NAME: Atharva Kangralkar

ROLL NO: 54

CLASS: TY - CSAIML

PRN NO: 12311493

BATCH: 2

LAB ASSIGNMENT 3

AIM: Analysis of finding out majority element from an array

Definition: an element x in array A is called majority element if it appears greater that ⌊n/2⌋ times where n is size of array

1)brute-force

2) sort method

3) divide and conquer

4) Boyer-Moore voting algorithm

STEP1: Pseudocode

1)brute-force

```
function majorityElementBruteForce(A):
  n = length(A)
  for i from 0 to n-1:
    count = 0
    for j from 0 to n-1:
      if A[j] == A[i]:
        count = count + 1
    if count > floor(n/2):
      return A[i]
  return "No Majority Element"
```

2) sort method

```
function majorityElementSort(A):
```

```
sort(A)                    // O(n log n)
candidate = A[floor(n/2)]   // middle element
count = 0
for i from 0 to n-1:
    if A[i] == candidate:
        count = count + 1
if count > floor(n/2):
    return candidate
else:
    return "No Majority Element"
```

3) divide and conquer

```
function majorityElementDivideAndConquer(A, left, right):
    if left == right:
        return A[left]

    mid = (left + right) // 2
    leftMajor = majorityElementDivideAndConquer(A, left, mid)
    rightMajor = majorityElementDivideAndConquer(A, mid+1, right)

    if leftMajor == rightMajor:
        return leftMajor

    countLeft = countFrequency(A, leftMajor, left, right)
    countRight = countFrequency(A, rightMajor, left, right)

    if countLeft > (right - left + 1) // 2:
        return leftMajor
    if countRight > (right - left + 1) // 2:
        return rightMajor
    return "No Majority Element"


function countFrequency(A, candidate, left, right):
    count = 0
    for i from left to right:
        if A[i] == candidate:
            count = count + 1
    return count
```

4) Boyer-Moore voting algorithm

```
function majorityElementBoyerMoore(A):
    // Phase 1: Find candidate
    candidate = None
    count = 0
    for num in A:
        if count == 0:
            candidate = num
            count = 1
        else if num == candidate:
            count = count + 1
        else:
            count = count - 1


    // Phase 2: Verify candidate
    count = 0
    for num in A:
        if num == candidate:
            count = count + 1
```

```
if count > floor(n/2):
    return candidate
else:
    return "No Majority Element"
```

STEP 2: Code

1)brute-force

```
#include <iostream>
#include <vector>
using namespace std;

int findMajority(const vector<int> &arr)
{
    int n = arr.size();

    for(int i = 0; i < n; i++)
    {
        int cnt = 0;
        for(int j = 0; j < n; j++)
        {
            if(arr[j] == arr[i])
                cnt++;
        }

        if(cnt > n/2)
            return arr[i];
    }
    return -1;
}

int main()
{
    vector<int> arr;
    int x;

    cout << "Enter elements: ";

    while(cin >> x)
    {
        arr.push_back(x);
    }

    int ans = findMajority(arr);

    if(ans != -1)
```

```
        cout << "Majority element is: " << ans << endl;
    else
        cout << "No majority element found." << endl;

    return 0;
}
```

2) sort method

```
#include <iostream>
#include <vector>
using namespace std;

void quickSort(vector<int>& arr, int low, int high) {
    if(low < high){
        int pivot = arr[high];
        int i = low - 1;

        for(int j = low; j < high; j++){
            if(arr[j] <= pivot){
                i++;
                swap(arr[i], arr[j]);
            }
        }
        swap(arr[i + 1], arr[high]);
        int p = i + 1;

        quickSort(arr, low, p - 1);
        quickSort(arr, p + 1, high);
    }
}

int main() {
    vector<int> arr;
    int x;

    cout << "Enter the elements: " << endl;
    while (cin >> x) {
        arr.push_back(x);
    }

    quickSort(arr, 0, arr.size() - 1);

    int majority = arr[arr.size() / 2];
    cout << "Majority element: " << majority << endl;

    return 0;
}
```

3) divide and conquer

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

int countInRange(vector<int>& nums, int num, int start, int end) {
   int count = 0;
   for (int i = start; i <= end; i++) {
      if (nums[i] == num) count++;
   }
   return count;
}

int majorityElementRec(vector<int>& nums, int start, int end) {
   // Base case: only one element
   if (start == end) {
      return nums[start];
   }

   int mid = start + (end - start) / 2;
   int left = majorityElementRec(nums, start, mid);
   int right = majorityElementRec(nums, mid + 1, end);

   // If both halves agree on the majority element
   if (left == right) return left;

   // Otherwise, count each candidate in the current range
   int leftCount = countInRange(nums, left, start, end);
   int rightCount = countInRange(nums, right, start, end);

   return (leftCount > rightCount) ? left : right;
}

int majorityElement(vector<int>& nums) {
   return majorityElementRec(nums, 0, nums.size() - 1);
}

int main() {
   int n;
   cout << "Enter size of array: ";
   cin >> n;

   vector<int> nums(n);
   cout << "Enter " << n << " elements: ";
   for (int i = 0; i < n; i++) {
      cin >> nums[i];
   }

   int result = majorityElement(nums);
   cout << "Majority Element: " << result << endl;
```

