Batch: B2 Roll No.: 1711091

Experiment No.__ 1 __

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of selection sort/ Insertion sort

Objective: To analyse performance of sorting methods

CO to be achieved:

Sr. No	Objective
CO 1	Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations.
CO 2	Analyse and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.
CO 3	Analyse and solve problems for different string matching algorithms.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Insertion_sort**
4. **http://www.sorting-algorithms.com/insertion-sort**
5. **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Insertion_sort.html**
6. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/insertionSort.htm**
7. **http://en.wikipedia.org/wiki/Selection_sort**
8. **http://www.sorting-algorithms.com/selection-sort**
9. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/selectionSort.htm**

10. **http://courses.cs.vt.edu/~csonline/Algorithms/Lessons/SelectionCardSort/selectioncardsort.html**

Pre Lab/ Prior Concepts:
Data structures, sorting techniques

Historical Profile:
There are various methods to sort the given list. As the size of input changes, the performance of these strategies tends to differ from each other. In such case, the priori analysis can helps the engineer to choose the best algorithm.

New Concepts to be learned:
Space complexity, time complexity, size of input, order of growth.

Algorithm InsertionSort

INSERTION_SORT (*A,n*)
//The algorithm takes as parameters an array *A*[1.. *n*] and the length *n* of the array.
//The array *A* is sorted in place: the numbers are rearranged within the array
// A[1..n] of eletype, n: integer

```
FOR j ← 2 TO length[A]
    DO  key ← A[j]
        {Put 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
```

Algorithm SelectionSort

SELECTION_SORT (A,n)
//The algorithm takes as parameters an array *A*[1.. *n*] and the length *n* of the array.
//The array *A* is sorted in place: the numbers are rearranged within the array
// A[1..n] of eletype, n: integer

```
FOR i ← 1 TO n-1 DO
    min j ← i;
    min x ← A[i]
    FOR j ← i + 1 to n do
        IF A[j] < min x then
```

min $j \leftarrow j$
min $x \leftarrow$ A[j]
A[min j] $\leftarrow$ A [i]
A[i] $\leftarrow$ min x

Implementation:

```
import random, time
import matplotlib.pyplot as plt

inc = 700
x1 = []
y1 = []
x2 = []
y2 = []

for val in range(10):

  a = random.sample(range(0,100000), 2200+(val*inc))
  b = a[:]
  tic = time.time()

  #Selection
  for i in range(len(a)-1):
    mini = i
    for j in range(i+1, len(a)):
      if a[mini] > a[j]:
        mini = j
    a[i],a[mini] = a[mini],a[i]

  toc = time.time()

  x1.append(2200+(val*inc))
  y1.append(toc - tic)

  toc = time.time()
  #Insertion
  for i in range(1, len(b)):
    j = i-1
    key = b[i]
    while j>=0 and key<b[j]:
      b[j+1] = b[j]
      j-=1
    b[j+1] = key

  tac = time.time()
  x2.append(2200+(val*inc))
```

```
    y2.append(tac - toc)

print(x1, y1, sep = '\n')
print(x2, y2, sep = '\n')

plt.plot(x1, y1, label = 'Selection', marker = 'o')
plt.plot(x2, y2, label = 'Insertion', marker = 'o')
plt.legend()
plt.xlabel(' Number of Values ')
plt.ylabel(' Time taken per loop(s) ')
plt.show()
```

The space complexity of Insertion sort:

Insertion sort is an O(1) space complexity algorithm, since the array is passed by reference, and all the other variables are input independent local variables. The array doesn't require extra space since its base address is available and the contents can be retrieved by iterating over it. In our example, we have i, j, key, a and n as the variables which are locally needed by the function.

The space complexity of Selection sort:

Selection sort is an O(1) space complexity algorithm, since the array is passed by reference, and all the other variables are input independent local variables. The array doesn't require extra space since its base address is available and the contents can be retrieved by iterating over it. In our example, we have i, j, key, min, a and n as the variables which are locally needed by the function.

