Heaven's Light is Our Guide

# RAJSHAHI UNIVERSITY OF ENGINEERING AND TECHNOLOGY

## Dept. of Computer Science & Engineering


***Course Code :*** *CSE 2202*
***Course Title :*** *Sessional based on CSE 2201*
***Experiment NO :*** *01*
***Experiment Name :*** *Complexity Analysis of Bubble, Selection, Insertion Sort & Linear, Binary search Algorithm*

***Date of Experiment :*** *10/10/2022*
***Date of Submission :*** *17/10/2022*

***MACHINE CONFIGURATION:***
Intel(R) Core(TM) i5-10210U CPU
@ 1.60GHz  2.11 GHz, 20.0 GB of RAM 1 TB of Hard disk


***Submitted By***

*Name: Md. Golam All Gaffar Tasin*
*Roll : 1903114*
*Section: B*
*Year: 2nd  Year(Even)*
*Dept. of CSE*

***Submitted to***

*Rizoan Toufiq*
*Assistant Professor, Dept. of CSE*
*Rajshahi University Of Engineering And Technology.*

Objective: Getting knowledge about complexity analysis of different algorithm.

Theory:

Bubble sort:

Procedure bubblesort (a: an array of element N>=2)

| | |
|---|---|
| for i :=1 to N-1 | 2 N |
| for j:=1 to N -i | $2\sum_{i=1}^{N}(N-i+1)$ |
| if(a[j] >a[j+1] ) | $\sum_{i=1}^{N}(N-i+1)$ |
| swap(a[j], a[j+1]) | T |

Procedure swap (i, j: two data will be swapped)

```
{ temp=i;
  i=j;
  J=temp; }
```

So the run-time $T(N) = 2N + 3\sum_{i=1}^{N}(N-i+1) + T$

$= 2N + 3N^2 - 3N(N+1)/2 + 3N + T$

$= (4N + 6N^2 - 3N^2 - 3N + 6N)/2 + T$

$= (3N^2 + 7N)/2 + T$

BEST CASE: In best case, never swap will be occurred since the data is sorted. So T=0

$T(N) = 2N + 3\sum_{i=1}^{N}(N-i+1)$

$= (3N^2 + 7N)/2$

$= O(N^2)$

WORST CASE: In worst case, swap will be occurred every time since the data is reversely sorted. So

$T = 3\sum_{i=1}^{N}(N-i+1)$

$T(N) = 2N + 3\sum_{i=1}^{N}(N-i+1) + 3\sum_{i=1}^{N}(N-i+1)$

$= 2N + 6N^2 - 6N(N+1)/2 + 6N$

$= (4N + 12N^2 - 6N^2 - 6N + 12N)/2$

$= 3N^2 + 5N$

$= O(N^2)$

AVERAGE CASE: In the average case, it is equally probable the number of comparisons is any number between 1 and k inclusive for each 'If' condition: if (a[j]> a[j+1])

So T = (1+2+3+…..+k)/k

= (k+1)/2

$T(N) = (3N^2 + 7N)/2 + \sum_{k=1}^{N} (k+1)/2$

$= (3N^2 + 7N)/2 + N(N+1)/4 + N/2$

$= (7N^2 + 17N)$

$= O(N^2)$

For Worst Case $O(N^2)$ where T maximum

For Best Case $O(N^2)$ where T=0

Average Case $O(N^2)$

The expected and best cases are identical, so bubble sort is Ω (N 2 ) and Θ (N 2 ).

