Lab Exercise on Brute Force and Divide and Conquer Techniques

1. Let $P_1, P_2, P_3, ..., P_n$ be points on the two dimensional plane. Write a C program to find the closet pair points (the distance between the points is minimum) . The distance the between the two points P_i and P_j is $d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.
2. Let S be a set of all rectangles. Assume that a rectangle is represented by its height and width. Write a C program to sort the rectangles in non-decreasing order of their areas.
3. The binomial coefficient $n_{C_k} = \frac{n!}{(n-k)!k!}$ is evaluated by the following recurrence relation

$$n_{C_k} = \begin{cases} 1 & \text{if } n = k \text{ or } k = 0 \\ (n-1)_{C_k} + (n-1)_{C_{(k-1)}} \end{cases} .$$

Write a recursive C program to evaluate n_{C_k} using divide-and-conquer technique .

4. You are given an array of n elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. Write a C program to remove all duplicates from the array in time O(nlogn) (Divide and Conquer).

Merge Sort :

```
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
#include<stdio.h>
void merge(int arr[], int p, int q, int r)
{
   int i, j, k;
   int n1 = q - p + 1;
   int n2 =  r - q;

   /* create temp arrays */
   int L[n1], R[n2];
```

```
/* Copy data to temp arrays L[] and R[] */
for (i = 0; i < n1; i++)
   L[i] = arr[p + i];
for (j = 0; j < n2; j++)
   R[j] = arr[q + 1+ j];

/* Merge the temp arrays back into arr[l..r]*/
i = 0; // Initial index of first subarray
j = 0; // Initial index of second subarray
k = p; // Initial index of merged subarray
while (i < n1 && j < n2)
{
   if (L[i] <= R[j])
   {
      arr[k] = L[i];
      i++;
   }
   else
   {
      arr[k] = R[j];
      j++;
   }
   k++;
}

/* Copy the remaining elements of L[], if there
  are any */
while (i < n1)
{
   arr[k] = L[i];
   i++;
   k++;
}

/* Copy the remaining elements of R[], if there
  are any */
while (j < n2)
```

```
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(int arr[], int p, int r)
{
    if (p < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int q = (r+p)/2;

        // Sort first and second halves
        mergeSort(arr, p, q);
        mergeSort(arr, q+1, r);

        merge(arr, p, q, r);
    }
}
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr)/sizeof(arr[0]);
     printf(" \n Array size =%d\n", arr_size);
    printf("Given array is \n");
```

```
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```