Data Analysis and Algorithm

Practical 1

Write a Program to implement Insertion sort

&

Find the run time of the algorithm

Name – Yash Vasudeo Prajapati

Rollno - 022

MSc. Computer Science

1) Write a program to implemet insertion sort & find the running time of the algorithm

→ Theory:-

Insertion sort is a simple sorting algorithm that build the final sorted array one item at a item. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort or merge sort.

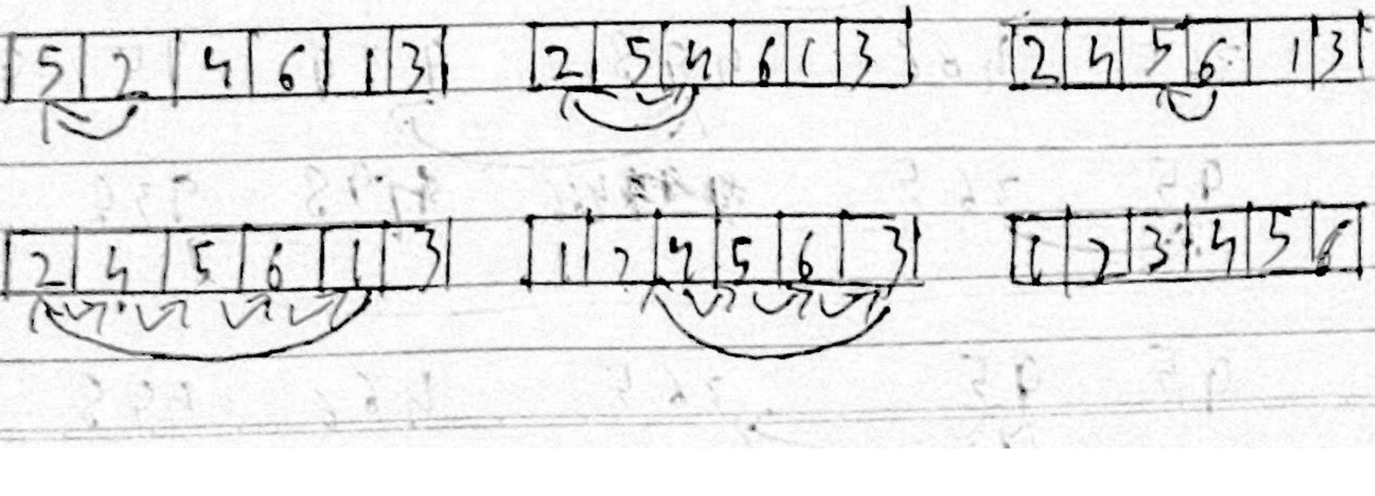
The Big O Notation for insertion sort is:-

$O(n^2)$ for n number of inputs

i.e

when $n = 10$, $n^2 = 100$

Insertion sort is slower then merge sort



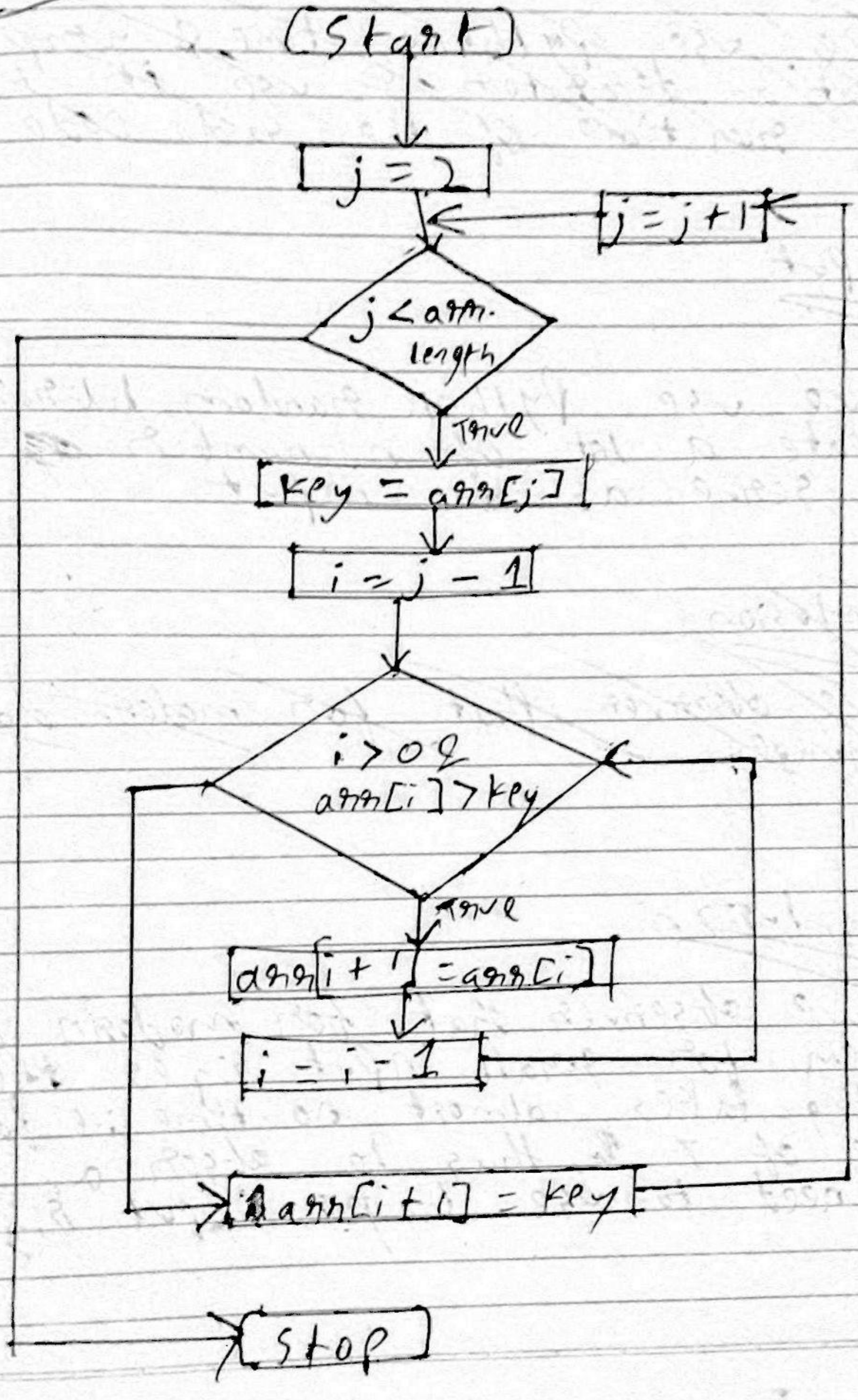
→ Algorithm

```
for i = 1 to arr.length
    key = arr[i]
    j = i - 1
    while j >= 0 & key < arr[j]:
        arr[j + 1] = arr[j]
        j -= 1
    arr[j+1] = key
```

→ Example

```
365   939   466   498   95    95
365   939   466   498   95    95
365   466   939   498   95    95
365   466   498   939   95    95
95    365   466   498   939   95
95    95    365   466   498   939
```

→ flowchart


Start
j = 1
j = j+1
j < arr.length
True
key = arr[j]
i = j - 1
i > 0 &
arr[i] > key
True
arr[i+1] = arr[i]
i = i - 1
arr[i+1] = key
Stop

⇒ Runtime of insertion sort algorithm

Insertion sort follows a linear method, i.e it loops over the indices of the array

We insert an element supposed 0 in an array [1, 2, 3, 4, 5] in this case 0 will be in the last position & thus every element has to be shifted once i.e K number of elements were shifted with c number of lines of code.

