Lab Session:

Date of the Session: __/__/__ **Time of the Session:** _____to_____

Pre-lab:

1. Mention the other name of Quick sort algorithm. Briefly mention the procedure for Quick sort algorithm.

a. Quick sort is also known as partition-exchange sort. Partitions the array around a pivot and recursively sorts the subarrays

2. What do you understand about partitioning in context of the quick sort algorithm.

a. Partitioning in quicksort splits the array int two parts left with smaller elements, right with larger ones, around a pivot.

3. Mention the significance of pivot element in quick sort. Can any element be considered as pivot element in the array?

a. The pivot divides the array into smaller and larger parts. Any element can be pivot.

4. Write down the time complexity (all cases) of quick and merge sort.

Quick sort:
Best case: O(nlogn) worstcase: O(n²)
Avg case: O(n log n)

Merge sort:
Best case: O(n log n) worstcase
Avg case: O(n log n) O(n log

5. Suggest a sorting technique for the below given scenario:
You have 8 GB data that is to be sorted. Your RAM capacity is only 1 GB. So its evident that the capacity of RAM is insufficient to sort the entire 8GB data. Justify your answer with a proper sorting technique

a. Use external merge sort to handle 8GB data with 1GB RAM. It divides data into smaller chunks, sorts them in RAM and merges them efficiently using disk storage

In-Lab

SORT AND FIND SECOND MAXIMUM

1. Implement Quick to sort an integer array in ascending order, and try to make it as fast as possible. and print the Second maximum element in the array Comment Your Name And Registration Number As The First Line In The Code You are prohibited form using the built in functions of any language Your Code will be Reviewed to prevent any cheating.

   LINK: https://www.hackerrank.com/contests/quickly-sort-and-print-max-element/challenges

Program:

```
#include <stdio.h>
void swap(int*a, int*b) {
    int temp=*a;
    *a=*b;
    *b=temp;
}
int partition(int arr[], int low, int 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]);
  return i+1;
}
```

```
Void quicksort(int arr[], int low, int high) {
    if(low < high){
      int pi = partition(arr, low, high);
      quicksort (arr, low, pi-1);
      quicksort(arr, pi+1, high);
  } }
int findSecondMax(int arr[], int n) {
  int max = -1, secondMax = -1;
  For(int i=0; i<n; i++) {
    if(arr[i] > max) {
      secondMax = max;
        max = arr[i];
    }else if(arr[i] > secondMax && arr[i] != max) {
      Second Max = arr[i];
  }
  return secondMax;
}
int main(){
  int n;
scanf("%d", &n);
  int arr[n];
for(int i=0; i<n; i++){
 scanf("%d", &arr[i]);
}
quicksort(arr, 0, n-1);
int secondMax = FindSecondMax(arr, n);
Printf("%d\n", secondMax);
  return 0;
}
```

Sample Input and Output:

Input:	Output:
6 1 9 3 5 4 7	7

2. **Merge Sorted Array**

You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

LINK: https://leetcode.com/problems/merge-sorted-array/description/?envType=problem-list-v2&envId=sorting

Program:

```
void merge(int* nums1, int nums1Size, int m, int* nums2, int nums2Size,
           int n) {
int i=m-1;
int j=n-1;
while(i>=0 && j>=0){
if(nums1[i]>nums2[j]) {
    nums1[i+j+1]=nums1[i];
     i--;
}else{
nums1[i+j+1]=nums2[j];
 j--;
}
}
while(j>=0){
nums1[i+j+1]=nums2[j];
 j--;
}
}
```

Sample Input and Output

Input:

nums1 = [1,2,3,0,0,0]

m = 3

nums2 = [2,5,6]

n = 3

Output:

[1,2,2,3,5,6]

3. **Quicksort 1 – Partition**

The previous challenges covered Insertion Sort, which is a simple and intuitive sorting algorithm with a running time of $O(n^2)$.
In these next few challenges, we're covering a *divide-and-conquer* algorithm called Quicksort (also known as *Partition Sort*). This challenge is a modified version of the algorithm that only addresses partitioning. It is implemented as follows:

