Week 1
Practice lab – Algorithms and Problem Solving (15B17CI471)

We are given an array of n distinct numbers; where n is large numbers are randomly generated. The task is to

a) Sort the entire array using selection sort, bubble sort, insertion sort, quick sort and merge sort. Print the total number of comparisons done in each of the sorting algorithm.

b) Take sorted array from part a) and again run all the sorting algorithm functions. Print the total number of comparisons done in each of the sorting algorithm.

c) Change the functions to take a flag "order" as argument. This order can be „d" or „a" for descending and ascending respectively. The function will sort the array in descending and ascending order depending on the flag value.

d) Take sorted array from part a). Again run all the sorting algorithm functions with the "order" flag changed. If array is already in ascending order pass flag value as „d" and vice versa. Print the total number of comparisons done in each of the sorting algorithm.

e) Implement quick sort with three overloaded functions. 1st taking pivot as first index, 2nd taking pivot as last index and 3rd taking pivot as middle index.

f) Analyse complexity of quick sort and write down your observation of best and worst case

g) Analyse complexity of merge sort and write down your observation of best and worst case

h) Analyse complexity of bubble sort and write down your observation of best and worst case

i) Analyse complexity of selection sort and write down your observation of best and worst case

j) Analyse complexity of insertion sort and write down your observation of best and worst case

k) Write a function to sort all even-placed numbers in increasing and odd-place numbers in decreasing order. The modified array should contain all sorted even-placed numbers followed by reverse sorted odd-placed numbers. Analyse the complexity of your implemented approach

Note that the first element is considered as even because of its index 0.

Example for part k):

```
Input: arr[] = {0, 1, 2, 3, 4, 5, 6, 7}
Output: arr[] = {0, 2, 4, 6, 7, 5, 3, 1}
Even-place elements : 0, 2, 4, 6
Odd-place elements : 1, 3, 5, 7
Even-place elements in increasing order :
0, 2, 4, 6
Odd-Place elements in decreasing order :
7, 5, 3, 1
Input: arr[] = {3, 1, 2, 4, 5, 9, 13, 14, 12}
Output: 1, 2, 4, 6, 7, 5, 3, 1
Even-place elements : 3, 2, 5, 13, 12
Odd-place elements : 1, 4, 9, 14
Even-place elements in increasing order : 2, 3, 5, 12, 13
Odd-Place elements in decreasing order :
14, 9, 4, 1
```

l) Given an integer array of which both first half and second half are sorted. Task is to merge two sorted halves of array into single sorted array. Analyse the complexity of your implemented approach

Example:
```
Input : A[] = { 2 ,3 , 8 ,-1 ,7 ,10 }
Output : -1 , 2 , 3 , 7 , 8 , 10
Input : A[] = {-4 , 6, 9 , -1 , 3 }
Output : -4 , -1 , 3 , 6 , 9
```

```cpp
#include <iostream>
#include <vector>
using namespace std;
int comparison1 = 0;
void merge(vector<int> &p, int l, int mid, int h)
{
    int i, j, k;
    i = l;
    j = mid + 1;
    vector<int> b;
    while (i <= mid && j <= h)
    {
        if (p[i] < p[j])
        {
            b.push_back(p[i++]);
        }
        else
        {
            b.push_back(p[j++]);
        }
    }
    for (; i <= mid; i++)
    {
        b.push_back(p[i]);
    }
    for (; j <= h; j++)
    {
        b.push_back(p[j]);
    }
    for (i = l; i <= h; i++)
    {
        p[i] = b[i];
    }
}
void mergesort(vector<int> &p, int l, int h)
{
    int mid;
    if (l < h)
    {
        mid = l + (h - l) / 2;
        mergesort(p, l, mid);
        mergesort(p, mid + 1, h);
        merge(p, l, mid, h);
    }
}
// insertion sort
void insertion_sort(vector<int> &p, int n)
{
    int comp = 0;
    for (int i = 1; i < n; i++)
    {
        int j = i - 1;
        int x = p[i];
        comp++;
        while (j > -1 && p[j] > x)
        {
            comp++;
            p[j + 1] = p[j];
            j--;
        }
        p[j + 1] = x;
    }
    cout << "Sorted array: ";
    for (int i = 0; i < p.size(); i++)
    {
        cout << p[i] << " ";
    }
    cout << "\nNo. of comparisons: " << comp;
}
int partition(int low, int high, vector<int> &p)
{

    int pivot = p[low];
    int i = low;
    int j = high;
    do
    {
        do
        {
            i++;
            comparison1++;
        } while (p[i] <= p[pivot]);
        do
        {
            j--;
            comparison1++;
        } while (p[j] > pivot);
        if (i < j)
        {
            int temp = p[i];
            p[i] = p[j];
            p[j] = temp;
        }
    } while (i < j);
    int temp = pivot;
    pivot = p[j];
    p[j] = temp;
    return j;
}
void quicksort(int low, int high, vector<int> &p)
{
    int mid;
    if (low < high)
    {
        mid = partition(low, high, p);
        quicksort(low, mid, p);
        quicksort(mid + 1, high, p);
    }
}
void bubble(vector<int> p)
{
    int comparison = 0;
    int len = p.size();
    int flag = 0;
    for (int i = 0; i < len; i++)

    {
        int temp = p[i];
        for (int j = 0; j < len - 1; j++)
        {
            flag = 0;
            if (p[j] > p[j + 1])
            {
                comparison++;
                int temp = p[j];
                p[i] = p[j];
                p[j] = temp;
                flag = 1;
            }
            else
            {
                comparison++;
            }
            if (flag == 0)
            {
                break;
            }
        }
    }
    cout << "Bubble Sorted array is: ";
    for (int i = 0; i < len; i++)
    {
        cout << p[i] << " ";
    }
    cout << "No. of comparison: " << comparison;
}
void selection_sort(vector<int> p)
{
    int comparison = 0;
    int len = p.size();
    for (int i = 0; i < len; i++)
    {
        int temp = p[i];

        for (int j = i + 1; j < len; j++)
        {
            if (p[j] < p[i])
            {
                comparison++;
                int temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
            else
            {
                comparison++;
            }
        }
    }
    cout << "Selection Sorted array is: ";
    for (int i = 0; i < len; i++)
    {
        cout << p[i] << " ";
    }
    cout << "No. of comparison: " << comparison;
}
int main()
{
    vector<int> arr;
    arr.push_back(2);
    arr.push_back(4);
    arr.push_back(6);
    arr.push_back(8);
    arr.push_back(7);
    selection_sort(arr);
    cout << endl;
    bubble(arr);
    return 0;
}
```

