(CS702: Computing Lab) Assignment 1

Name of Student:Akshit Pokhriyal
Roll No.:2520535

1 Qns 1.

Given an array of nums with n objects colored red, white, or blue, sort them in place so that objects of the same color are adjacent, with the colors in red, white, and blue. We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively. You must solve this problem without using the library's sort function.

1.1 Algorithm

Algorithm to sort three different colored objects (0,1,2):

Algorithm 1 Counting Sort for 0,1,2

1: Start
2: Initialize three counters: redCount, whiteCount, blueCount
3: Traverse the array and count the frequency of 0's, 1's, and 2's
4: Overwrite the array:

- Fill first redCount positions with 0
- Fill next whiteCount positions with 1
- Fill remaining blueCount positions with 2

5: Print the modified array
6: Stop =0

1.2 Code

C++ program to sort colored objects:

```cpp
#include <iostream>
using namespace std;
int main() {
    int n;
```

```
    cout<<"Enter amount of objects to store:"<<endl;
    cin>>n;
    int arr[n];
    cout<<"Enter values to store in array where Red(0),
       White(1) and Blue(2)"<<endl;
    for(int i=0;i<n;i++){cin>>arr[i];}
    int a=0,b=0,c=0;
    for(auto it:arr){
        if(it==0) a++;
        else if(it==1)b++;
        else if(it==2)c++;
    }
    int i=0;
    while(a--){arr[i++]=0;}
    while(b--){arr[i++]=1;}
    while(c--){arr[i++]=2;}
    cout<<"Sorted Array:"<<endl;
    for(auto it:arr){cout<<it<<" ";}
    return 0;
}
```

1.3 Test Cases

The test cases are given in Table 1

Table 1: Test Cases for Sorting Colors

N	Input Array	Output Array
6	2 0 2 1 1 0	0 0 1 1 2 2
3	2 0 1	0 1 2
5	2 0 1 2 2	0 1 2 2 2
10	2 0 1 0 1 2 1 2 0 0	0 0 0 0 1 1 1 2 2 2
7	2 0 1 0 0 0 1	0 0 0 0 1 1 2

2 Qns 2.

Given an array of prices where prices[i] is the price of a given stock on the ith day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return the maximum profit you can achieve from this transaction. If you cannot make a profit, return 0.

2.1 Algorithm

Algorithm to find maximum profit from a single buy-sell transaction:

Algorithm 2 Max Profit in Stock

1: Start
2: Initialize variable `minPrice` as first element and `maxProfit` as 0
3: Traverse array from day 2 to n:
4: For each price:

- Calculate profit = price - minPrice
- Update maxProfit if profit is larger
- Update minPrice if current price is smaller

5: Return maxProfit
6: Stop =0

2.2 Code

C++ program to sort colored objects:

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

int main(){
    int n;
    cout<<"Enter number of elements:"<<endl;
    cin>>n;
    cout<<"Enter stock prices(elements of array):"<<endl;
    int arr[n];
```

```
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int ans=0;
    int mini=arr[0];
    for(int i=1;i<n-1;i++){
        int sum=arr[i]-mini;
        mini=min(mini,arr[i]);
        ans=max(sum,ans);
    }