## Selection Sort:

| Alg.: SELECTION-SORT(A) | | |
|---|---|---|
| | cost | Times |
| n ← length[A] | $c_1$ | 1 |
| for j ← 1 to n - 1 | $c_2$ | n-1 |
| do smallest ← j | $c_3$ | n-1 |
| for i ← j + 1 to n | $c_4$ | $\sum_{j=1}^{n-1}(n-j+1)$ |
| ≈ n2/2 comparisons, do if A[i]<A[smallest] | $c_5$ | $\sum_{j=1}^{n-1}(n-j)$ |
| then smallest ← i | $c_6$ | $\sum_{j=1}^{n-1}(n-j)$ |
| ≈ n exchanges, exchange A[j] ↔ A[smallest] | $c_7$ | n-1 |

So the run time $T(N) = 1 + 2(N-1) + \sum_{j=1}^{N-1} (N-j+1) + 2\sum_{j=1}^{N-1} (N-j) + (N-1)$

BEST CASE: In best case, never swap will be occurred since the data is sorted. So c5,c6 and c7 is 0

$T(N) = (3N-2) + N(N+1)/2$

$= (3N-2) + (N^2 + N)/2$

$= (6N-4+N^2+N)/2$

$=(N^2+7N-4)/2$

$= O(N^2)$

WORST CASE: In worst case, swap will be occurred every time since the data is reversely sorted. So

$T(N) = (3N-2) + N(N+1)/2 + N(N+1) +(N-1)$

$= (4N-3) + (N^2+N)/2 + (N^2+N)$

$= (8N-6+ N^2+N+2 N^2+2N)/2$

$= (3 N^2+11N -6)/2$

$= O(N^2)$

AVERAGE CASE: In the average case, it is equally probable the number of comparisons is any number between 1 and k inclusive for each c5 and c6

$T(N) = (4N-3)+(N^2+N)/2 + 2\sum_{k=1}^{N} (k+1)/2$

$= (8N-6+ N^2+N)/2 + N(N+1)/2$

$=(N^2+9N-6+N^2+N)/2$

$=(2N^2+10N-6)/2$

$= O(N^2)$

## Insertion Sort:

INSERTION-SORT(A)

| | |
|---|---|
| for i = 0 to n | n |
| key = A[i] | n-1 |
| j = i – 1 | n-1 |
| while j >= 0 and A[j] > key | $\sum_{j=1}^{n-1} (n)$ |
| A[j + 1] = A[j] | $\sum_{j=1}^{n-1} (n-1)$ |
| j = j – 1 | $\sum_{j=1}^{n-1} (n-1)$ |
| A[j + 1] = key | n-1 |

Total Running Time of Insertion sort $T(n) = C_1 * n + ( C_2 + C_3 ) * ( n - 1 ) + C_4 * \Sigma^{n-1}_{j=1}( n_j ) + ( C_5 + C_6 ) * \Sigma^{n-1}_{j=1}( n_j ) + C_8 * ( n - 1 )$

Best Case

In Best Case i.e., when the array is already sorted, n = 1

$T(n) = C_1 * n + ( C_2 + C_3 ) * ( n - 1 ) + C_4 * ( n - 1 ) + ( C_5 + C_6 ) * ( n - 2 ) + C_8 * ( n - 1 )$

$= O(n)$

Worst Case

In Worst Case i.e., when the array is reversely sorted (in descending order), n = j

$T(n) = C_1 * n + ( C_2 + C_3 ) * ( n - 1 ) + C_4 * ( n - 1 ) ( n ) / 2 + ( C_5 + C_6 ) * ( ( n - 1 ) (n ) / 2 - 1) + C_8 * ( n - 1 )$

$= O(n^2)$

Average Case

Let's assume that $n_i = (j-1)/2$ to calculate the average case

$T( n ) = C_1 * n + ( C_2 + C_3 ) * ( n - 1 ) + C_4/2 * ( n - 1 ) ( n ) / 2 + ( C_5 + C_6 )/2 * ( ( n - 1 ) (n ) / 2 - 1) +$

$C_8 * ( n-1 )$

$= O(n^2)$

Linear Search iterative:

```
procedure linear_search (list, value)
  for each item in the list                    n
      if match item == value                   1
          return the item's location           1
  return null                                  1
```

Total Running Time $T(n) = n + 3$

Best Case

The number of comparisons in this case is 1. Thereforce, Best Case Time Complexity of Linear Search is O(1).