LINK: https://www.hackerrank.com/challenges/quicksort1/problem

Program:

```
#include<stdio.h>
void quickSort(int arr[],int n,int pivot){
 int left[n], mid[n], right[n];
  int leftIndex =0 , midIndex =0, rightIndex =0;
  for(int i=0;i<n;i++)
  if(arr[i] <pivot){
   left[leftIndex++]=arr[i];
  }else if(arr[i]==pivot){
    mid[midIndex++]=arr[i];
  }else{
    right[rightIndex++]=arr[i];
  }
}
   int index=0;
   for(int i=0;i<leftIndex;i++){
      arr[index++]=left[i];
  }
   for(int i=0;i<midIndex;i++){
      arr[index++]=mid[i];
  }
```

```
    for(int i=0; i<rightIndex; i++){
        arr[index++] = right[i];
    }
}
int main(){
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++){
    scanf("%d", &arr[i]);
    }
    int pivot =arr[0];
    quickSort(arr,n,pivot);
    for(int i=0;i<n;i++){
    printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Sample Input and Output:

Input:

1. 5
2. 4 5 3 7 2

Output:

3 2 4 5 7

Post-lab:

1. You are tasked with sorting an array of integers using Quick Sort. However, the catch is that you must implement a custom partition strategy to improve the sorting process. Instead of simply picking a pivot and partitioning around it in the standard manner (with elements less than the pivot on the left and greater on the right), you are required to partition the array into three segments:

   Low segment: All elements less than the pivot.
   Pivot segment: All elements equal to the pivot.
   High segment: All elements greater than the pivot.

   Your goal is to modify the partition step of Quick Sort so that the array is divided into three regions — the low, pivot, and high segments — rather than just low and high. Once the array is partitioned into these segments, Quick Sort should recursively sort the "low" and "high" segments, while the "pivot" segment will already be correctly placed.

Program:

```
#include <stdio.h>

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void threeWayPartition(int arr[], int low, int high) {
    int pivot = arr[low];
    int lt = low;
    int gt = high;
    int i = low;
    while (i <= gt) {
        if (arr[i] < pivot) {
            swap(&arr[lt], &arr[i]);
            lt++;
            i++;
```

```
    }else if (arr[i] > pivot){
      swap(&arr[i], &arr[gt]);
        gt--;
      }else{
        i++;
    }}
    for(int j=low; j<=high; j++){
      Printf("%d ", arr[j]);
    }
    Printf("\n");
    if(low < lt-1){
      threeWayPartition(arr, low, lt-1);
    }
    if(gt+1 < high){
      threeWayPartition(arr, gt+1, high);
  }
}
  int main(){
int n;
scanf("%d", &n);
int arr[n];
for(int i=0; i<n; i++){
  scanf("%d", &arr[i]);
}
  threeWayPartition(arr, 0, n-1);
    return 0;
}
```

Sample Input and Output:

Input:

```
7
5 8 1 3 7 9 2
```

Output:

```
1 2 3 5 7 8 9
```

2. Given the following array and pivot element, **perform the partitioning step of the Quick Sort** algorithm and write the correct configuration of the array after partitioning around the pivot element.

   Array: [9,3,8,4,7,2,6,1,5] , **Pivot** : 9

Program:

```
#include<stdio.h>
void qsort(int[], int, int);
 int main()
{ int n;
  Printf("Enter size");
  Scanf("%d", &n);
   int a[n], i;
 Printf("\n Enter elements:");
 for(i=0;i<n;i++)
{
  Scanf("%d", &a[i]);
}
Printf("Elements before sorting:");
for(i=0;i<n;i++)
{
 Printf("%d\t", a[i]);
 }
 qsort(a,0,n-1);
 Printf("After sort elements are:");
  for(i=0;i<n;i++)
   {
```

```
Printf("%d\t", a[i]);
}
return 0;
}
void qsort(int a[], int l, int h)
{
 int i, j, P, t;
 if (l<h) {
 P=l;
 i=l;
 j=h;
 while (i<j) {
 while (a[i]<=a[P])
  i++;
while (a[j]>a[P])
    j--;
if(i<=j) {
 t=a[i];
 a[i]=a[j];
 a[j]=t;
}
}
```

```
t=a[j];
a[j]=a[P];
 a[P]=t;
qsort(a,l,j-1);
qsort(a,j+1,h);
}
}
```

Sample Input and Output:

Input:

17 12 11 5 2 20 1 23 26

Output:

1 2 5 11 12 17 20 23 26

Skill Session:

1. **Merge Sort**

   A Student named Ramu have given an array of elements to sort in ascending order by her teacher Lalitha. Ramu is too lazy to sort the elements so he divided the array into two parts and given to two of his friends. His two friends are also lazy to sort So, each of them has divided the array into two parts and given to two of their friends. The process continues until each boy gets one element and they finally merge all the elements. Now your task is to find which person gets which elements.

   LINK: https://www.codechef.com/problems/MESO

   Program:

```
#include<stdio.h>
 struct array
{
 long int Fir;
 long int sec;
};
void merge(long int arr[], long int l, long int m, long int r) {
 long int i,j,k;
 long int n1=m-l+1;
 long int n2=r-m;
 long int L[n1],R[n2];
 for(i=0; i<n1;i++)
 L[i]=arr[l+i];
 for(j=0;j<n2;j++)
 R[j]=arr[m+1+j];
 i=0,j=0,k=l;
 while(i<n1&&j<n2) {
```