```
    return 0;
}
```

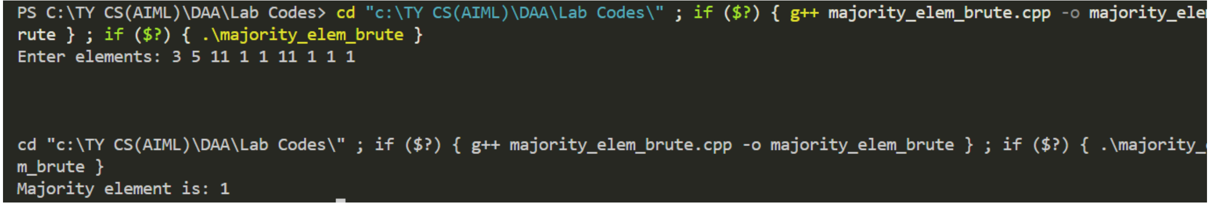
4) Boyer-Moore voting algorithm

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

int majorityElement(vector<int> v) {
    int cnt = 0;
    int el;

    for (int i = 0; i < v.size(); i++) {
        if (cnt == 0) {
            cnt = 1;
            el = v[i];
        } else if (v[i] == el) {
            cnt++;
        } else {
            cnt--;
        }
    }

    int cnt1 = 0;
    for (int i = 0; i < v.size(); i++) {
        if (v[i] == el) cnt1++;
    }

    if (cnt1 > (v.size() / 2)) {
        return el;
    }

    return -1;
}

int main() {
    int n;
    cout << "Enter size of array: ";
    cin >> n;

    vector<int> v(n);
    cout << "Enter elements: ";
    for (int i = 0; i < n; i++) {
        cin >> v[i];
    }

    int ans = majorityElement(v);
    if (ans != -1)
        cout << "Majority Element: " << ans << endl;
    else
```

```
        cout << "No Majority Element found" << endl;

    return 0;
}
```

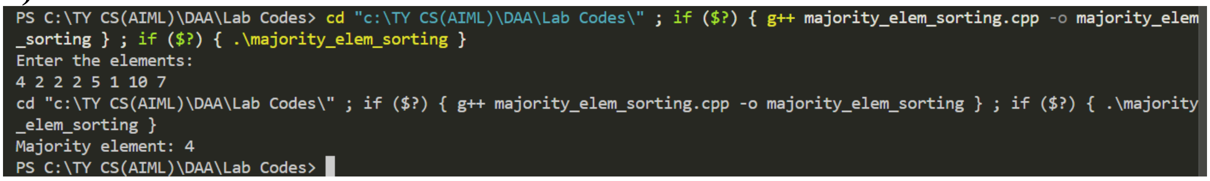
STEP 3:Output

1)brute-force



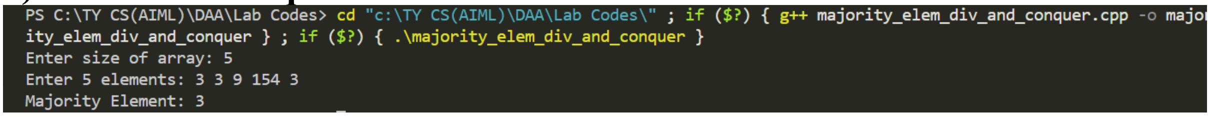
```
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ majority_elem_brute.cpp -o majority_ele
rute } ; if ($?) { .\majority_elem_brute }
Enter elements: 3 5 11 1 1 11 1 1 1



cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ majority_elem_brute.cpp -o majority_elem_brute } ; if ($?) { .\majority_
m_brute }
Majority element is: 1
```

2) sort method



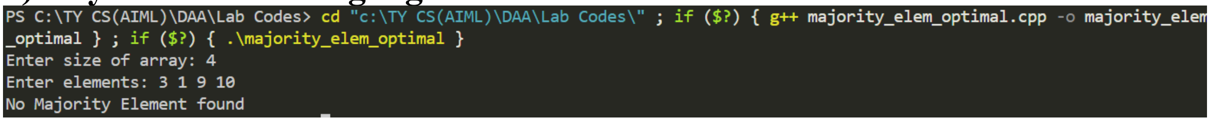
```
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ majority_elem_sorting.cpp -o majority_elem
_sorting } ; if ($?) { .\majority_elem_sorting }
Enter the elements:
4 2 2 2 5 1 10 7
cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ majority_elem_sorting.cpp -o majority_elem_sorting } ; if ($?) { .\majority
_elem_sorting }
Majority element: 4
PS C:\TY CS(AIML)\DAA\Lab Codes>
```

3) divide and conquer



```
 PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ majority_elem_div_and_conquer.cpp -o major
 ity_elem_div_and_conquer } ; if ($?) { .\majority_elem_div_and_conquer }
 Enter size of array: 5
 Enter 5 elements: 3 3 9 154 3
 Majority Element: 3
```

4) Boyer-Moore voting algorithm



```
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ majority_elem_optimal.cpp -o majority_elem
_optimal } ; if ($?) { .\majority_elem_optimal }
Enter size of array: 4
Enter elements: 3 1 9 10
No Majority Element found
```

STEP 4: Time Complexity Analysis

1)brute-force

Best case:

T(n)=O(n)

Worst case:

T(n)=O(n^2)

2) sort method

Best case:

T(n) = O(nlogn)

Worst case:
T(n) = O(nlogn)

3) divide and conquer
Best case:
T(n) = O(nlogn)

Worst case:
T(n) = O(nlogn)

4) Boyer-Moore voting algorithm
Best case:
O(n)

Worst case:
O(n)

STEP 5: Comparison Table

SrNO	Algorithm method	Time Complexity
1	brute-force	O(n^2)
2	sort	O(nlogn)
3	divide and conquer	O(nlogn)
4	Boyer-Moore voting algorithm (optimal solution)	O(n)