    cout<<ans<<endl;
    return 0;
}
```

2.3 Test Cases

Table 2: Test Cases for Stock Buy-Sell

N	Input Prices	Max Profit
6	7 1 5 3 6 4	5
5	7 6 4 3 1	0
5	2 4 1 7 5	6
4	1 2 3 4	3
3	2 1 2	1

3 Qns 3.

Given an integer array of size n, find all elements that appear more than $n/3$ times.

3.1 Algorithm

Algorithm to find elements occurring more than $\lfloor n/3 \rfloor$ times:

Algorithm 3 Boyer–Moore Majority Vote (n/3 variant)

1: Start
2: Initialize two candidate variables (m1, m2) and counters (c1, c2)
3: Traverse array:

- If current element = m1, increment c1
- Else if = m2, increment c2
- Else if c1 = 0, set m1 = element, c1=1
- Else if c2 = 0, set m2 = element, c2=1
- Else decrement both counters

4: After traversal, recount occurrences of m1 and m2
5: Output candidates with frequency ¿ n/3
6: Stop =0

3.2 Code

C++ program to sort colored objects:

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

int main(){
    int n;
    cout<<"Enter number of elements:"<<endl;
    cin>>n;
    cout<<"Enter elements of array:"<<endl;
    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int m1,m2,c1,c2;
    m1=arr[0],c1=1,c2=1;
    for(auto it:arr){
        if(it!=m1){
            m2=it;
```

```
            break;
        }
    }
    for(int i=1;i<n;i++){
        if(arr[i]==m1) c1++;
        else if(arr[i]==m2) c2++;
        else if(c1<0){
            m1=arr[i];
            c1=1;
        }
        else if(c2<0){
            m2=arr[i];
            c2=0;
        }
        else{
            c1--;
            c2--;
        }
    }
    int count1 = 0, count2 = 0;
    for(int i = 0; i < n; i++) {
        if(arr[i] == m1) count1++;
        else if(arr[i] == m2) count2++;
    }
    int cnt = n/3;
    if(count1 > cnt) cout << m1 << " ";
    if(count2 > cnt) cout << m2 << " ";
    return 0;
}
```

3.3 Test Cases

4 Qns 4.

Given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once. Return the single element that occurs only once. The solution must run in O(log n) time and O(1) space.

Table 3: Test Cases for Majority Elements ($n/3$)

N	Input Array	Output
7	3 2 3 3 4 5 3	3
5	1 2 3 1 2	(none)
8	2 2 1 1 1 2 2 3	2
6	1 1 1 2 2 3	1
10	4 4 4 4 5 5 5 5 6 7	4 5

4.1 Algorithm

Algorithm to find only element occuring once from the array:

Algorithm 4

1: Start
2: Create a function for binary search;
3: Modify the binary function to check for duplicate values next to or previous to mid point and search for single element
4: return the element that is the only single element in the array
5: Stop =0

4.2 Code

C++ program to sort colored objects:

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

int main(){
    int n;
    cout<<"Enter number of elements"<<endl;
    cin>>n;
    if(n%2==0) return 0;
    cout<<"Enter an array with pair of integers except for 1
        integer:"<<endl;
    int arr[n];
    for(int i=0;i<n;i++) cin>>arr[i];

```

```
    int low=0,high=n;
    int ans=-1;
    while(low<high){
        int mid=low+((high-low)/2);
        if(arr[mid]!=arr[mid+1] && arr[mid]!=arr[mid-1]){
            ans=mid;
            break;
        }
        else if(arr[mid]==arr[mid+1]){
            if(mid%2==0) low=mid+1;
            else high=mid-1;
        }
        else{
            if(mid%2==0) high=mid-1;
            else low=mid+1;
        }
    }
    ans=(ans==-1?-1:arr[ans]);
    cout<<ans<<endl;

}
```

4.3 Test Cases

Table 4: Test Cases for Single Non-Duplicate Element

N	Input Array	Unique Element
9	1 1 2 3 3 4 4 8 8	2
7	3 3 7 7 10 11 11	10
5	0 0 1 1 2	2
11	1 2 2 3 3 4 4 5 5 6 6	1
1	99	99

5 Qns 5.

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value. If the target is not found in the array, return [-1, -1]. The algorithm must have an O(log n) runtime complexity

5.1 Algorithm

Algorithm to find index of starting and ending position of target using binary search:

Algorithm 5

1: Start
2: Create a function that uses binary search to find the first occurrence of the given target, else returns -1
3: Create a function that uses binary search to find the last occurrence of the given target, else returns -1
4: Return the first and last occurrence using those functions
5: Stop =0

5.2 Code

C++ program to sort colored objects:

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

int firstOccurence(int low,int high,int arr[],int target){
    int ind=-1;
    while(low<=high){
        int mid=low+((high-low)/2);
        if(arr[mid]==target){
            ind=mid;
            high=mid-1;
        }
        if(arr[mid]>target) high=mid-1;
        else low=mid+1;
    }
```

```
15      return ind;
16  }
17
18  int lastOccurence(int low,int high,int arr[],int target){
19      int ind=-1;
20      while(low<=high){
21          int mid=low+((high-low)/2);
22          if(arr[mid]==target){
23              ind=mid;
24              low=mid+1;
25          }
26          if(arr[mid]<target) low=mid+1;
27          else if(arr[mid]>target) high=mid-1;
28      }
29      return ind;
30  }
31
32  int main(){
33      int n;
34      cout<<"Enter number of elements"<<endl;
35      cin>>n;
36      cout<<"Enter elements in increasingly sorted order:"<<
            endl;
37      int arr[n]={0};
38      for(int i=0;i<n;i++){
39          cin>>arr[i];
40      }
41
42      int target;
43      cout<<"Enter value of element whose position is to be
            found"<<endl;
44      cin>>target;
45
46      int start=firstOccurence(0,n,arr,target),end=
            lastOccurence(0,n,arr,target);
47
48      cout<<start<<" "<<end<<endl;
49
```

```
    return 0;
}
```