```
if(L[i] <= R[i]){
arr[k] = L[i];
  i++;
}else {
arr[k] = R[i];
  j++;
}
k++;
}
while(i<n1)
 {
  arr[k]= L[i];
   i++;
  k++;
 }
 while(j<n2) {
  arr[k]= R[j];
    j++;
    k++;
  }
  }
  void mergesort(long int arr[], long int l, long int r) {
   if(l<r) {
```

```
int m = l+(r-l)/2;
mergesort(arr, l, m);
mergesort(arr, m+1, r);
merge(arr, l, m, r);
}
}
int main() {
long int i,j,k,l,m,n,u,v,r,in;
for(v=0;v<=(2^i-2);v++){
Printf("%ld person is given with these numbers
for(long int p=a[v].first;p<a[v].sec;p++){
 Printf("%ld", b[p]);
}
printf("\n");
}
```

Sample Input and output:

Input:

7 7 6 5 4 3 2 1

Output:

1 person is given with 7 6 5 4 3 2 1

:

13 person is given with 2

After sorting elements ar

56 1 2 3 4 5 6 7

2. **Sorting tool**

Chaitya is a great problem solver. He found that most of his problems could be solved by arranging them in a particular order. The problem input consists of N. numbers, smaller than or equal to M. Chaitya's analysis consists of sorting this sequence according to their occurrences.

The output must be sorted so that for two numbers A and B, A will appear before B if the number of times A appears in the original order is larger than the number of times B does. If the number of occurrences is equal, the number whose value appears sooner in the input should appear sooner in the sorted sequence.

Help Chaitya by creating a custom sorting tool.

Link: https://www.codechef.com/problems/KJCP01

Program:

```
#include <stdio.h>
#include <stdlib.h>
void FreqSorting(int a[], int N, int M){
 int freq[M+1], firstOcc[M+1];
 for(int i=0; i<=M; i++){
    freq[i]=0;
    firstOcc[i] = -1;
}
 for(int i=0; i<N; i++){
   freq[a[i]]++;
   if(firstOcc[a[i]]==-1){
     firstOcc[a[i]]=i;
}}
```

```
for(int i=0;i<N-1;i++){
for(int j=i+1;j<N;j++){
if(freq[a[i]]<freq[a[j]] || (freq[a[i]]==freq[a[j]] && firstOcc[a[i]]
    firstOcc[a[j]])){
    int temp=a[i];
    a[i]=a[j];
    a[j]=temp;
}}}
for(int i=0;i<N;i++){
Printf("%d ",a[i]);
}}
int main()
{
int N,M;
Scanf("%d %d",&N,&M);
int a[N];
for(int i=0;i<N;i++)
Scanf("%d",&a[i]);
Freq Sorting(a,N,M);
return 0;
}
```

Sample Input and Output:

Input:	Output:
4 2 2 1 2 2	2 2 2 1.

3. Sort the People
   - You are given an array of strings names, and an array heights that consists of **distinct** positive integers. Both arrays are of length n.
   - For each index i, names[i] and heights[i] denote the name and height of the i[th] person.
   - Return names *sorted in* ***descending*** *order by the people's heights.*

   LINK: https://leetcode.com/problems/sort-the-people/description/

Program:

```
int compare(const void *a, const void *b) {
  return (*(int*) b - *(int *) a);
}
char** sortPeople (char** names, int namesSize, int *heights, int heightsSize,
int* returnSize){
char** result=(char**)malloc(namesSize* sizeof(char*));
 int * sortedHeights =(int*)malloc (heightsSize* sizeof(int));
 memcpy(sortedHeights, heights, heightsSize * sizeof(int));
 qsort(sortedHeights, heightsSize, sizeof(int), compare);
  for(int i=0; i<namesSize; i++) {
   for(int j=0; j< heightsSize; j++) {
    if (heights[j] == sortedHeights[i]){
      result[i] = names[j];
        break;
}  }
}
```

