

| **Title: Implementation of selection sort/ Insertion sort** |
| --- |



**Objective:** To analyse performance of sorting methods



**CO to be achieved:**

| CO 1 | Analyze the asymptotic running time and space complexity of 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**](http://en.wikipedia.org/wiki/Insertion_sort)
4. [**http://www.sorting-algorithms.com/insertion-sort**](http://www.sorting-algorithms.com/insertion-sort)
5. [**http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Insertion\_sort.html**](http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Insertion_sort.html)
6. [**http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/insertionSort.htm**](http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/insertionSort.htm)
7. [**http://en.wikipedia.org/wiki/Selection\_sort**](http://en.wikipedia.org/wiki/Selection_sort)
8. [**http://www.sorting-algorithms.com/selection-sort**](http://www.sorting-algorithms.com/selection-sort)
9. [**http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/selectionSort.htm**](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.



**Topic: Sorting Algorithms**

**Theory:** Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, 1< k ≤n, yielding k sub problems. These sub problems must be solved and then a method must be found to combine sub solutions into a solution of the whole. If the sub problems are still relatively large, then the divide-and-conquer strategy can possibly be reapplied. Often the sub problems resulting from a divide-and-conquer design are the same type as the original problem. For those cases the reapplication of the divide-and- conquer principle is naturally expressed by a recursive algorithm. Now smaller and smaller subproblems of the same kind are generated until eventually sub problems that are small enough to be solved without splitting are produced.

**Algorithm Insertion Sort**

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 Selection Sort**

SELECTION\_SORT (A,n)

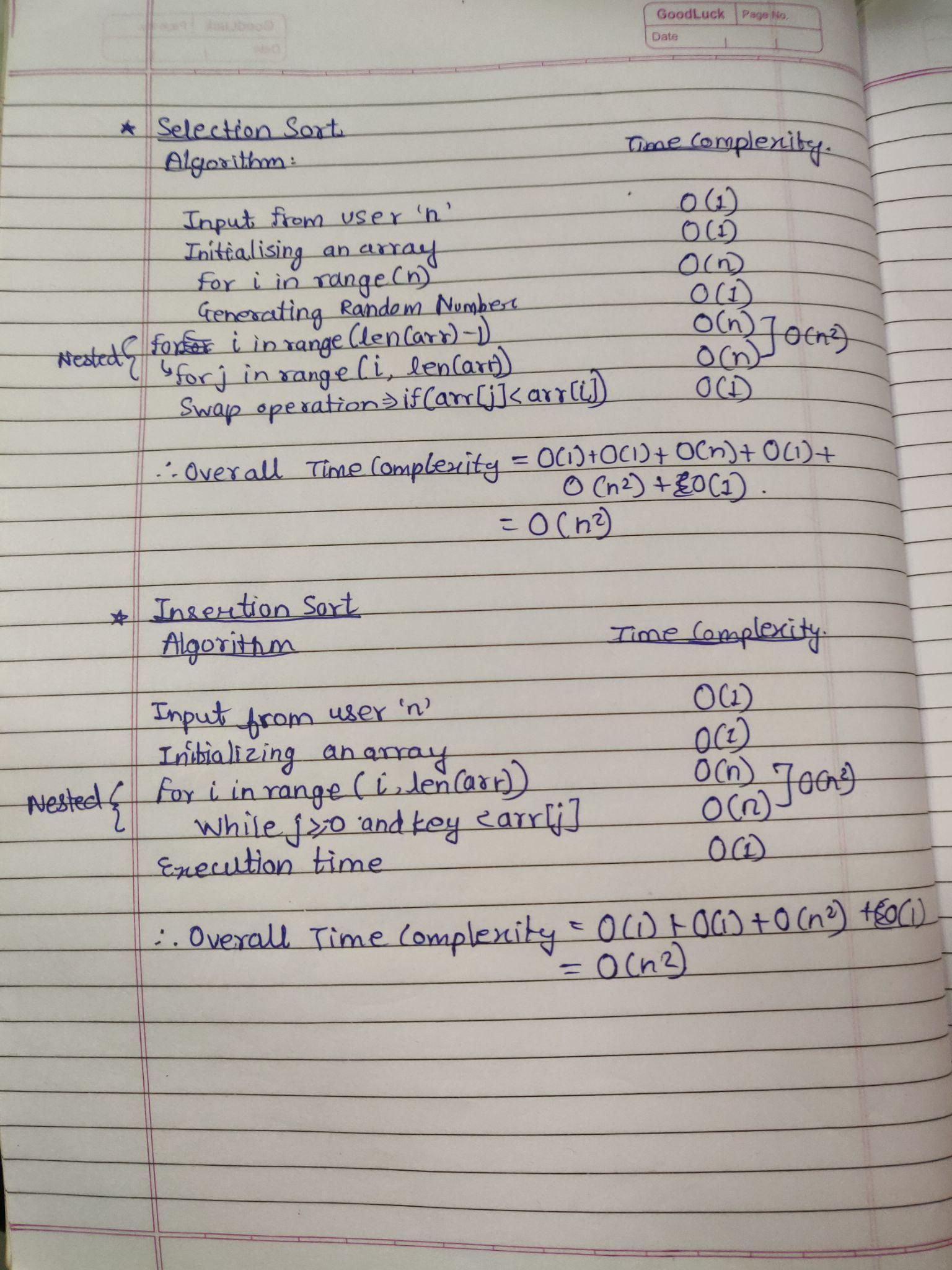
//The algorithm takes as parameters an array *A*[1.. *n*] and the length *n* of the array.