Worst Case:

T(n) = n+3

= O(n)

Average Case

Number of comparisons = 1+2+3+…..+n

= n(n+1) / 2

If element P is not in the list, then Linear Search will do n comparisons

Average number of comparison = [{n(n+1) / 2}+1]/(n+1)

= n/2 + n/(n+1)

= O(n)

## Binary Search recursive:

```
binarySearch(arr, x, low, high)
    repeat till low = high
        mid = (low + high)/2
        if (x == arr[mid])
        return mid

        else if (x > arr[mid]) // x is on the right side
            low = mid + 1

        else                   // x is on the left side
            high = mid - 1
```

Best Case

The best case of Binary Search occurs when:

- The element to be search is in the middle of the list

Therefore, Best Case Time Complexity of Binary Search is O(1).

Average Case

There are two cases:

Case 1: The element P can be in N distinct indexes from 0 to N-1.
Case 2: There will be a case when the element P is not present in the list. There are N case 1 and one case 2. So, there are N+1 distinct cases to consider in total.

If element P is in index K, then Binary Search will do K+1 comparisons.

This is because:

The element at index N/2 can be found in 1 comparison as Binary Search starts from middle.

Similarly, in the 2nd comparisons, elements at index N/4 and 3N/4 are compared based on the result of 1st comparison.

Total number of comparisons = 1 * (1) + 2 * (2) + 3 * (4) + ... + logN * (2^(logN-1))

= 1 + 4 + 12 + 32 + ... = 2^logN * (logN - 1) + 1

= N * (logN - 1) + 1

= O(nlogN)

Worst Case

The worst case of Binary Search occurs when:

- The element is to search is in the first index or last index

In this case, the total number of comparisons required is logN comparisons.

Therefore, Worst Case Time Complexity of Binary Search is O(logN).

## Data Table:

| No.of Data | Bubble sort | Selection sort | Insertion Sort | Linear search | Binary search |
|---|---|---|---|---|---|
| 10000 | 239 m s | 234 m s | 62 m s | 631000 ns | 107000 ns |
| 30000 | 2968 m s | 3156 m s | 500 m s | 999000 ns | 299000 ns |
| 50000 | 8640 m s | 8624 m s | 1781 m s | 6323000 ns | 7974000 ns |
| 70000 | 12349 ms | 14247 ms | 2105 ms | 7345000 ns | 9653000 ns |
| 90000 | 14586 ms | 16120 ms | 2351 ms | 8105466 ns | 10255165 ns |