5.3 Test Cases

Table 5: Test Cases for First and Last Position of Element in Sorted Array

No. of elements	Array	Target	Output [Start, End]
6	5 7 7 8 8 10	8	3 4
6	5 7 7 8 8 10	6	-1 -1
0		0	-1 -1
6	7 7 7 7 7 7	7	0 5
3	0 1 2	1	1 1

6 Qns 6.

Search in Rotated Sorted Array. There is an integer array nums sorted in ascending order (with distinct values). The array is rotated at an unknown pivot index k (0 ¡= k ¡ n). Given the array nums and an integer target, return the index of target if it is in nums, or -1 if it is not. You must write an algorithm with O(log n) runtime complexity

6.1 Algorithm

Algorithm to search target in rotated sorted array:

6.2 Code

C++ program to sort colored objects:

```
#include<iostream>
using namespace std;
int main(){
    int n;
    cout<<"Enter no. of elements"<<endl;
    cin>>n;
```

Algorithm 6

1: Start
2: Input rotated sorted array and target
3: Initialize low = 0, high = n-1
4: While low $\leq$ high
5: Compute mid = (low + high)/2
6: If arr[mid] == target, return mid
7: If left half is sorted:
8: If target lies in left half, set high = mid-1
9: Else set low = mid+1
10: Else right half is sorted:
11: If target lies in right half, set low = mid+1
12: Else set high = mid-1
13: Return -1
14: Stop =0

```
7       int arr[n];
8       cout<<"Enter elements of array"<<endl;
9       for(int i=0;i<n;i++){cin>>arr[i];}
10      int target;
11      cout<<"Enter elements to search"<<endl;
12      cin>>target;
13
14      int low=0;
15      int high=n-1;
16      int ind=-1;
17      while(low<=high){
18          int mid=low+(high-low)/2;
19          if(arr[mid]==target){
20              ind=mid;
21              break;
22          }
23          else if(arr[mid]>target){
24              if(arr[mid]>=arr[high]) low=mid+1;
25              else high=mid-1;
26          }
27          else{
```

```
28                 if(arr[mid]<arr[low]) high=mid-1;
29                 else low=mid+1;
30             }
31         }
32
33         cout<<ind<<endl;
34
35         return 0;
36     }
```

6.3 Test Cases

Table 6: Test Cases for Search in Rotated Sorted Array

No. of elements	Array	Target	Output
7	4 5 6 7 0 1 2	0	4
7	4 5 6 7 0 1 2	3	-1
1	1	0	-1
3	2 0 1	1	2
5	6 7 8 1 2	7	1

7 Qns 7.

Median of Two Sorted Arrays. Given two sorted arrays, nums1 and nums2, of size m and n, return the median of the two sorted arrays. The overall runtime complexity should be O(log (m+n)).