//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* ← *j*  
            min *x* ← A[j]  
    A[min *j*] ← A [*i*]  
    A[*i*] ← min *x*

**Priori Analysis(Selection sort & Insertion sort):**

****

**CODE(Selection sort):**

import random

import time

start=time.time()

n = int(input('Enter the number of input: '))

arr = []

for i in range(n):

element = random.randint(1,n)

arr.append(element)

for i in range(len(arr) - 1):

for j in range(i, len(arr)):

if(arr[j]<arr[i]):

temp = arr[i]

arr[i] = arr[j]

arr[j] = temp

end = time.time()

print(f 'Execution time: {(end - start)/1000}')

**CODE(Insertion Sort):**

import random

import time

def insertion\_sort(arr):

for i in range(1, len(arr)):

key = arr[i]

j = i - 1

while j >= 0 and key < arr[j]:

arr[j + 1] = arr[j]

j -= 1

arr[j + 1] = key

size = int(input("Enter the size of the array: "))

random\_numbers = [random.randint(1, 100) for \_ in range(size)]

start\_time = time.time()

insertion\_sort(random\_numbers)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time: {:.6f} seconds".format(execution\_time))

**The space complexity of Insertion sort:**

**O(1)**

**The space complexity of Selection sort:**

**O(1)**

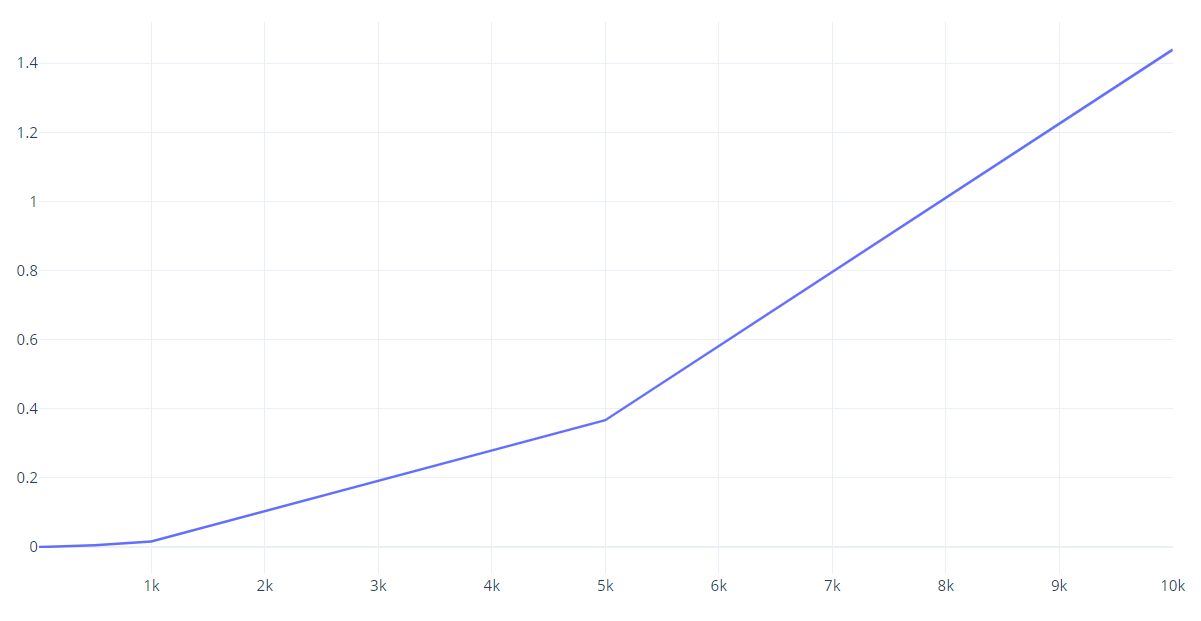
**Time complexity for Insertion sort:**

| **Input** | **1** | **2** | **3** | **4** | **5** | **Avg** |
| --- | --- | --- | --- | --- | --- | --- |
| **10** | 0.000012 | 0.000014 | 0.000014 | 0.000016 | 0.000013 | 0.0000138 |