## Graph:

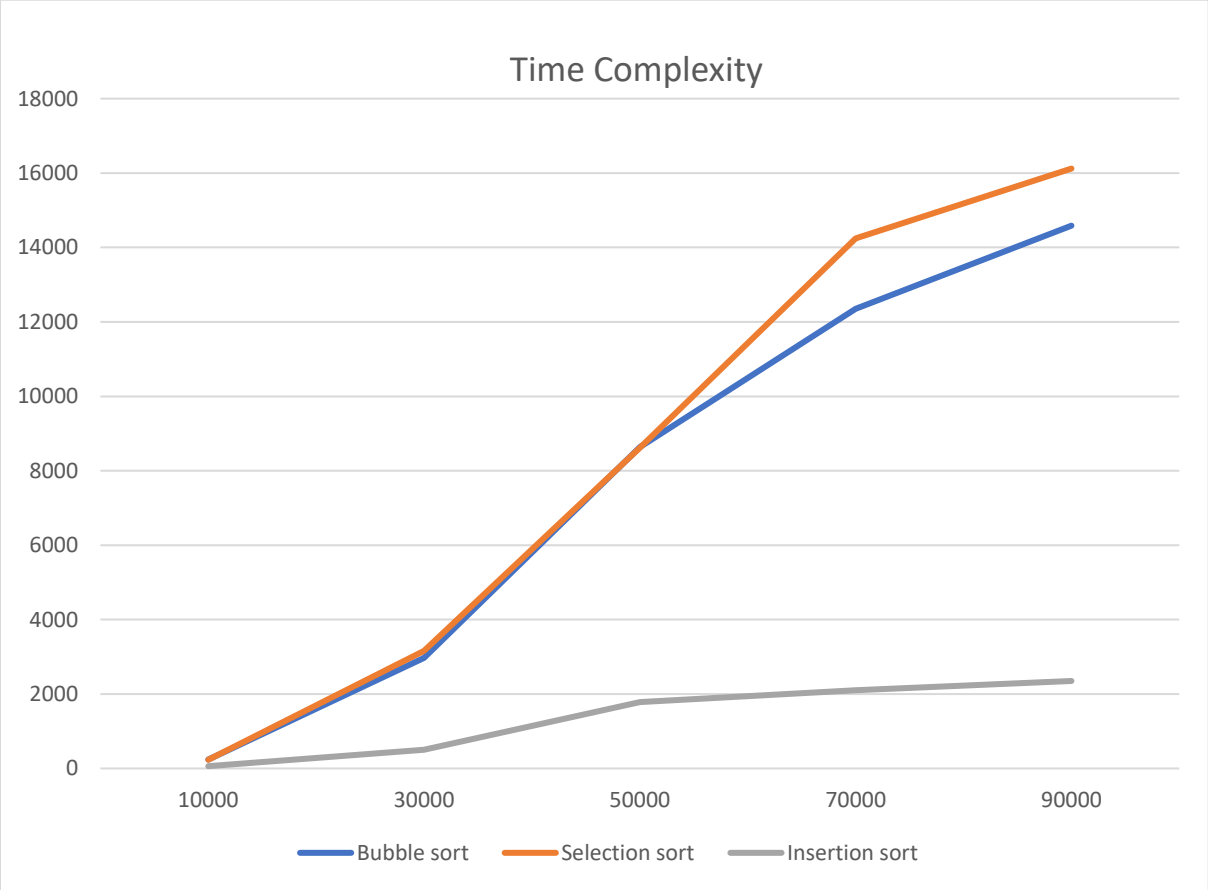


Fig 1: line graph of various sorting

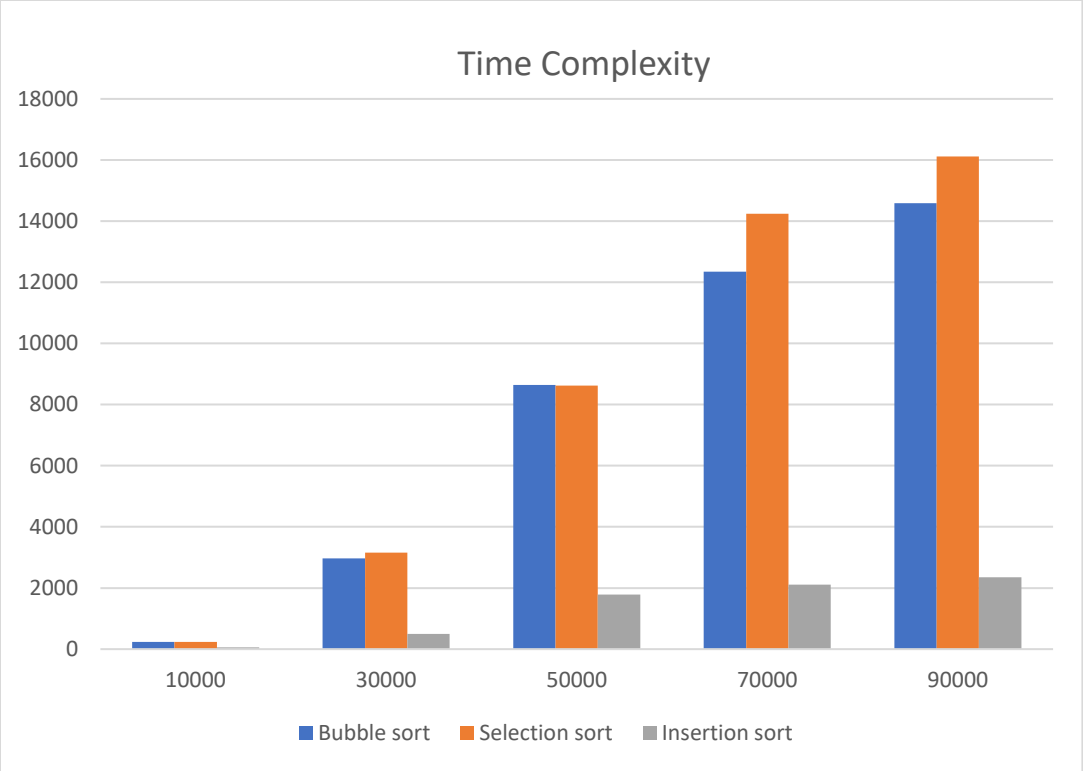


Fig 2: bar chart for various sorting

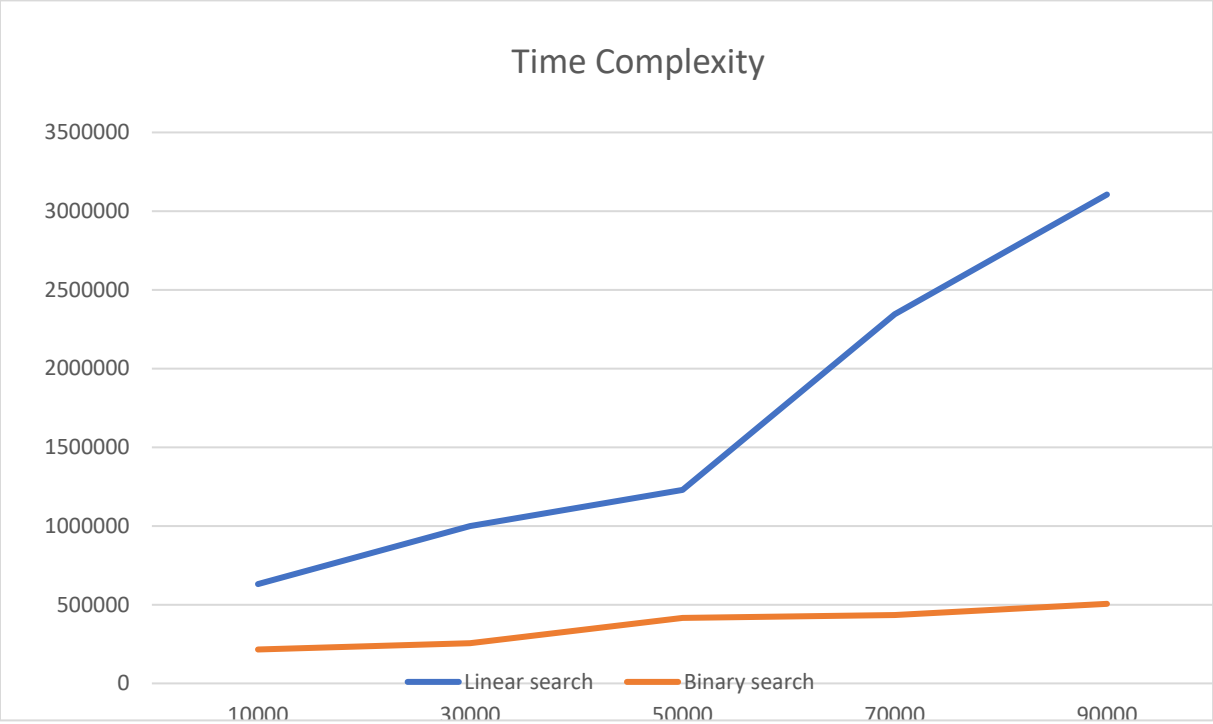


Fig 3: Time complexity of linear and binary search

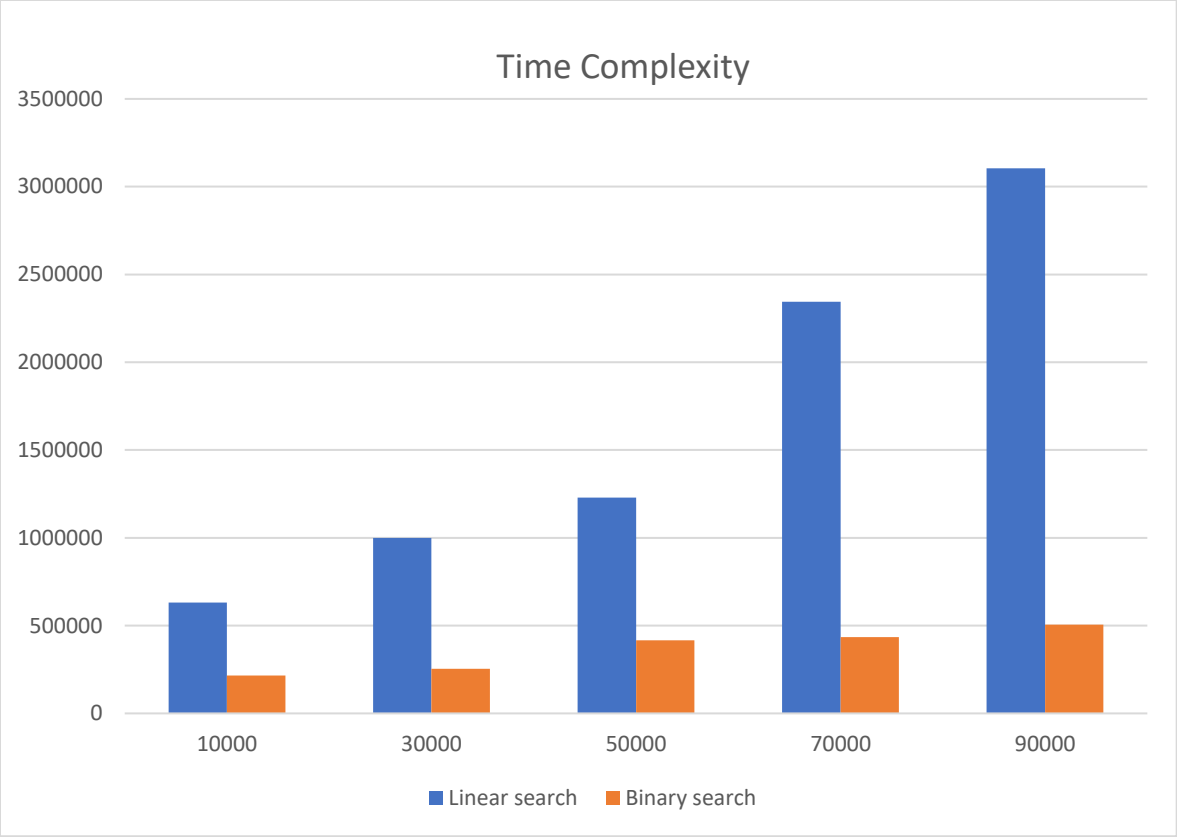


Fig 4: Linear and Binary search

## Discussion:

Today we have analyzed the time complexity of some searching and sorting algorithms. We have used time function to calculate the time between the required function to be executed. We have measured the time in milli seconds. If we see the graphical output and bar graph for average case complexity, we can see that the average case for bubble sort is the worst. Clearly selection sort and insertion sort are better in terms of time complexity. Between insertion sort and selection sort insertion sort is slightly better. Though the average case complexity for both of them is O(n^2) theoretically. But in real time we can observe that insertion sort is better. A huge difference comes in selection sort. Its best case, average case worst case all are same theoretically but in fig 1 we can see that the average case runs more efficiently. And its not a better choice if the data is in the best case(sorted). Finally, in insertion sort we can see a better output in every term. Its average case is average not only theoretically but also practically, its best case is just like bubble sort but its worst case is more efficient than bubble sort. So, after analyzing all we can say insertion sort is best among them. Now for searching algorithm we see that the binary search is far better than linear search. In terms of output we have observed in our machine we see that when the input size increases the time taken by linear search increases visually. But it has little effect on binary search. Compared to linear search it is tensed to very low. So binary search is very efficient. But the disadvantage of it is that the data needs to be sorted.