COMPLEXITY OF QUICKSORT
For quicksort, worst case time, O(n^2)
Average case time=O(nlogn)
Best case time, O(nlogn) —When partitioning is done in the middle.

COMPLEXITY OF MERGE SORT
Best Case Time: O(nlogn)
Worst Case Time: O(nlogn)
Average Case Time: O(nlogn)
Because the mergesort always divides the array into two halves and takes linear time to merge two halves.

COMPLEXITY OF BUBBLE SORT
Best Case TIme: O(n) *When implementing the optimized approach using flag
Average Case TIme: O(n2)
Worst Case TIme: O(n2)

COMPLEXITY OF SELECTION SORT
Best Case TIme: O(n2)
Average Case TIme: O(n2)

Worst Case TIme: O(n2)

COMPLEXITY OF INSERTION SORT
Best Case TIme: O(n) *When List is sorted
Average Case TIme: O(n2)
Worst Case TIme: O(n2)

K.

```
#include <bits/stdc++.h>
using namespace std;
vector<int> sorts(vector<int> &p)
{
    vector<int> even;
    vector<int> odd;
    for (int i = 0; i < p.size(); i = i + 2)
    {
        even.push_back(p[i]);
    }
    for (int i = 1; i < p.size(); i = i + 2)
    {
        odd.push_back(p[i]);
    }
    sort(even.begin(), even.end());
    sort(odd.begin(), odd.end(), greater<int>());
    even.insert(even.end(), odd.begin(), odd.end());
    return even;
}
int main()
{
    vector<int> vec = {0, 1, 2, 3, 4, 5, 6, 7};
    vec = sorts(vec);
    cout << "Result is: ";
    for (int i = 0; i < vec.size(); i++)
    {
        cout << vec[i] << " ";
    }
    return 0;
}
```

As we have created two separate arrays and inserted the even and odd place values in those vectors. Both take O(n) time. The sort function uses mergesort therefore O(n logn) time.
.The insert function takes O(n) time.
In total, O(n+n+nlogn+n)=O(3n+nlogn).

```
#include <bits/stdc++.h>
using namespace std;
vector<int> merge(vector<int> &p)
{
    int i = 0;
    int mid = (p.size() - 1) / 2;
    int j = mid + 1;
    int h = p.size() - 1;
    vector<int> b;
    while (i <= mid && j <= h)
    {
        if (p[i] < p[j])
        {
            b.push_back(p[i++]);
        }
        else
        {
            b.push_back(p[j++]);
        }
    }
    for (; i <= mid; i++)
    {
        b.push_back(p[i]);
    }
    for (; j <= h; j++)
    {
        b.push_back(p[j]);
    }
    return b;
}
int main()
{
    vector<int> vec = {-4, 6, 9, -1, 3};
    vec = merge(vec);
    cout << "Final result: ";
    for (int i = 0; i < vec.size(); i++)
    {
        cout << vec[i] << " ";
    }
    return 0;
}
```

Since, we are using the merge function of mergesort therefore time complexity is of order of O(n).