| **50** | 0.000066 | 0.000064 | 0.000067 | 0.000057 | 0.000068 | 0.0000644 |
| **100** | 0.000283 | 0.000176 | 0.000350 | 0.000241 | 0.000186 | 0.0002472 |
| **500** | 0.004589 | 0.003328 | 0.005432 | 0.005128 | 0.004178 | 0.004531 |
| **1000** | 0.013851 | 0.013984 | 0.014239 | 0.021104 | 0.015574 | 0.0157504 |
| **5000** | 0.383953 | 0.359406 | 0.356862 | 0.341238 | 0.392071 | 0.366706 |
| **10000** | 1.506540 | 1.452116 | 1.371526 | 1.485340 | 1.380041 | 1.4391126 |

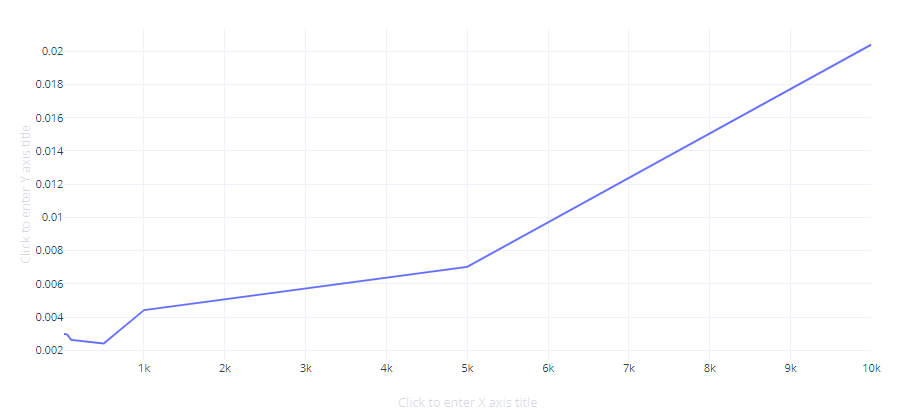
**Time complexity for selection sort:**

| **Input** | **1** | **2** | **3** | **4** | **5** | **Avg** |
| --- | --- | --- | --- | --- | --- | --- |
| **10** | 0.002042 | 0.002753 | 0.002682 | 0.001873 | 0.005588 | 0.002988 |
| **50** | 0.003228 | 0.002802 | 0.001757 | 0.00217 | 0.00473 | 0.002937 |
| **100** | 0.001563 | 0.001987 | 0.005545 | 0.0019 | 0.00208 | 0.002615 |
| **500** | 0.003249 | 0.002392 | 0.002147 | 0.001863 | 0.002374 | 0.002405 |
| **1000** | 0.002683 | 0.003161 | 0.001798 | 0.004844 | 0.009586 | 0.004415 |
| **5000** | 0.008973 | 0.00581 | 0.006293 | 0.006932 | 0.007081 | 0.007018 |
| **10000** | 0.022314 | 0.020872 | 0.02061 | 0.020702 | 0.017521 | 0.020404 |

**Graphs for varying input sizes: (Insertion Sort)**

****

**Graphs for varying input sizes: (Selection Sort)**



**CONCLUSION:** Studied about Selection sort and Insertion sort and priori analysis of the algorithm.