If an array is naturally in desending order then to sort such array it will have to go through all the elements i.e shift all numbers of elements, $K=1$, $K=2$ upto $K=n-1$ thus we spend

$$c\cdot 1 + c\cdot 2 + c\cdot 3 \ldots c(n-1) \text{ time}$$

which could be simplified as:

$$c(1+2+3\ldots(n-1))$$

i.e

$1+2+3\ldots(n-1)$ is the arithmetic series given by

$$c\cdot(n-1+1)((n-1)/2) \rightarrow \frac{cn^2}{2} - \frac{cn}{2}$$

$$\frac{c}{2}(n^2) - \frac{c}{2}(n)$$

$$\frac{c}{2}(n^2 - n)$$

Using big O notation we discard low-order term n & c/2 as c/2 can be considered c thus we get the runtime of worst case

$$O(n^2).$$

On the other hand if all element were at a constant number of position from sorted position, then they would have a constant c

eg. if all elements would be 2 away from position that would give us

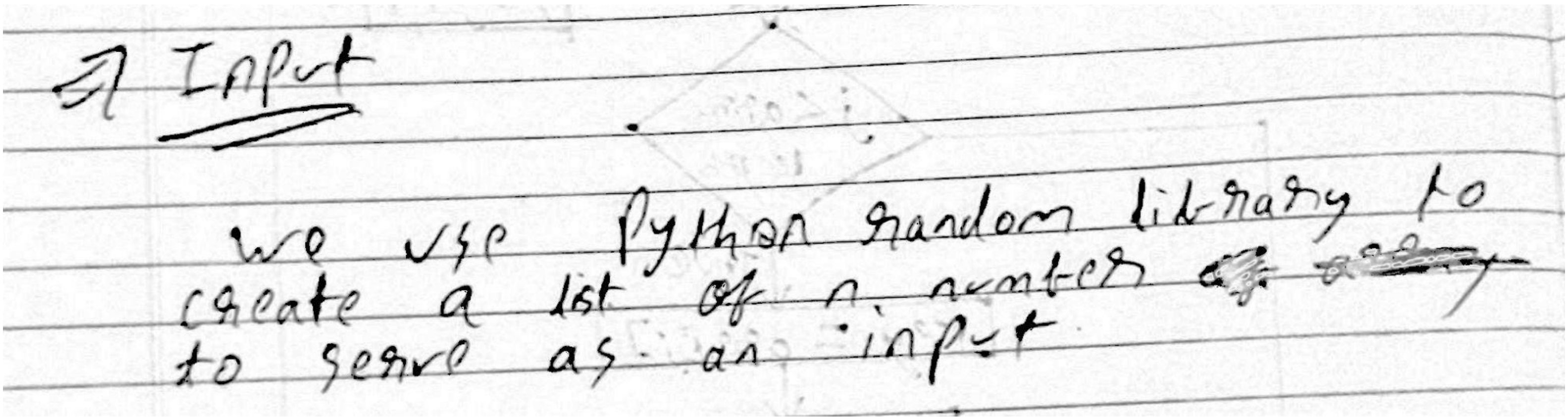
$$2 \cdot c(n-1)$$

$$2cn - 2c$$

$$\Rightarrow c_1 n$$

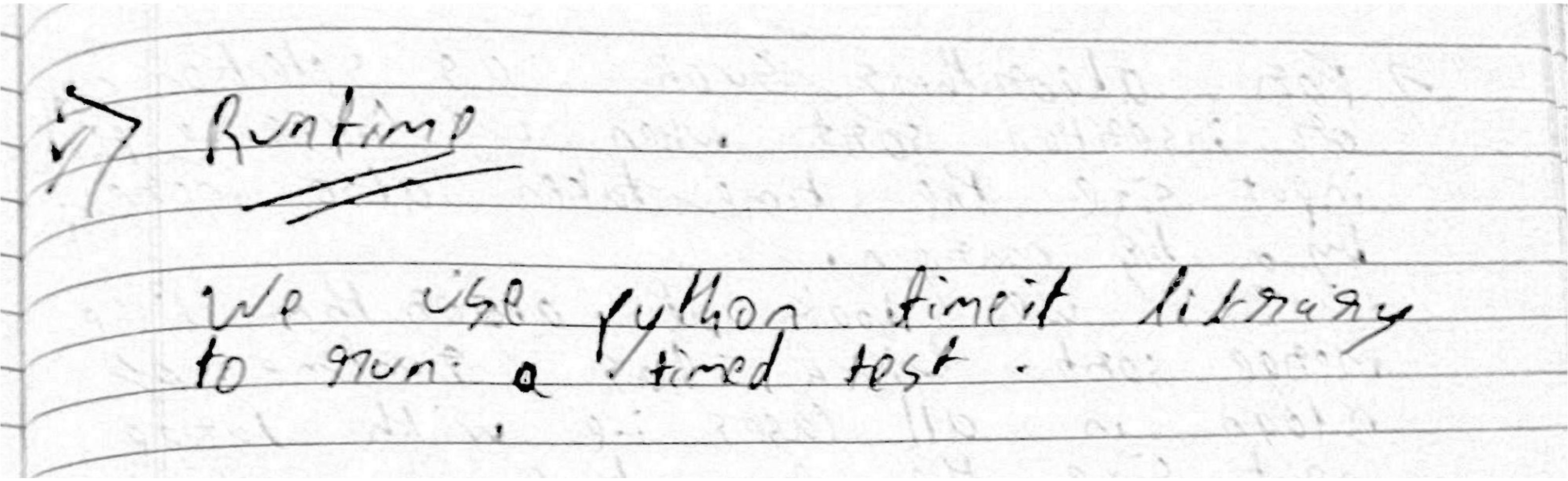
thus giving use the best case scene

$$O(n)$$



⇒ Input

We use python random library to create a list of n number to serve as an input



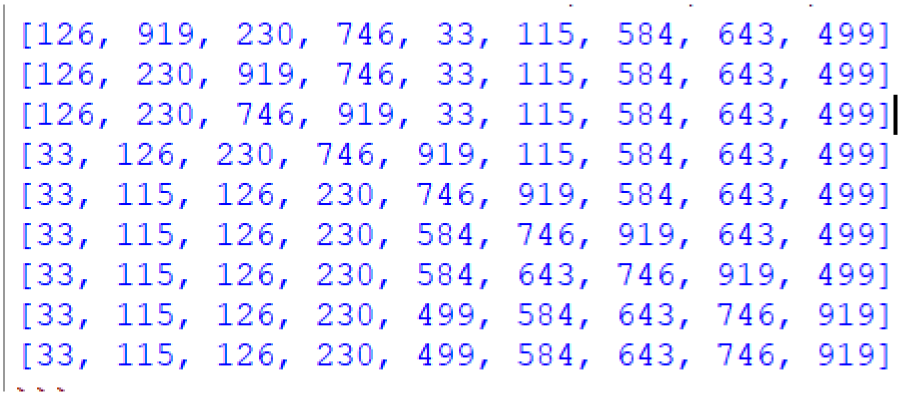
⇒ Runtime

We use python timeit library to run a timed test.

