TUTORIAL 3

SOLUTION

1. If A^2 mod C = **(A * A)** mod C = (**(A mod C)** * **(A mod C)**) mod C then calculate 7^{256} mod 13.

Solution:

For solving the problem visit:

https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/modular-multiplication

2. Write a program in any programming language to find the leader:

Leaders in an array

Write a program to print all the LEADERS in the array. An element is leader if it is greater than all the elements to its right side. And the rightmost element is always a leader. For example int the array {16, 17, 4, 3, 5, 2}, leaders are 17, 5 and 2. Let the input array be arr[] and size of the array be *size*.

Also calculate the complexity of the code.

Solution:

Method 1 (Simple)

Use two loops. The outer loop runs from 0 to size – 1 and one by one picks all elements from left to right. The inner loop compares the picked element to all the elements to its right side. If the picked element is greater than all the elements to its right side, then the picked element is the leader.

```
void printLeaders(int arr[], int size)
{
   for (int i = 0; i < size; i++)
   {
      int j;
      for (j = i+1; j < size; j++)
      {
         if (arr[i] <= arr[j])
            break;
      }
      if (j == size) // the loop didn't break
         cout << arr[i] << " ";
  }
}
```

Method 2 (Scan from right)

Scan all the elements from right to left in array and keep track of maximum till now. When maximum changes it’s value, print it.

```
void printLeaders(int arr[], int size)
{
   int max_from_right =  arr[size-1];
```

```
    /* Rightmost element is always leader */
    cout << max_from_right << " ";

    for (int i = size-2; i >= 0; i--)
    {
        if (max_from_right < arr[i])
        {
            max_from_right = arr[i];
            cout << max_from_right << " ";
        }
    }
}
```

3. Write a program in any language to find the number of inversion.

Inversion Count for an array indicates – how far (or close) the array is from being sorted. If array is already sorted then inversion count is 0. If array is sorted in reverse order that inversion count is the maximum.
Formally speaking, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

Example:

The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

Use Bubble Sort, Insertion Sort and Selection sort for the same.

METHOD 1 (Simple)

For each element, count number of elements which are on right side of it and are smaller than it.

```
int getInvCount(int arr[], int n)
{
  int inv_count = 0;
  for (int i = 0; i < n - 1; i++)
    for (int j = i+1; j < n; j++)
      if (arr[i] > arr[j])
        inv_count++;

  return inv_count;
}
```

4. An Array of integers is given, both +ve and -ve. You need to find the two elements such that their sum is closest to zero. Write code in any programming language for the same.

Solution:

Algorithm :

```
hasArrayTwoCandidates (A[], ar_size, sum)
1) Sort the array in non-decreasing order.
2) Initialize two index variables to find the candidate
   elements in the sorted array.
      (a) Initialize first to the leftmost index: l = 0
```

```
    (b) Initialize second  the rightmost index:  r = ar_size-1
3) Loop while l < r.
    (a) If (A[l] + A[r] == sum)  then return 1
    (b) Else if( A[l] + A[r] <  sum )  then l++
    (c) Else r--
4) No candidates in whole array - return 0
```

5. Find the largest and second largest element with its index in a given array. The runtime complexity of this code should be linear.

Solution:

```
void print2largest(int arr[], int arr_size)
{
   int i, first, second;

   /* There should be atleast two elements */
   if (arr_size < 2)
   {
      printf(" Invalid Input ");
      return;
   }

   first = second = INT_MIN;
   for (i = 0; i < arr_size ; i ++)
   {
      /* If current element is smaller than first
        then update both first and second */
      if (arr[i] > first)
      {
         second = first;
         first = arr[i];
      }

      /* If arr[i] is in between first and
        second then update second  */
      else if (arr[i] > second && arr[i] != first)
         second = arr[i];
   }
   if (second == INT_MIN)
      printf("There is no second largest element\n");
   else
      printf("The second largest element is %dn", second);
}
```

6. Write a function to find if a given integer x appears more than n/2 times in a sorted array of n integers.

Basically, we need to write a function say is Majority() that takes an array (arr[]), array's size (n) and a number to be searched (x) as parameters and returns true if x is a majority element_(present more than n/2 times).

Examples:

Input: arr[] = {1, 2, 3, 3, 3, 3, 10}, x = 3

Output: True (x appears more than n/2 times in the given array)

Input: arr[] = {1, 1, 2, 4, 4, 4, 6, 6}, x = 4

Output: False (x doesn't appear more than n/2 times in the given array)

Input: arr[] = {1, 1, 1, 2, 2}, x = 1

Output: True (x appears more than n/2 times in the given array)

Solution:

```
bool Morenooftimes(int array[], int n, int x)
{
    int i;
    int final_index = n % 2 ? n / 2 : (n / 2 + 1);

    for (i = 0; i < final_index; i++)
    {
        /* check if x is presents more than n/2 times */
        if (array[i] == x && array[i + n / 2] == x)
            return 1;
    }
    return 0;
}
```

7. You are given an array of 0s and 1s in random order. Segregate 0s on left side and 1s on right side of the array. Traverse array only once.

}

Input array   = [0, 1, 0, 1, 0, 0, 1, 1, 1, 0]

Output array = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]

Solution:

```
void segregate0and1(int arr[], int n)
```

```
{
    int count = 0; // Counts the no of zeros in arr

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

    // Loop fills the arr with 0 until count
    for (int i = 0; i < count; i++)
        arr[i] = 0;

    // Loop fills remaining arr space with 1
    for (int i = count; i < n; i++)
        arr[i] = 1;
}
```

8. Given two sorted arrays, find their union and intersection.

Example:

Input : arr1[] = {1, 3, 4, 5, 7}

arr2[] = {2, 3, 5, 6}

Output : Union : {1, 2, 3, 4, 5, 6, 7}

Intersection : {3, 5}

Input : arr1[] = {2, 5, 6}

arr2[] = {4, 6, 8, 10}

Output : Union : {2, 4, 5, 6, 8, 10}

Intersection : {6}

Solution:

```
int printUnion(int arr1[], int arr2[], int m, int n)
{
  int i = 0, j = 0;
  while (i < m && j < n)
  {
    if (arr1[i] < arr2[j])


    else if (arr2[j] < arr1[i])

    else
    {

      i++;
    }
  }

  /* Print remaining elements of the larger array */
  while(i < m)


  while(j < n)
}

int printIntersection(int arr1[], int arr2[], int m, int n)
{
  int i = 0, j = 0;
  while (i < m && j < n)
  {
    if (arr1[i] < arr2[j])
      i++;
    else if (arr2[j] < arr1[i])
      j++;
    else /* if arr1[i] == arr2[j] */
    {
      cout << arr2[j] << " ";
      i++;
      j++;
    }
  }}
```

9. A sorted array has n elements, how many searches M are going to be made for Binary search, when

   n= 1000, M = 10
   n = 16,000, M = 14
   n = 64000, M =16
   n = 1000000, M = 20

n = 900, M =10
n = 5000, M = 13