St. Francis Institute of Technology, Mumbai-400 103
Department of Information Technology

A.Y. 2020-2021
Class: SE-ITA/B, Semester: III
Subject: DATA STRUCTURE LAB

Experiment – 6 Insertion sort and Quick sort algorithms

1. Aim: Write a menu driven program to implement Insertion sort, Quick sort.

2. Objectives: After study of this experiment, the student will be able to • To use basic principles of programming as applied to searching and sorting • To learn fundamentals of sorting techniques

3. Outcomes: After study of this experiment, the student will be able to • Implement sorting algorithms on given set of data and understand its operations • Understand the concepts and apply the techniques of searching, hashing and sorting

4. Prerequisite: Sorting techniques.

5. Requirements: PC and Turbo C compiler version 3.0

6. Pre-Experiment Exercise:
Brief Theory:

A. Sorting

A sorting algorithm is defined as an algorithm that puts elements of a list in a certain order (that can either be numerical order, lexicographical order or any user-defined order). Efficient sorting algorithms are widely used to optimize the use of other algorithms like search and merge algorithms which require sorted lists to work correctly. There are two types of sorting:

• Internal sorting which deals with sorting the data stored in computer's memory • External sorting which deals with sorting the data stored in files. External sorting is applied when there is voluminous data that cannot be stored in computer's memory.

B. Types of Sorting Techniques

1. Insertion Sort

• Idea: like sorting a hand of playing cards

• Start with an empty left hand and the cards facing down on the table. • Remove one card at a time from the table, and insert it into the correct position in the left hand

• compare it with each of the cards already in the hand, from right to left

• The cards held in the left hand are sorted, these cards were originally the

top cards of the pile on the table

```
Alg.: INSERTION-SORT(A)
  for j ← 2 to n
      do key ← A[ j ]
         Insert A[ j ] into the sorted sequence A[1 . . j -1]
         i ← j - 1
         while i > 0 and A[i] > key
            do A[i + 1] ← A[i]
               i ← i - 1
         A[i + 1] ← key
```

- Insertion sort – sorts the elements in place

2. Quicksort

Quicksort is based on the ***divide-and-conquer*** paradigm.
It works as follows:
1. First, it partitions an array into two parts,
2. Then, it sorts the parts independently,
3. Finally, it combines the sorted subsequences by a simple concatenation.

The quick-sort algorithm consists of the following three steps:
1. ***Divide***: Partition the list.
– To partition the list, we first choose some element from the list for which we hope about half the elements will come before and half after. Call this element the ***pivot***.
– Then we partition the elements so that all those with values less than the pivot come in one sublist and all those with greater values come in another.
2. ***Recursion***: Recursively sort the sublists separately.
3. ***Conquer***: Put the sorted sublists together.

7. Laboratory Exercise

A. Procedure

Write a C program to implement Insertion sort, Quick sort show all the following operations in switch case,

i) Enter values in array
ii) Insertion sort
iii) Quick sort
iv) Display array
v) Exit

```c
// C program to implement Insertion sort and Quick sort algorithms
#include <stdio.h>
#define MAX 10
int A[MAX];



// function declaration
```

```c
//Mention all functions declarations
void insertionSort(int arr[], int n);
void createArray(int size);
void quickSort(int arr[], int beg, int end);
int partition(int a[], int beg, int end);
void displayArray(int size);


// main function
int main()
{
    int option, size;
    //int n = size/sizeof(A[0]);
    do
    {
        printf("\n ******MAIN MENU******* \n");
        printf("\n 1. Enter values in array");
        printf("\n 2. Insertion sort ");
        printf("\n 3. Quick sort ");
        printf("\n 4. Display array");
        printf("\n 5. Exit ");
        printf("\n\n Enter your option : ");
        scanf("%d", &option);
        switch(option)
        {
            case 1: printf("\n Enter the array size : ");
                    scanf("%d", &size);
                    createArray(size);
                    break;
            case 2: insertionSort(A, size);
                    displayArray(size);
                    break;
            case 3: quickSort(A, 0, size-1);
                    displayArray(size);
                    break;
            case 4: displayArray(size);
                    break;
        }
    }while(option!=5);

    return 0;
}
```

```
void createArray(int size){
    if(size<=10){
        printf("\nEnter the elements of the array:");
        for (int i = 0; i < size; i++)
        {
            scanf("%d",&A[i]);
        }
    }
    else{
        printf("\nSize of array should be less than 10");
    }
}

void displayArray(int size){
    printf("\nDisplaying array:");
    for (int i = 0; i < size; i++)
    {
        printf("\n%d",A[i]);
    }

}

void insertionSort(int arr[], int n)
{
    int i, temp, j;
    for (i = 1; i < n; i++) {
        temp = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > temp) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = temp;
    }
}

void quickSort(int arr[], int beg, int end){
    int loc;
    if (beg<end)
    {
        loc=partition(arr,beg,end);
        quickSort(arr,beg,loc-1);
```

```
        quickSort(arr,loc+1,end);
    }

}
int partition(int a[], int beg, int end)
{
    int left, right, temp, loc, flag;
    loc = left = beg;
    right = end;
    flag = 0;
    while(flag != 1)
    {
        while((a[loc] <= a[right]) && (loc!=right))
            right--;
        if(loc==right)
            flag =1;
        else if(a[loc]>a[right])
        {
            temp = a[loc];
            a[loc] = a[right];
            a[right] = temp;
            loc = right;
        }
        if(flag!=1)
        {
            while((a[loc] >= a[left]) && (loc!=left))
                left++;
            if(loc==left)
                flag =1;
            else if(a[loc] <a[left])
            {
                temp = a[loc];
                a[loc] = a[left];
                a[left] = temp;
                loc = left;
            }
            }
        }
    return loc;
}
```

