Design Algorithm and Analysis

Lab – 1: Insertion Sort

```
Aim :
To Perform Insertion sort on the Randomly generated arrays
```

```
Introduction :
Insertion sort algorithm involves the sorted list created based on an
iterative comparison of each element in the list with its adjacent element.
An index pointing at the current element indicates the position of the sort.
At the beginning of the sort (index=0), the current value is compared to the
adjacent value to the left. If the value is greater than the current value, no
modifications are made to the list; this is also the case if the adjacent
value and the current value are the same numbers.
However, if the adjacent value to the left of the current value is lesser,
then the adjacent value position is moved to the left, and only stops moving
to the left if the value to the left of it is lesser.
The diagram illustrates the procedures taken in the insertion algorithm on an
unsorted list. The list in the diagram below is sorted in ascending order
(lowest to highest).
```

```
PseudoCode:
insertionSort(array)
  mark first element as sorted
  for each unsorted element X
    'extract' the element X
    for j <- lastSortedIndex down to 0
      if current element j > X
        move sorted element to the right by 1
    break loop and insert X here
end insertion sort
```

```
FOR AVERAGE CASE

import numpy
arr = numpy.random.randint(10, size=(10))

import time
start_time = time.time()

def insertion_sort(arr):
    count = 0
    for i in range(0,len(arr)):
        val=arr[i]
        j=i-1
        while j>=0 and val<arr[j]:
            arr[j+1] = arr[j]
            j-=1
            count+=1
        arr[j+1]=val
    return(arr, count)


print("unsorted array is : " )
print(arr)

arr, count = insertion_sort(arr)
end_time =time.time()
compilation_time = end_time - start_time

print("sorted array is : " ,arr)
print("number of times loop : ",count)
print("compilation time : ",compilation_time)
```

Output :

```
unsorted array is : [8 8 0 4 5 0 2 9 2 0]

sorted array is : [0 0 0 2 2 4 5 8 8 9]

number of times loop : 26

compilation time : 0.0009992122650146484
```

```
FOR BEST CASE :
import numpy
arr = numpy.random.randint(10, size=(10))

import time
start_time = time.time()

def insertion_sort(arr):
    count = 0
    for i in range(0,len(arr)):
        val=arr[i]
        j=i-1
        while j>=0 and val<arr[j]:
            arr[j+1] = arr[j]
            j-=1
            count+=1
        arr[j+1]=val
    return(arr, count)

arr, count = insertion_sort(arr)

print("unsorted array is : " ,arr)

arr, count = insertion_sort(arr)
end_time =time.time()
compilation_time = end_time - start_time

print("sorted array is : " ,arr)
print("number of times loop : ",count)
print("compilation time : ",compilation_time)
```

Output :

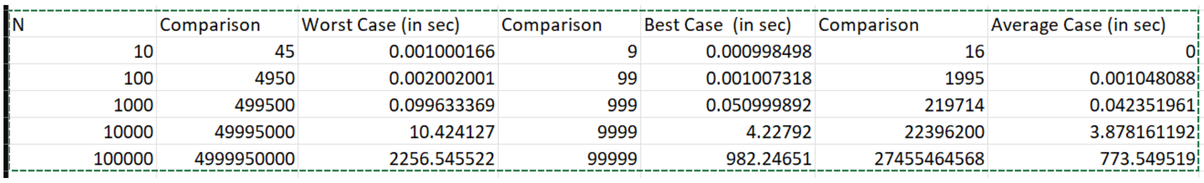
```
unsorted array is :

[ 0 0 1 1 1 3 6 7 7 7 8 8 11 11 13 13 13 14 15 16 16 18 18 19 20 20 21 22 24
27 27 28 32 32 33 33 38 39 39 40 40 40 43 43 44 46 47 48 48 49 49 54 55 58 61
62 62 63 63 64 65 66 66 67 68 69 71 71 72 72 72 73 74 78 78 79 80 80 83 84 84
84 87 92 92 92 93 93 93 94 94 94 95 96 96 97 98 98 98 99]

sorted array is :

[ 0 0 1 1 1 3 6 7 7 7 8 8 11 11 13 13 13 14 15 16 16 18 18 19 20 20 21 22 24
27 27 28 32 32 33 33 38 39 39 40 40 40 43 43 44 46 47 48 48 49 49 54 55 58 61
62 62 63 63 64 65 66 66 67 68 69 71 71 72 72 72 73 74 78 78 79 80 80 83 84 84
84 87 92 92 92 93 93 93 94 94 94 95 96 96 97 98 98 98 99]

 number of times loop : 0

compilation time : 0.0011460781097412
```

```
for Worst Case
import numpy
a = numpy.random.randint(100, size=(10))
b = numpy.sort(a)
arr = numpy.flip(b)

import time
start_time = time.time()

def insertion_sort(arr):
    count = 0
    for i in range(0,len(arr)):
        val=arr[i]
        j=i-1
        while j>=0 and val<arr[j]:
            arr[j+1] = arr[j]
            j-=1
            count+=1
        arr[j+1]=val
    return(arr, count)



print("unsorted array is : " ,arr)

arr, count = insertion_sort(arr)
end_time =time.time()
compilation_time = end_time - start_time

print("sorted array is : " ,arr)
print("number of times loop : ",count)
print("compilation time : ",compilation_time)
```

Output :

```
unsorted array is :

[99 99 98 98 97 96 96 96 95 95 92 92 90 88 88 85 85 84 83 82 80 79 79 74 73 72
71 70 69 68 67 66 66 65 65 65 65 63 63 57 57 57 57 56 56 54 54 53 53 52 51 50
50 50 47 46 43 42 42 42 42 41 41 36 36 35 35 34 32 31 30 30 30 26 25 24 21 20
19 17 17 15 15 15 14 13 12 10 10 9 9 9 6 4 4 2 2 1 0 0]

sorted array is :

[ 0 0 1 2 2 4 4 6 9 9 9 10 10 12 13 14 15 15 15 17 17 19 20 21 24 25 26 30 30
30 31 32 34 35 35 36 36 41 41 42 42 42 42 43 46 47 50 50 50 51 52 53 53 54 54
56 56 57 57 57 57 63 63 65 65 65 65 66 66 67 68 69 70 71 72 73 74 79 79 80 82
83 84 85 85 88 88 90 92 92 95 95 96 96 96 97 98 98 99 99]

number of times loop : 4897

compilation time : 0.0010004043579101562
```

Result :



N	Comparison	Worst Case (in sec)	Comparison	Best Case (in sec)	Comparison	Average Case (in sec)
10	45	0.001000166	9	0.000998498	16	0
100	4950	0.002002001	99	0.001007318	1995	0.001048088
1000	499500	0.099633369	999	0.050999892	219714	0.042351961
10000	49995000	10.424127	9999	4.22792	22396200	3.878161192
100000	4999950000	2256.545522	99999	982.24651	27455464568	773.549519

Application :

1. Since the time complexity of Insertion sort can go to O(N^2), it is only useful when we have a lesser number of elements to sort in an array.
2. Insertion sort is an in-place algorithm, meaning it requires no extra space.
3. Maintains relative order of the input data in case of two equal values (stable).

References :

Insertion Sort is O(n log n)
June 2006Theory of Computing Systems 39(3)
DOI:10.1007/s00224-005-1237-z
by Micheal A Bender , Martin Farah -Colton and Miguel A Mosterio