Time complexity for Insertion sort:

The worst case time complexity for insertion sort is given by the following equation:

$T_{insertion} >= (n-1)(2 + 1 +1) + (3)(n-1)(n-2)/2 + 1 = 4n – 3 + 1.5(n^2 – 3n + 2) = 1.5n^2 - 0.5n$

So,

$T_{insertion} >= 1.5n^2 - 0.5n$

In asymptotic terms,

Insertion sort is an $O(n^2)$ algorithm

In best case, lets say the sort works on an array already sorted.
Then, the inner while loop will never execute and only the outer for loop runs, which gives the **best case as O(n)**

Time complexity for selection sort:

The worst case time complexity for selection sort is given by the following equation:

$T_{selection} >= (n-1)(2 + 2 + 1) + (4)(n-1)(n-2)/2 + 1 = 5n – 4 + 2(n^2 – 3n + 2) = 2n^2 - n$

So,

$T_{selection} >= 2n^2 - n$

In asymptotic terms,

Insertion sort is an $O(n^2)$ algorithm

Even in the worst case, the sorting algorithm will run through both of its for loops, which gives the algorithm a **best case running time of $O(n^2)$**

Graphs for varying input sizes: (Insertion Sort & Selection sort)

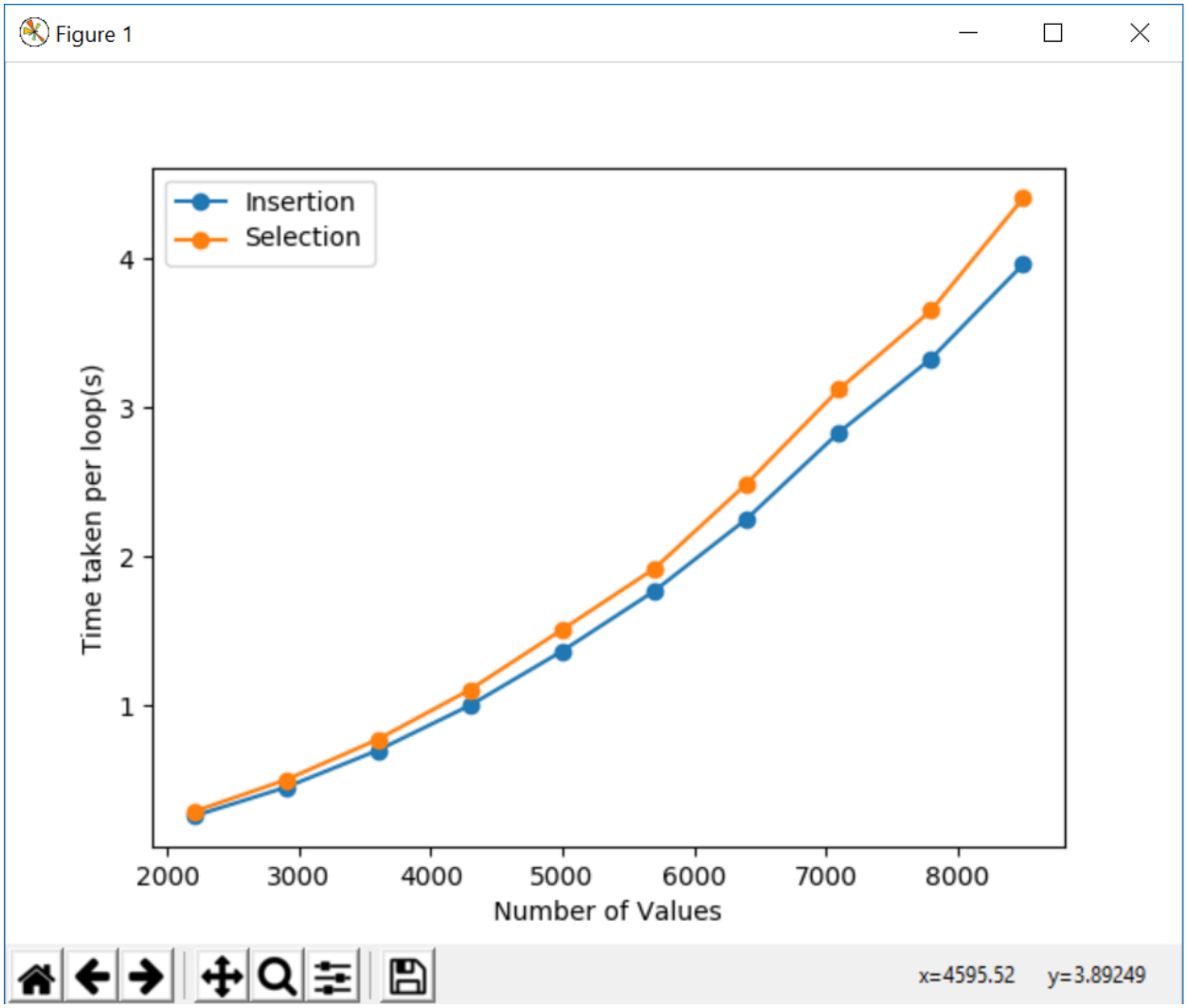
Figure 1
Insertion
Selection
Time taken per loop(s)
1
2
3
4
2000
3000
4000
5000
6000
7000
8000
Number of Values
x=4595.52
y=3.89249

CONCLUSION: