Lab Manual – 03

Course Title: Algorithm Lab
Course Code: CSE 223
Semester: Spring 2020
Topic: Sorting Algorithm
(Selection Sort, Merge Sort, Quick Sort)

::::Course Teacher::::
Masud Rabbani
Lecturer
Dept. of CSE, DIU

::::Prefect::::

Muhaiminul Islam
Student of DIU
Id: 163-15-8473
Mail: muhaiminul15-8473@diu.edu.bd

Muhiyminul Islam
Student of DIU
Id: 172-15-10187
Mail: muhiyminul15-8473@diu.edu.bd

Hasan Imam Bijoy
Student of DIU
Id: 182-15-11743
Mail: hasan15-11743@diu.edu.bd

Md Mahbubur Rahman
Student of DIU
Id: 182-15-11742
Mail: mahbubur15-11742@diu.edu.bd

Selection Sort

Introduction: In computer science, selection sort is a sorting algorithm, specifically an in-place comparison sort. It has O(n2) time complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and it has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

Applications:

- (Real world application) Let's consider you have 8 shoes pairs with sizes 6 to 13. You find them not in order. Their arrangement from smallest to largest will be an example of selection sort How it works

- If small list needs to be sorted
- The entire cost of swapping does not matter
- checking of all the elements is compulsory
- cost of writing to a memory matters like in flash memory (number of writes/swaps is O(n) as compared to O(n2) of bubble sort).

Complexity:

- Best case: $O(n^2)$
- Worst case: $O(n^2)$
- Average case: $O(n^2)$

Outcome:

It is very simple algorithm to implement. And may perform better than other completed sorting algorithms in various perspectives. So, the outcome is nothing but a sorted dataset using a very simple approach.

Advantages and disadvantages:

Advantages are –

- It is very simple and easy to understand.
- Its performance is very good in case of small array's or list's.

Disadvantages are:

- It is very simple and easy to understand.
- Similar to the bubble sort, the selection sort requires n-squared number of steps for sorting n elements.

Basic Code

```
#include <stdio.h>
int main()
{
   int a[100], n, i, j, position, swap, min_idx,temp;
   printf("Enter Number of Elements n :");
   scanf("%d", &n);
   printf("Enter %d Numbers n: \n", n);
   for (i = 0; i < n; i++)
   {
      scanf("%d", &a[i]);
   }
   for (i = 0; i < n-1; i++)
   {
      min_idx = i;
      for (j = i+1; j < n; j++)
      {
         if (a[j] < a[min_idx])
         {
            min_idx = j;
         }
      }
      temp = a[min_idx];
      a[min_idx] = a[i];
      a[i] = temp;
   }
   printf("\nSorted Array:\n");
   for(i = 0; i < n; i++)
   {
      printf("%d ", a[i]);
   }
   return 0;
}
```

Practice Problem:

1. Suppose, you have some of random data. Then you sorted the data and Find the power of second minimum value and largest value (Formula: x^y). Input contains (0<=1<=10).

Input: Ar[] = {1,4,3,2}	**Input:** Ar[] = {10,5,1,4,6,8}
Output: Sorted Data: 1 2 3 4 Result: $2^4 = 16$	**Output:** Sorted Data: 1 4 5 6 8 10 Result: $4^{10} = 1048576$

2. Suppose, you have some random data and total number of data is N and an integer t. Follow the above algorithm and print the state of the array after t times have been performed.

Input: Ar[] = {1,5,9,3,7,4} t = 2	**Input:** Ar[] = {10,5,12,15,20,8} t =3
Output: After 2 Times Array: 1 3 9 5 7 4	**Output:** After 3 Times Array: 5 8 10 15 20 12

Merge Sort

Introduction: In computer science, **merge sort** is an efficient, general-purpose, comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the order of equal elements is the same in the input and output. It is a divide and conquer algorithm that was invented by John von Neumann in 1945.A detailed description and analysis of bottom-up mergesort appeared in a report by Goldstine and von Neumann as early as 1948.

Applications:

- Merge Sort is useful for sorting linked lists in O(nLog(n)) time.
- Inversion count problem.
- Used in external sorting.

Complexity:

- Best case: O(n log n)
- Worst case: O(n log n)
- Average case: O(n log n)

Outcome:

Merge Sort is useful for **sorting** linked lists. **Merge Sort** is a stable **sort** which means that the same element in an array maintain their original positions with respect to each other. Overall time complexity of **Merge sort** is O(nLogn). So after the sorting we can get a sorted list/array with less time complexity .

Advantages and disadvantages:

Advantages are –

- It is quicker for larger lists because unlike insertion and bubble sort it doesnt go through the whole list several times.
- It has a consistent running time, carries out different bits with similar times in a stage.

Disadvantages are:

- Slower comparative to the other sort algorithms for smaller tasks.
- Goes through the whole process even the list is sorted (just like insertion and bubble sort)
- Uses more memory space to store the sub elements of the initial split list.

Basic Code

```
#include<stdlib.h>
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    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[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++;
    }
}
```

```
void mergeSort(int arr[], int l, 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);
    }
}
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
    {
        printf("%d ", A[i]);
    }
    printf("\n");
}
int main()
{
    int arr[1000], arr_size;
    printf("Enter Number of Elements in Array : ");
    scanf("%d", &arr_size);
    printf("Enter Elements of Array : \n");
    for(int i = 0; i < arr_size; i++)
    {
        scanf("%d", &arr[i]);
    }
    mergeSort(arr, 0, arr_size - 1);
    printf("\nSorted Array is : ");
    printArray(arr, arr_size);
    return 0;
}
```

Practice Problem:

1. Suppose, you have some of random data. Then your job is finding out the Median value and print it.

Input: Ar[]={1,5,9,3,7}	**Input:** Ar[]={21,32,20,31,27,25,29,70}
Output: Sort: 1 3 5 7 9 Median Value: 5	**Output:** Sort: 20 21 25 27 29 31 32 70 Median Value: 28.00

2. You have some of various data. Then those data print ascending order and calculate sum of data and also print the value of digit sum.