Program

```python
from functools import wraps
import timeit
import random

def insertion_sort(arr):
    # Traverse through 1 to len(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
    return arr


if __name__ == "__main__":
    arr=[]
    for i in range (1,10):
        n = random.randint(0,1000)
        arr.append(n)
    print(insertion_sort(arr))

```

For input size of 10

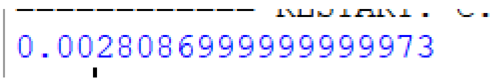


```
[126, 919, 230, 746, 33, 115, 584, 643, 499]
[126, 230, 919, 746, 33, 115, 584, 643, 499]
[126, 230, 746, 919, 33, 115, 584, 643, 499]
[33, 126, 230, 746, 919, 115, 584, 643, 499]
[33, 115, 126, 230, 746, 919, 584, 643, 499]
[33, 115, 126, 230, 584, 746, 919, 643, 499]
[33, 115, 126, 230, 584, 643, 746, 919, 499]
[33, 115, 126, 230, 499, 584, 643, 746, 919]
[33, 115, 126, 230, 499, 584, 643, 746, 919]
```

Timing code

```python
SETUP_CODE = '''
from __main__ import insertion_sort
import random
'''

TEST_CODE = '''
arr=[]
for i in range (1,5):
    n = random.randint(0,1000)
    arr.append(n)
insertion_sort(arr)'''

times = timeit.timeit(setup = SETUP_CODE,
                      stmt = TEST_CODE,
                      number = 100)

print(times)