7.1 Algorithm

Algorithm to find median of two sorted arrays:

7.2 Code

C++ program to sort colored objects:

Algorithm 7

1: Start
2: Let nums1 be the smaller array, nums2 the larger
3: Use binary search on nums1 to partition both arrays
4: At each step, compute partition i in nums1 and j = (m+n)/2 - i in nums2
5: Compare left_max and right_min of partitions
6: If valid partition is found:
7: If total length is odd, return min(right1, right2)
8: Else return average of max(left1,left2) and min(right1,right2)
9: Adjust search space until partition is found
10: Stop =0

```
#include<iostream>
#include<vector>
using namespace std;
double findMedianSortedArrays(vector<int>& nums1, vector<int
    >& nums2) {
      int n1=nums1.size();
      int n2=nums2.size();
      int total=n1+n2;
      int half=total/2;
      vector<int> A=n1<n2?nums1:nums2;
      vector<int> B=n1<n2?nums2:nums1;
      int low=0,high=A.size();
      while(low<=high){
          int i=(low+high)/2;
          int j=half-i;
          int l1=i>0?A[i-1]:INT_MIN;
          int r1=i<A.size()?A[i]:INT_MAX;
          int l2=j-1>=0?B[j-1]:INT_MIN;
          int r2=j<B.size()?B[j]:INT_MAX;
          if(l1<=r2 && l2<=r1){
              if(total%2==1) return (double)min(r1,r2);
              else{return (max(l1,l2)+min(r1,r2))/2.0;}
          }
          else if(l1>r2) high=i-1;
          else low=i+1;
```

```
    }
    return -1.0;
}
int main() {
    int n, m;
    cout << "Enter number of elements in first array:" <<
       endl;
    cin >> n;
    vector<int> A(n);
    cout << "Enter elements of first array in sorted order:"
        << endl;
    for (int i = 0; i < n; i++) {cin >> A[i];}
    cout << "Enter number of elements in second array:" <<
       endl;
    cin >> m;
    vector<int> B(m);
    cout << "Enter elements of second array in sorted order:
       " << endl;
    for (int i = 0; i < m; i++) {cin >> B[i];}
    double median = findMedianSortedArrays(A, B);
    cout << "Median of the array is: " << median << endl;
    return 0;
}
```

7.3 Test Cases

Table 7: Test Cases for Median of Two Sorted Arrays

Array 1	**Array 2**	**Merged**	**Median**
1 3	2	1 2 3	2.0
1 2	3 4	1 2 3 4	2.5
0 0	0 0	0 0 0 0	0.0
1 2 3	4 5 6 7	1 2 3 4 5 6 7	4.0
2	1 3 4	1 2 3 4	2.5

8 Qns 8.

Kth Smallest Element in a Sorted Matrix. Given an $n \times n$ matrix where each row and column is sorted in ascending order, return the k^{th} smallest element in the matrix. You must find a solution with a memory complexity better than $O(n^2)$.

8.1 Algorithm

Algorithm to find kth smallest element in sorted matrix:

Algorithm 8 Kth Smallest in Sorted Matrix

1: Start
2: Input $n \times n$ sorted matrix and k
3: Set low = smallest element, high = largest element
4: While low < high
5: mid = (low + high)/2
6: Count number of elements ≤ mid (using each row)
7: If count < k, set low = mid+1
8: Else set high = mid
9: Return low
10: Stop =0

8.2 Code

C++ program to find Kth Smallest Element in Sorted Matrix:

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

// Function to count elements <= mid
int countLessEqual(vector<vector<int>>& matrix, int mid, int
    n) {
    int count = 0;
    int row = n - 1, col = 0; // start from bottom-left
    while (row >= 0 && col < n) {
        if (matrix[row][col] <= mid) {
```

