KJ's Educational Institutes

K J College Of Engineering & Management Research, Pune.

Department of E & TC

CLASS: S. E. (E &TC) **SUBJECT:-DSA**

Ex. No: 1 **Date:**

AIM : Student Database Management

You are developing a student result management system. The database should support updating records, adding new entries, searching for specific students, and sorting based on performance.

Using an array of structures, implement a student database with attributes: roll no, name, program, course, subject marks, total, and average. Support operations: display, search, and sort. (Students can additionally perform modify, append.)

OBJECTIVES

After performing this experiment, student should be able to:

- Bubble Sort.
- Linear Search

THEORY

1) Linear search:

In computer science, linear search or sequential search is a method for finding a particular value in a list , that consists in checking every one of its elements, one at a time and in sequence, until the desired one is found.

For a list with n items, the best case is when the value is equal to the first element of the list, in which case only 1 comparison is needed. The worst case is when the value is not in the list (or occurs only once at the end of the list), in which case n comparisons are needed.

Analysis of Linear Search:

The way in which big O is determined for a particular algorithm is algorithm analysis. Take for example the simple linear search algorithm that we mentioned previously and apply it to the value 3 and the ordered set

$$S = \{0, 1, 2, 3, 4, 5\}$$

Sequentially the algorithm would start from the index of the set, 0, and then look at each next value until it finds the value 3, thus making four comparisons.
But what about all of the other infinite number of integers that we could have looked for? Let's try the value 6. The algorithm will first check 6 against 0, 1, 2, 3, 4, and 5 until it halts without finding the value and thus makes six comparisons.
From this we can draw that the algorithm will make a number of comparisons equal to the input size, n, and thus is $O(n)$.

Input:
Data in the form an array.

Output :
For each algorithm output should be in the form of:
Original List of numbers.
Number to be searched with its position.

Difference between binary and linear search:
Binary search runs in O(log n) time whereas linear search runs in O(n) times thus binary search has better performance.

Applications:
1. Searching Telephone number of a particular person in Telephone book.
2. Finding information of a particular student in a large Student Database.
3. Simple Number Guessing game.

2) Bubble sort:

Bubble sort is a simple sorting algorithm. It works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements "bubble" to the top of the list. Because it only uses comparisons to operate on elements, it is a comparison sort.

Analysis of Bubble Sort:

Bubble sort has best-case complexity Ω (n). When a list is already sorted, bubble sort will pass through the list once, and find that it does not need to swap any elements. Thus bubble sort will make only n comparisons and determine that list is completely sorted. It will also use considerably less time than O (n^2) if the elements in the unsorted list are not too far from their sorted places. The worst case complexity is O (n2).

Efficiency:

For each pass the number of elements scanned for comparisons reduces by 1.

Number of comparisons in the first pass =(n-1)

Number of comparisons in the Second pass =(n-2)

Number of comparisons in the last pass =1

Thus the total number of comparisons at the end of the algorithm would have been

(n-1) + (n-2) +(n-3)+…+2+1= n(n-1)/2 =n2 /2 +O(n) = O(n2)

Hence the order of the bubble sort algorithm is O(n2) in worst case

Step-by-step example

Let us take the array of numbers **"5 1 4 2 8"**, and sort the array from lowest number to greatest number using bubble sort algorithm. In each step, elements written in bold are being compared.

First Pass:

(5 1 4 2 8) (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps them.
(1 5 4 2 8) (1 4 5 2 8), Swap since 5 > 4
(1 4 5 2 8) (1 4 2 5 8), Swap since 5 > 2
(1 4 2 5 8) (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(1 4 2 5 8) (1 4 2 5 8)
(1 4 2 5 8) (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) (1 2 4 5 8)
(1 2 4 5 8) (1 2 4 5 8)
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

Third Pass:

(1 2 4 5 8) (1 2 4 5 8)
(1 2 4 5 8) (1 2 4 5 8)
(1 2 4 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)
Finally, the array is sorted, and the algorithm can terminate.

Input:
Data in the form an array.

Output:
For each algorithm output should be in the form of:
Original List of numbers.
Sorted list of numbers.

Application:
1) Useful in managing large amount of data.
2) Finding information of a particular student in a large Student Database.
3) Simple Number Guessing game.

Algorithm:

1. **Linear Search:**

X is the number to be searched, A[N] is the Array of elements N, I is the index of array.

```
1. [Initialize Search]
I=1
A[N+1]=X
2. [Search Vector]
Repeat while A[I]!=X
I=I+1
3. [Successful Search?]
If I=N+1
Then Print 'Unsuccessful Search'
Return(0)
Else Print 'Successful Search'
Return (I).
```

2. **Bubble Sort:**

Given array K of N elements, algorithm sorts the elements in increasing(ascending) order. The variables Pass & Last denote pass counter & position of last unsorted element respectively. I is index of array elements and EXCHs is to count no of exchanges made by any pass.

```
1. (Initialize)
```

```
                                Last =N
2. (Loop)
        Repeat through step 5 for Pass= 1,2,…N-1
3. (Initialize Exchanges counter for this Pass)
                                EXCHS=0
4. Repeat for I=1,2,….,Last-1
If  K[I]>K[I+1]
Then    swap (K[I] & K[I+1])
EXCHS=EXCHS+1
5. (Were any exchanges made on this pass?)
If EXCHS==0
Then Return // mission accomplished; return early
Else Last=Last-1
6. (Finished)
Return //maximum number of passes required
```

CONCLUSION: -