B. Result/Observation/Program code:

Observe the output for the above code and print it.

```
D:\College\DSA\Experiments\Exp6>Exp6

 ******MAIN MENU*******

 1. Enter values in array
 2. Insertion sort
 3. Quick sort
 4. Display array
 5. Exit

 Enter your option : 1

 Enter the array size : 5

Enter the elements of the array:123 66 1 29 7

 ******MAIN MENU*******

 1. Enter values in array
 2. Insertion sort
 3. Quick sort
 4. Display array
 5. Exit

 Enter your option : 2

Displaying array:
1
7
29
66
123
```

```
 ******MAIN MENU*******

 1. Enter values in array
 2. Insertion sort
 3. Quick sort
 4. Display array
 5. Exit

 Enter your option : 1

 Enter the array size : 10

Enter the elements of the array:66 71 22 99 10 544 190 1 8 100

 ******MAIN MENU*******

 1. Enter values in array
 2. Insertion sort
 3. Quick sort
 4. Display array
 5. Exit

 Enter your option : 3

Displaying array:
1
8
10
22
66
71
99
100
190
544
```

```
******MAIN MENU*******

1. Enter values in array
2. Insertion sort
3. Quick sort
4. Display array
5. Exit

Enter your option : 5

D:\College\DSA\Experiments\Exp6>
```

8. Post-Experiments Exercise

A. Questions:

1. Sort the following numbers using Insertion sort.
   12,56,97,2,4,86,15,94,23,48
2. Sort the following numbers using Quick sort.
   56,88,74,64,35,12,95,37,24
3. List and compare the running time various sorting techniques.

B. Conclusion:

1. Summary of Experiment
2. Importance of Experiment

C. References:

1. S. K Srivastava, Deepali Srivastava; Data Structures through C in Depth; BPB Publications; 2011.
2. Reema Thareja; Data Structures using C; Oxford.
3. Data Structures A Pseudocode Approach with C, Richard F. Gilberg & Behrouz A. Forouzan, second edition, CENGAGE Learning.

Yash Mahajan SE IT B 04 Page: Date: / / 191061

A) Questions :-

1) Sort the following numbers using insertion sort

12, 56, 97, 2, 4, 86, 15, 94, 23, 48

PASS 1

12	56	97	2	4	86	15	94	23	48

PASS 2

12	56	97	2	4	86	15	94	23	48

PASS 3

12	56	97	2	4	86	15	94	23	48

PASS 4

2	12	56	97	4	86	15	94	23	48

PASS 5

2	4	12	56	97	86	15	94	23	48

PASS 6

2	4	12	56	86	97	15	94	23	48

PASS 7

2	4	12	15	56	86	97	94	23	48

PASS 8

2	4	12	15	56	86	94	97	23	48

Yash Mahajan SE IT B 64 191061

PASS 9

2	4	12	15	23	56	86	94	97	48

Sorted ~~list~~ list

2	4	12	15	23	~~48~~	~~56~~	~~86~~	94	97

2. Sort the following numbers using quick sort

56, 88, 74, 64, 35, 12, 95, 37, 24

0	1	2	3	4	5	6	7	8
56	88	74	64	35	12	95	37	24

PASS 1:-

i) left = 0, Right = 8, loc = 0

A[loc] < A[right] → ~~True~~ False

~~right --~~

swap A[loc], A[right]

ii)

24	88	74	64	35	12	95	37	~~56~~

left = 0, right = 8, loc = 8

iii) A[loc] > A[left] → true

left ++

(iii)

24	88	74	64	35	12	95	37	56

left = 1, loc = 8, right = 8

A[loc] > A[left] → false

Swap A[loc], A[left]

loc = left

(iv)

24	56	74	64	35	12	95	37	88

left = 1, loc = 1, right = 8

A[loc] < A[right] → true

Right --

(v)

24	56	74	64	35	12	95	37	88

left = 1, loc = 1, right = 7

A[loc] < A[right] → false

Swap A[loc], A[right]

loc = right

(vi)

24	37	74	64	35	12	95	56	88

left = 1, loc = 7, right = 7

A[loc] > A[left] → true.

left ++

Page: Date: / /

(vii)

24	37	74	64	35	12	95	56	88

left = 2 , Right = 7, loc =>

A[loc] > A[left] → False

Swap A[loc], A[left]

loc = left.

