Assignment – 2

6. Write a function to remove duplicates from sorted array.

Source Code: main()

```
#include <stdio.h>
#define MAX 10

void remove_duplicate(int *arr, int *num);

int main()
{
   int arr[MAX], num;
   printf("Enter how many elements you want (max %d): ", MAX);
   scanf("%d", &num);

   if (num <= 0 || num > MAX)
   {
      printf("\n\tInvalid input!\n\n");
      return 1; // Exit on invalid input
   }

   printf("Enter the array elements: ");
   for (int i = 0; i < num; i++)
   {
      scanf("%d", &arr[i]);
   }

   remove_duplicate(arr, &num);

   printf("The array after remove duplicate is: ");
   for (int i = 0; i < num; i++)
   {
      printf(" %d", arr[i]);
   }
   printf("\n");
   return 0;
}
```

Source Code: remove_duplicate()

```
void remove_duplicate(int *arr, int *num)
{
    int temp[MAX], k = 0; // Start k at 0
    for (int i = 0; i < *num; i++)
    {
        int flag = 0;              // Reset flag for each element
        for (int j = 0; j < k; j++) // Check against current unique elements
        {
            if (arr[i] == temp[j])
            {
                flag = 1;
                break; // Break early if a duplicate is found
            }
        }
        if (!flag)
        {
            temp[k++] = arr[i]; // Only add if not a duplicate
        }
    }
    *num = k; // Update the count of unique elements
    for (int i = 0; i < k; i++)
    {
        arr[i] = temp[i]; // Copy unique elements back to original array
    }
}
```

Output

```
Enter how many elements you want (max 10): 5
Enter the array elements: 1 2 2 8 8
The array after remove duplicate is:  1 2 8
```

7. Implement an algorithm to find the majority element in an array.

Source Code: main()

```
#include <stdio.h>
#define MAX 10

int main(int argc, char const *argv[])
{
   int arr[MAX], num;
   printf("How many elements you want: ");
   scanf("%d", &num);
   printf("Enter the array elements: ");
   for (int i = 0; i < num; ++i) // For inserting array elements
      scanf("%d", &arr[i]);

   int foundMajority = 0; // Flag to track if a majority element is found
   for (int i = 0; i < num; ++i) // Calculating the frequency of each element
   {
      int frequency = 0;
      for (int j = 0; j < num; ++j)
      {
         if (arr[i] == arr[j])
         {
            frequency++;
         }
      }
      if (frequency > num / 2)
      {
         printf("Majority element is: %d", arr[i]);
         foundMajority = 1; // Set the flag to indicate a majority element is found
         break;
      }
   }

   if (!foundMajority) // Check if no majority element was found
      printf("No majority element exists");

return 0;
}
```

Output

```
How many elements you want: 5
Enter the array elements: 6 5 5 4 2
Majority element is: 5
```

8. Create a program to find the largest subarray with a sum less than or equal to a given value.

Source Code: main()

```
#include <iostream>
#include <climits> // Use INT_MIN from <climits>
using namespace std;
int main()
{
   int arr[] = {1, 2, 3, 4, 5}, len = INT_MIN, givenSum = 6;
   int start = -1, end = -1; // Initialize start and end

   cout << "The array is: ";
   for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)
      cout << arr[i] << " ";
   cout << "\nThe given sum is: " << givenSum;

   for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)
   {
      int sum = 0;
      for (int j = i; j < sizeof(arr) / sizeof(arr[0]); j++)
      {
         // for (int k = i; k <= j; k++) // Change to <= to include arr[j]
         sum += arr[j];
         if (sum <= givenSum)
         {
            if (j - i + 1 > len) // Update length and start/end indices
            {
               len = j - i + 1;
               start = i;
               end = j;
            }
         }
      }
   }

   if (start == -1 && end == -1)
      cout << "No subarray found with the given sum." << endl;
   else
   {
      cout << "\nThe longest subarray of sum " << givenSum << " is: [";
      for (int i = start; i <= end; i++)
      {
         cout << arr[i];
         if (i < end)
            cout << ", ";
      }
      cout << "]" << endl;
   }

   return 0;
}
```

Output

```
The array is: 1 2 3 4 5
The given sum is: 6
The longest subarray of sum 6 is: [1, 2, 3]
```

9. Write a function to find the contiguous subarray with the largest sum (Kadane's algorithm).

Source Code: main()

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

int largestSumSubarray(int *, int *);

int main()
{
   int arr[] = {3, -4, 5, 4, -1, 7, -8};
   int size = sizeof(arr) / sizeof(arr[0]);
   cout << "The array is: ";
   for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)
      cout << arr[i] << " ";
   cout << endl;
   cout << " with sum " << largestSumSubarray(arr, &size);
}
```

Source Code: largestSumSubarray()

```
int largestSumSubarray(int *arr, int *size)
{
   int maxSum = INT_MIN, currSum = 0;
   int start = 0, end = 0, tempStart = 0;
   for (int i = 0; i < *size; i++)
   {
      currSum += arr[i];
      maxSum = max(currSum, maxSum);
      start = tempStart, end = i;
      if (currSum < 0)
      {
         currSum = 0;
         tempStart = i + 1;
      }
   }
   cout << "The largest subarray is: [ ";
   for (int i = start; i <= end; i++)
      cout << arr[i] << " ";
   cout << "]";
   return maxSum;
}
```

Output

```
The array is: 3 -4 5 4 -1 7 -8
The largest subarray is: [ 5 4 -1 7 -8 ] with sum 15
```

10. Implement an algorithm to search for an element in a sorted and rotated array (rotated binary search).

Source Code: main()

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

int binarySearch(int *, int *, int *, int *);

int main()
{
   int arr[] = {4, 5, 6, 1, 2, 3}, target = 5;
   int start = 0, end = (sizeof(arr) / sizeof(arr[0]) - 1);
   int result = binarySearch(arr, &start, &end, &target);

   cout << "The array is: ";
   for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)
      cout << arr[i] << " ";
   cout << "\nThe target element is: " << target;

   if (result == -1)
   {
      cout << "\nElement not present in the array!\n\n";
      exit(0);
   }
   cout << "\nThe element found at " << result + 1 << "th position\n\n";
}
```

Source Code: binarySearch()

```
int binarySearch(int *arr, int *start, int *end, int *target)
{
   int mid = *start + (*end - *start) / 2;

   if (arr[mid] == *target)
      return mid;

   if (arr[*start] <= arr[mid]) // Left sorted or not
   {
      if (arr[*start] <= *target <= arr[mid]) // Checks the target will exists in left or not
      {
         if (*target < arr[mid])
         {
            *end = mid - 1;
            return binarySearch(arr, start, end, target);
         }
         else
         {
            *start = mid + 1;
            return binarySearch(arr, start, end, target);
         }
      }
   }
   else // Right sorted
   {
      if (arr[mid] <= *target <= arr[*end]) // Checks the target will exists in right or not
      {
         if (*target < arr[mid])
         {
            *end = mid - 1;
            return binarySearch(arr, start, end, target);
         }
         else
         {
            *start = mid + 1;
            return binarySearch(arr, start, end, target);
         }
      }
   }
   return -1;
}
```

Output

```
The array is: 4 5 6 1 2 3
The target element is: 5
The element found at 2th position
```