```

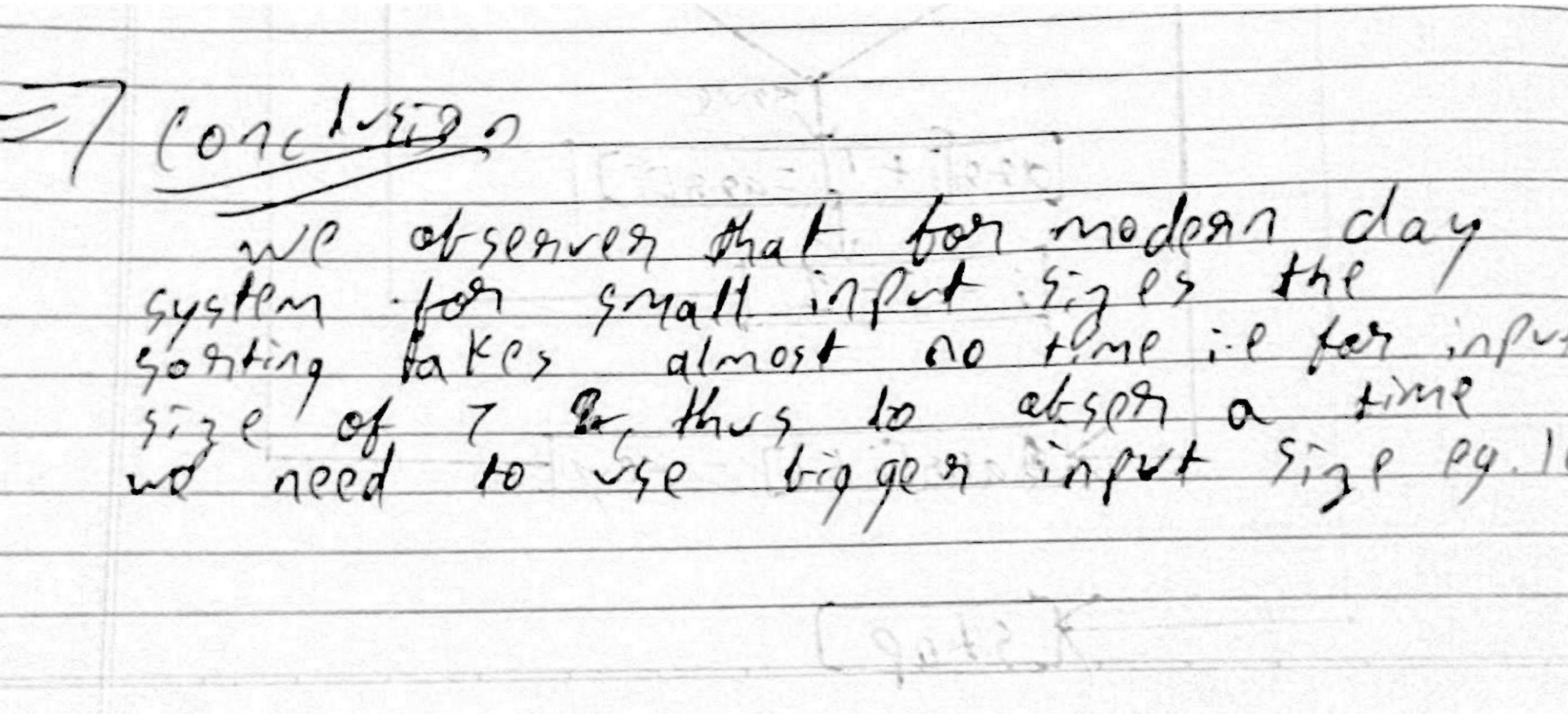
Output time with array of size 10



```
============ RESTART: C:
0.0028086999999999973
```

Output with array of size 100000

```
------------ |
469.4414816
```



=> Conclusion

we observer that for modern day system for small input sizes the sorting takes almost no time i.e for inpu size of 7, thus to obser a time we need to use bigger input size eg. 1