(viii)

24	37	56	64	35	12	95	74	88

left = 2, right = 7, loc = 2

A[loc] < A[right] → True

right --

(ix)

24	37	56	64	35	12	95	74	88

left = 2, right = 6, loc = 2

A[loc] < A[right) → ~~False~~ True

~~Swap A[loc], A[right]~~

~~loc = right~~ right --

(x)

24	37	56	64	35	12	95	74	88

left = 2, right = 5, loc = 2

A[loc] < A[right] → False

Swap A[loc], A[right]

loc = right.

Yash Mahajan SE IT B 04 Page: Date: / / 191061

(xi)

24	37	12	64	35	56	95	74	88

left = 2, right = 5, loc = 5

A[loc] > A[left] → true

left ++

(xii)

24	37	12	64	35	56	95	74	88

left = 3, right = 5, loc = 5

A[loc] > A[left] → false

swap A[left], A[loc],

loc = left

(xiii)

24	37	12	56	35	64	95	74	88

left = 3, right = 5, loc = 3

A[loc] < A[right] → true, right --

24	37	12	56	35	64	95	74	88

left = 3, right = 4, loc = 3

A[loc] < A[right]

swap A[loc], A[right]

loc = Right

24	37	12	35	56	64	95	74	88

left = 3, right = 4, loc = 3

A[loc] > A[left]

left ++

left = 4, right = 4, Stop pass.

Yash Mahajan SE IT B 04 Page: Date: / / 191061

24	37	12	35	56	64	95	74	88

Partition 1 — Pivot — Partition 2

PASS 2

24	37	12	35

left = 0, right = 3, loc = 0

A[loc] < A[right] - True

right --

24	37	12	35

loc = 0, left = 0, right = 2

A[loc] < A[right] - false

swap A[loc], A[right]

loc = right

12	37	24	35

loc = 2, right = 2, left = 0

A[loc] > A[left] → true

left++

12	37	24	35

left = 1, right = 2, loc = 2

A[loc] > A[left] → false

swap A[loc], A[left]

loc = left

Yash Mahajan SE IT B 04 191061

Page: Date: / /

12	24	37	35

left = 1, loc = 1, right = 2

A[loc] < A[right] true
right --

loc = 1, right = left

Stop pass.

~~PASS~~

12	24	37	35

Partition1 Pivot Partition 2

PASS 3

37	35

loc = 0, left = 0, right = 1

A[loc] < A[right] -> false
swap A[loc], A[right]
loc = right

35	37

left = 0, loc = 1, right = 1
A[loc] > A[left] -> True
* left++

left = right , Stop pass

Pass 4

64	95	74	88

loc=0, left=0, right=3

A[loc] < A[right] -> true

right --

64	95	74	88

loc=0, left=0, right=2

A[loc] < A[right] - true

right --

64	95	74	88

loc=0, left=0, right=1

A[loc] < A[right] -> true

right --

64	95	74	88

loc=0, left=0, right=0

left=right, stop pass

64	95	74	88

↑ Pivot

Partition.

Yash Mahajan SE IT B 04 Page: Date: / / 191061

PASS 5

95	74	88

left=0, right=2, loc=0

A[loc] < A[right] -> false
swap A[loc], A[right]
loc = right

88	74	95

left=0, loc=2, right=2
A[loc] > A[left] -> true
left++

88	74	95

left=1, loc=2, right=2

A[loc] > A[left] -> true
left++

left=2, right=2, Stop pass

88	74	95

Partition Pivot

Yash Mahajan SE ITB 04 Page: Date: 191061

PASS 6 $(1+2)^n$

88	74

left = 0, right = 1, loc = 0

A[left] < A[right] -> false
Swap A[loc], A[right]
loc = right

74	88

loc = 1, right = 1, left = 0
A[loc] > A[left] -> true
left ++

left = right = 1, stop pass

12	24	35	37	56	64	74	88	95

3) List and compare, the running time of various sorting techniques.

Algorithm	Best case	Average case	Worst case
Bubble Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Selection Sort	$\Omega(N^2)$	$\Theta(N^2)$	$O(N^2)$
Insertion Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Merge Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N \log N)$
Quick Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N^2)$
Radix Sort	$\Omega(NK)$	$\Theta(NK)$	$O(NK$

B] Conclusion :-

In this experiment we have written C program to implement sorting algorithms. The algorithms that were implemented Insertion sort and Quick sort to sort an unsorted array.

A sorting algorithm is used to rearrange a given array or list elements according to a comparision operator on the elements.

When data is unsorted, one may need to search one item at a time, and he may not even find it. This is the reason why sorting algorithms are important. They help optimize searching the data from the dataset.