of elements in an array : ";
    cin >> n;
```

```
    vector<int> v, arr;
    int num;
        cout << "\nEnter elements in an array : ";
    for (int i = 0; i < n; i++)
    {
        cin >> num;
        v.push_back(num);
    }
    arr = v;
    int k = max_num(v, arr, 0, n - 1);
    if (k == -1)
        cout << "\nNo majority element!\n" << endl;
    else
        cout << "\nMajority element is : " << k;
}
```

Time Complexity: O(nlogn)

Space Complexity : O(n)

O/P:



Approach 3: *Using BST*

CODE:

```
#include <bits/stdc++.h>

using namespace std;


struct node {

    int key;

    int c = 0;

    struct node *left, *right;

};


// create BST node

struct node* newNode(int item)

{

    struct node* temp

        = (struct node*)malloc(sizeof(struct node));

    temp->key = item;

    temp->c = 1;

    temp->left = temp->right = NULL;

    return temp;

}


// BST insert

struct node* insert(struct node* node, int key, int& ma)

{

    if (node == NULL) {

        if (ma == 0)

            ma = 1;
```

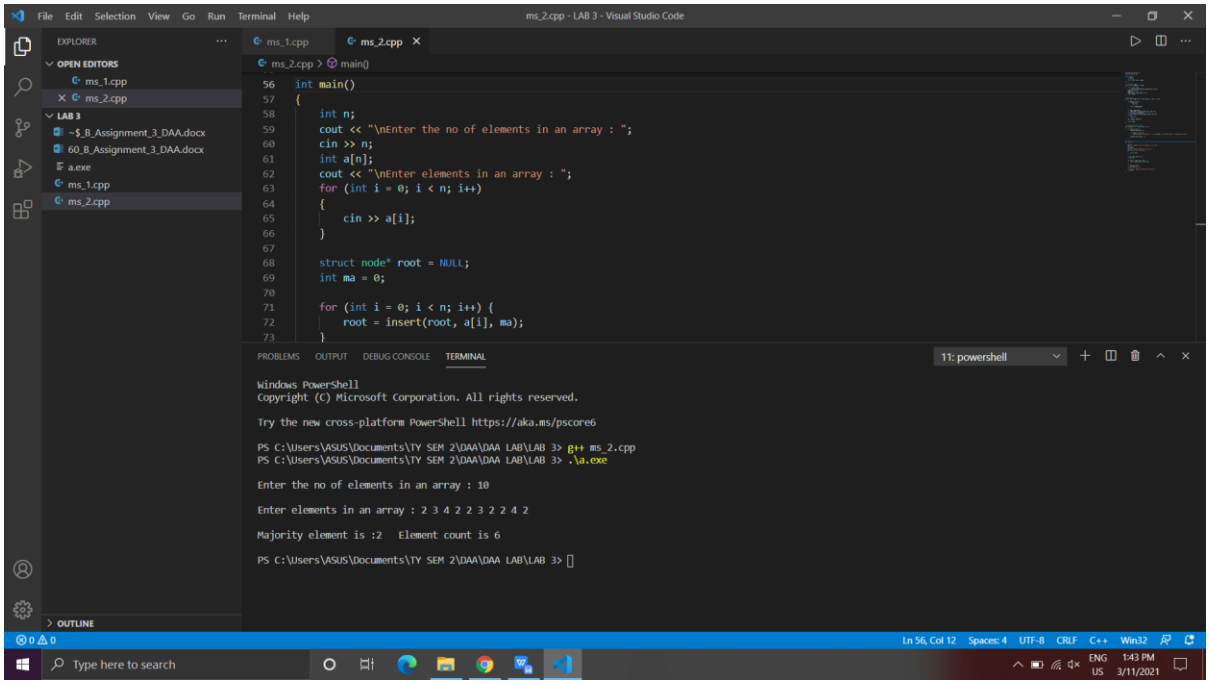
```
        return newNode(key);
    }

    if (key < node->key)
        node->left = insert(node->left, key, ma);
    else if (key > node->key)
        node->right = insert(node->right, key, ma);
    else
        node->c++;

    ma = max(ma, node->c);

    return node;
}

// inorder traversal of BST
void inorder(struct node* root, int s)
{
    if (root != NULL) {
        inorder(root->left, s);

        if (root->c > (s / 2))
            cout<<"\nMajority element is :"<< root->key<< "\t Element count is "<<root->c<<"\n\n";

        inorder(root->right, s);
    }
}

int main()
{
    int n;
    cout << "\nEnter the no of elements in an array : ";
    cin >> n;
    int a[n];
    cout << "\nEnter elements in an array : ";
    for (int i = 0; i < n; i++)
```

```
    {
        cin >> a[i];
    }

    struct node* root = NULL;
    int ma = 0;

    for (int i = 0; i < n; i++) {
        root = insert(root, a[i], ma);
    }

    if (ma > (n / 2))
        inorder(root, n);
    else
        cout << "Majority element not found!\n";
    return 0;
}
```

Time Complexity: O(nlogn)

Space Complexity : O(n)

O/P:



Approach 4: *Using Moore's Voting Algorithm*

CODE:

```
#include <bits/stdc++.h>

using namespace std;


//Candidate function

int findCandidate(int a[], int size)

{

    int maj_index = 0, count = 1;

    for (int i = 1; i < size; i++) {

        if (a[maj_index] == a[i])

            count++;

        else

            count--;

        if (count == 0) {

            maj_index = i;

            count = 1;

        }

    }
```

```
    return a[maj_index];
}

// check candidate
bool isMajority(int a[], int size, int cand)
{
    int count = 0;
    for (int i = 0; i < size; i++)

        if (a[i] == cand)
            count++;

    if (count > size / 2)
        return 1;

    else
        return 0;
}

void printMajority(int a[], int size)
{
    // Find the candidate for Majority
    int cand = findCandidate(a, size);

    // Print the candidate if it is Majority
    if (isMajority(a, size, cand))
        cout << "\nMajority element is : " << cand << "\n";

    else
        cout << "\nNo Majority Element\n";
}

int main()
{
    int n;
    cout << "\nEnter the no of elements in an array : ";
```

```
    cin >> n;

    int a[n];

    cout << "\nEnter elements in an array : ";

    for (int i = 0; i < n; i++)

    {

        cin >> a[i];

    }


    printMajority(a, n);


    return 0;

}
```

Time Complexity: O(n)

Space Complexity : O(1)

O/P: