**Linear search**

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

**Sample:**

int linearSearch(int ara[],int x)

{

for(int i=0; i<n; i++)

if(x==ara[i])

return i;

return -1;

}

**Analysis:**

Suppose have a Array of 5 element,

Ara={10,15,35,5,25}

Let x=20, we want know the existence of x in the array so the procedure is given below:

Step 1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 15 | 35 | 5 | 25 |

Compare x with 1st element of the array

ara[0], x!=ara[0] ,go to the next step

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 15 | 35 | 5 | 25 |

Step 2:

Compare x with 2nd element of the array

ara[1], x!=ara[1] , go to the next step

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 15 | 35 | 5 | 25 |

Step 3:

Compare x with 3rd element of the array

ara[1], x!=ara[2] , go to the next step

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 15 | 35 | 5 | 25 |

Step 4:

Compare x with 4th element of the array

ara[1], x!=ara[3] , go to the next step

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 15 | 35 | 5 | 25 |

Step 5:

Compare x with 5th element of the array

X==ara[4] ,so function will return index of ara[4]

**Time complexity**

Best case:

We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in the best case is constant (not dependent on n). So time complexity in the best case would be: Θ(1)

Worst case:

For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search() functions compares it with all the elements of arr[] one by one. Therefore, the worst case time complexity of linear search would be : Θ(n).

Average case:

For average case Sum all the calculated values and divide the sum by total number of inputs.

=

=

=

Ignoring the constant co-efficient, we can say that the complexity in average case of linear search is : O(n).

**Bubble Sort**

Bubble Sort is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares all the element one by one and sort them based on their values.

**Sample:**

void bubble(int ara[])

{

for(int i=0; i<n; i++)

for(int j=0; j<(n-i-1); j++)

if(ara[j]>ara[j+1])

swap(ara[j],ara[j+1]);

}

**Analysis:**

Let’s consider an array: (5, 1, 4, 2, 8)

We want to sort it in ascending order:

So,

**1st Iteration:**  
( **5** **1** 4 2 8 ) –> ( **1** **5** 4 2 8 ), Here, algorithm compares the first two elements, and swaps since 5 > 1.  
( 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.

**2nd Iteration:**  
( **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.

**3rd Iteration:**  
( **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** )

**Time Complexity**

In Bubble Sort, (n-1) comparisons will be done in the 1st pass, (n-2) in 2nd pass, (n-3) in 3rd pass and so on. So the total number of comparisons will be,

(n-1) + (n-2) + (n-3) + (n-4) + …… + 3 + 2 + 1

= (n-1)\*n/2

Ignoring the constant co-efficient, we can say that the complexity is: O(n^2)

By using the following process we have to do same number operation for best case, worst case and average case.

So complexity is: O(n^2)

**Optimized Bubble sort**

**Sample :**

void bubbleOpt(int ara[])

{

bool flg;

for(int i=0; i<n; i++)

{

flg=true;

for(int j=0; j<(n-i-1); j++)

if(ara[j]>ara[j+1])

swap(ara[j],ara[j+1]),flg=false;

if(!flg)

break;

}

}

It can be optimized by stopping the algorithm if inner loop didn’t cause any swap.

**Worst and Average Case Time Complexity:**O(n\*n). Worst case occurs when array is reverse sorted.

**Best Case Time Complexity:** O(n). Best case occurs when array is already sorted.