Input: Ar[]={1,5,9,3,7,4}	**Input:** Ar[]={21,20,32,27,45,29}
Output: Sort list: 1 3 5 7 9 Sum: 25 Digit Sum: 7	**Output:** Sort list: 20 21 27 29 32 45 Sum: 174 Digit Sum: 12

Quick Sort

Introduction: Developed by British computer scientist Tony Hoare in 1959 and published in 1961, and it is still a commonly used algorithm for **sorting**. When implemented well, it can be about two or three times faster than its main competitors, merge **sort** and heapsort. It is a divide-and-conquer algorithm.

Applications:

It is Very popular sequential sorting algorithm that performs well with an average sequential time complexity of O(nlogn). Well it's applications are similar with other sorting algorithms. such as:

- Commercial computing.
- Search for information.
- Operations research.
- Event-driven simulation.
- Numerical computations.

Complexity:

- Best case: $O(n^2)$
- Worst case: O(n log n)
- Average case: O(n log n)

Outcome:

It reduces the space complexity and removes the use of the auxiliary array that is used in merge sort. Selecting a random pivot in an array results in an improved time complexity in most of the cases. After performing it we can have a sorted list / array or dataset with less space complexity.

Advantages and disadvantages:

Advantages are –

- high performance, easy implementation, can easily combine with caching and internal memory mechanisms.

Disadvantages are:

- Unstable, heavily decreases in speed down to O(n2) in the case of unsuccessful pivot selections. Lame implementation of the algorithm may result in stack overflow error, since it may require O(n) embedded recursive calls.

Basic Code

```
#include <stdio.h>
int partitions(int A[], int low, int high)
{
    int pivot, i, j, t;
    pivot = A[high];
    for(i = low - 1, j = low; j < high; j++)
    {
        if(A[j] < pivot)
        {
            i += 1;
            t = A[i];
            A[i] = A[j];
            A[j] = t;
        }
    }
    t = A[i+1];
    A[i+1] = A[high];
    A[high] = t;
    return i+1;
}
```

```
void quick_sort(int A[], int low, int high)
{
    if(low >= high)
    {
        return;
    }
    int p;
    p = partitions(A,low,high);
    quick_sort(A,low,p-1);
    quick_sort(A,p+1,high);
}
int main ()
{
    int x, A[100];
    printf("Array length : ");
    scanf("%d", &x);
    printf("Input Array : \n");
    for(int i = 0; i < x; i++)
    {
        scanf("%d", &A[i]);
    }
    quick_sort(A,0,x-1);
    printf("The sorted Array : ");
    for(int i = 0; i < x; i++)
    {
        printf("%d ", A[i]);
    }
}
```

Practice Problem:

1. Suppose, you have some random data and total number of data is N. Then The task is to complete the partition () function which is used to implement Quick Sort.

Input: Test Case: 2 Ar[] = {1,5,9,3,7} Ar[] = {10,5,15,25,20}	**Input:** Test Case :3 Ar[] = {2,1,3,6,4,7} Ar[] = {8,2,9,3,7,4} Ar[] = {3,1,2,5,4,7}
Output: Sorted Data: 1 3 5 7 9 Sorted Data: 5 10 15 20 25	**Output:** Sort: 20 21 25 27 29 31 32 70 Median Value: 28.00

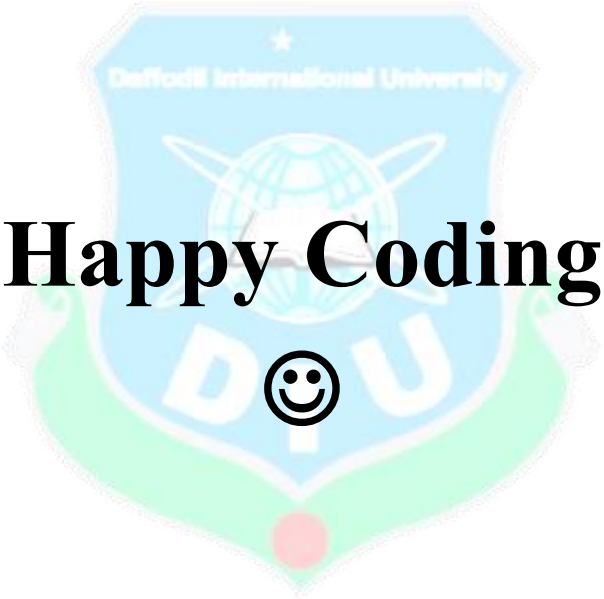
2. Suppose, you have some of random data. Your job is print the data in ascending order, then pick a value and find out how many numbers are small and large in an array.

Input: Ar[] = {1,5,9,3,7,4} Pick Value: 4	**Input:** Ar[] = {10,5,25,15,20,30} Pick Value: 15
Output: Ascending Order: 1 3 4 5 7 9 Large value: 3 Small value: 1	**Output:** Ascending Order: 5 10 15 20 25 30 Large value: 3 Small value: 2

Edited By

Md Atiqur Rahman

"Run for excellence, Success will definitely come"



'Happy Coding'

☺