```
free(sortedHeights);
*returnSize = namesSize;
return result;
}
```

Sample Input and Output:

Input:

names = ["Mary", "John", "Emma"]

heights = [180, 165, 170]

Output:

["Mary", "Emma", "John"]

4. **Quicksort 2 – Sorting**

In the previous challenge, you wrote a *partition* method to split an array into two sub-arrays, one containing smaller elements and one containing larger elements than a given number. This means you 'sorted' half the array with respect to the other half. Can you repeatedly use *partition* to sort an entire array?

Guideline

In Insertion Sort, you simply went through each element in order and inserted it into a sorted sub-array. In this challenge, you cannot focus on one element at a time, but instead must deal with whole sub-arrays, with a strategy known as "divide and conquer".

When *partition* is called on an array, two parts of the array get 'sorted' with respect to each other. If *partition* is then called on each sub-array, the array will now be split into four parts. This process can be repeated until the sub-arrays are small. Notice that when partition is called on just one of the numbers, they end up being sorted.

Can you repeatedly call partition so that the entire array ends up sorted?

LINK: https://www.hackerrank.com/challenges/quicksort2/problem

Program:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
# include <stdlib.h>
#include <assert.h>
void quickSort(int *ar, int size) {
  if(size>1){
int temp, pivot=ar[0], ub=size-1;
for(int i= size-1; i>=0; --i){
```

```
if (ar[i] >= pivot) {
temp = ar[i];
  int j = i;
while (j < ub) {
ar[i] = ar[j+1];
  ++j;
ar[ub--] = temp;
}
}
++ub;
quickSort(ar, ub);
quick sort(ar+ub+1, size-ub-1);
 for(int i=0; i<size; ++i) {
Printf("%d", ar[i]);
 }
 Printf("\n");
}
int main(void) {
```

```
int N;
Scanf("%d", &N);
  int ar[N];
for(int i=0; i<N; i++) {
Scanf("%d", &ar[i]);
 }
quickSort(ar, N);
return 0;
}
```

Sample Input and Output:

5. **ARRAY SORT-MERGE**

Merge Sort algorithm

Implement Merge algorithm to sort an integer array in ascending order, and try to make it as fast as possible.

- Comment Your Name and Registration Number as The First Line In The Code
- You are prohibited form using the built in functions of any language
- Your Code will be Reviewed to prevent any cheating

Input Format

- First line:the size of the array
- Second line:the array to be sorted(numbers seperated by spaces)

LINK: https://www.hackerrank.com/data-strucutres-merge-sort

Program:

```
void merge(int arr[], int left, int mid, int right){
 int i,j,k;
int n1=mid-left+1;
 int n2=right-mid;
int L[n1], R[n2];
for(i=0;i<n1;i++)
 L[i]=arr[left+i];
for(j=0;j<n2;j++)
R[j]= arr[mid+1+j];
 i=0;
 j=0;
 k=left;
while(i<n1 && j<n2){
 if(L[i]<=R[j]){
```

```
    arr[k] = L[i];
    i++;
  } else {
    arr[k] = R[j];
    j++;
  }
  k++;
}
while (i < n1) {
  arr[k] = L[i];
  i++;
  k++;
}
}
void mergeSort(int arr[], int left, int right) {
  if (left < right) {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    merge(arr, left, mid, right);
  }
}
void printArray(int arr[], int size) {
  Printf("[");
  for(int i = 0; i < size; i++) {
    Printf("%d", arr[i]);
    if (i < size - 1) {
      Printf(",");
    }
  }
  Printf["]\n");
}
int main() {
  int n;
  scanf("%d", &n);
  int arr[n];
  for(int i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
  }
  mergeSort(arr, 0, n - 1);
  printArray(arr, n);
  return 0;
}
```

Sample Input and Output:

Input:	Output:
• 6 • 1 9 3 5 4 7	[1, 3, 4, 5, 7, 9]

6.SORT-MERGE

Given a list of N array elements apply Merge sort.
*Note: MergeSort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

Input Format

- The first line contains an integer, N, the number of elements in Array.
- The second line contains N space-separated integers.

LINK: https://www.hackerrank.com/data-strucutres-merge-sort

Program:

```
#include<stdio.h>
void merge(int arr[], int left, int mid, int right){
  int n1=mid-left+1;
  int n2=right-mid;
  int L[n1], R[n2];
  for(int i=0;i<n1;i++)
  L[i]=arr[left+i];
  for(int j=0;j<n2;j++)
  R[j]=arr[mid+1+j];
  int i=0, j=0, k=left;
while(i<n1 && j<n2){
if(L[i]<=R[j]){
```

```
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = R[j];
        j++;
    }
    k++;
}
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
}
for (int m = left; m <= right; m++) {
    printf("%d ", arr[m]);
}
printf("\n");
}
```

```
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid+1, right);
        merge(arr, left, mid, right);
    }
}
int main() {
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    mergeSort(arr, 0, n-1);
    return 0;
}
```

Sample Input and Output:

Input:

```
10
10 1 9 2 8 3 4 7 5 6
```

output

```
1 10
1 9 10
2 8
1 2 8 9 10
3 4
3 4 7
5 6
3 4 5 6 7
1 2 3 4 5 6 7 8 9 10
```