```
            count += row + 1; // all elements in this column up to row
            col++;
        } else {
            row--;
        }
    }
    return count;
}

int kthSmallest(vector<vector<int>>& matrix, int k) {
    int n = matrix.size();
    int low = matrix[0][0], high = matrix[n-1][n-1];

    while (low < high) {
        int mid = low + (high - low) / 2;
        int count = countLessEqual(matrix, mid, n);
        if (count < k) low = mid + 1;
        else high = mid;
    }
    return low;
}

int main() {
    vector<vector<int>> matrix = {
        {1, 5, 9},
        {10, 11, 13},
        {12, 13, 15}
    };
    int k = 8;
    cout << "Kth Smallest Element: " << kthSmallest(matrix, k) << endl;
    return 0;
}
```

Table 8: Test Cases for Kth Smallest Element in Sorted Matrix

Matrix	k	Output
1 5 9; 10 11 13; 12 13 15	8	13
1 2; 1 3	2	1
1 3 5; 6 7 12; 11 14 14	6	11
2 6; 3 7	3	6
1 2 3; 4 5 6; 7 8 9	5	5

8.3 Test Cases

9 Qns 9.

Median of Two Sorted Arrays. Given two sorted arrays, nums1 and nums2, of size m and n, return the median of the two sorted arrays. The overall runtime complexity should be O(log (m+n)).

9.1 Algorithm

Algorithm to find maximum subarray sum (Kadane's Algorithm):

Algorithm 9

1: Start
2: Initialize max_sum = -, current_sum = 0
3: For each element x in array:
4: current_sum = max(x, current_sum + x)
5: max_sum = max(max_sum, current_sum)
6: Return max_sum
7: Stop =0

9.2 Code

C++ program to sort colored objects:

```
#include<iostream>
using namespace std;
int main(){
    int n;
```

```
5       cout<<"Enter number of elements:"<<endl;
6       cin>>n;
7       int arr[n];
8       cout<<"Enter elements of array:"<<endl;
9       for(int i=0;i<n;i++){
10          cin>>arr[i];
11      }
12
13      int sum=0;
14      int ans=0;
15      for(int i=0;i<n;i++){
16          sum+=arr[i];
17          if(sum<0) sum=0;
18          ans=max(ans,sum);
19      }
20
21      cout<<ans;
22      return 0;
23  }
```

9.3 Test Cases

Table 9: Test Cases for Maximum Subarray Sum

Array	**Max Sum**
-2 1 -3 4 -1 2 1 -5 4	6
1	1
5 4 -1 7 8	23
4 3 -8 6 -5	7
2 0 -1 -8 -5	2

10 Qns 10.

A peak element is an element that is strictly greater than its neighbors. Given a 0-indexed integer array nums, find a peak element and return its index. If the array

contains multiple peaks, return the index of any of the peaks. Assume that nums[-1] = nums[n] = -. The algorithm should run in O(log n) time.

10.1 Algorithm

Algorithm to find a peak element:

Algorithm 10

1: Start
2: Input array nums
3: If n=1 return 0
4: If nums[0] > nums[1], return 0
5: If nums[n-1] > nums[n-2], return n-1
6: Initialize low=0, high=n-1
7: While low ≤ high
8: mid = (low+high)/2
9: If nums[mid] > nums[mid-1] and nums[mid] > nums[mid+1], return mid
10: Else if nums[mid] < nums[mid+1], set low = mid+1
11: Else set high = mid-1
12: Return -1
13: Stop =0

10.2 Code

C++ program to sort colored objects:

```
#include <iostream>
using namespace std;
int bisearch(int n,int arr[]){
    if(n==1) return 0;
    if (arr[0]>arr[1]){return 0;}
    if (arr[n-1]>arr[n-2]){return n-1;}
    int low=0,high=n-1;
    while(low<high){
        int mid=low+(high-low)/2;
        if(arr[mid]>arr[mid+1] && arr[mid]>arr[mid-1]){
            return mid;}
        else if (arr[mid]<arr[mid+1]) low=mid+1;
```

```
12          else high=mid-1;
13      }
14      return -1;
15  }
16  int main(){
17      int n;
18      cout<<"Enter no. of elements"<<endl;
19      cin>>n;
20      int arr[n];
21      cout<<"Enter elemets:"<<endl;
22      for(int i=0;i<n;i++){cin>>arr[i];}
23      int ans=bisearch(n,arr);
24      cout<<ans<<endl;
25      return 0;
26  }
```

10.3 Test Cases

Table 10: Test Cases for Find Peak Element

Array	**Peak Index (any valid)**
1 2 3 1	2
1 2 1 3 5 6 4	1 or 5
5	0
2 1	0
1 2	1