```
import math
import random as rn
import matplotlib.pyplot as plt
import timeit

n=[5,10,20,50,100,500,1000,5000,10000]

swapInsertion=[]
compareInsertion=[]
swapSelection=[]
compareSelection=[]
swapBubble=[]
compareBubble=[]
timeInsertion=[]
timeSelection=[]
timeBubble=[]

arr=[]
arr1=[]
arr2=[]
arr3=[]
a=0
a1=0

for a in range(5):
    element=rn.randint(0,1000000000)
    arr.append(element)

for a1 in range (5):
    arr1.append(arr[a1])
    arr2.append(arr[a1])
    arr3.append(arr[a1])

print("The unsorted array is ",arr)

#insertionsort
start = timeit.default_timer()
numComparisons=0
numSwaps=0
temp=0

for i in range(1,len(arr1)):
    temp=arr1[i]
    b=i-1


    while b >=0 and temp < arr1[b] :
        arr1[b+1] = arr1[b]
        b-= 1
        numComparisons+=1
```

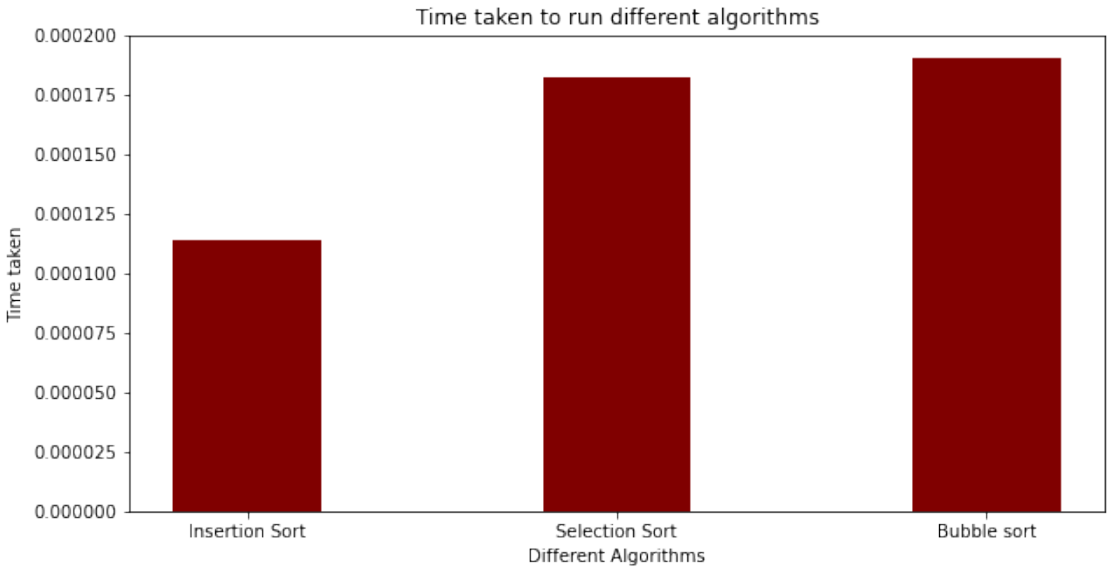
```
    arr1[b+1] = temp
    numSwaps=numSwaps+1

stop=timeit.default_timer()


swapInsertion.append(numSwaps)
compareInsertion.append(numComparisons)
timeInsertion.append(stop-start)


print("Number of swaps done in insertion sort ",numSwaps)
print("Number of comparisons done in insertion sort ",numComparisons)
print("Time for running insertion sort ",stop-start)

#selectionsort
start = timeit.default_timer()
numComparisons=0
numSwaps=0
temp=0

for i in range(len(arr2)):
    min=i
    for k in range(i,len(arr2)):
        numComparisons=numComparisons+1
        if arr2[min] > arr2[k]:
            min=k
    temp=arr2[min]
    arr2[min]=arr2[i]
    arr2[i]=temp
    numSwaps=numSwaps+1


stop=timeit.default_timer()


swapSelection.append(numSwaps)
compareSelection.append(numComparisons)
timeSelection.append(stop-start)

print("Number of swaps done in selection sort ",numSwaps)
print("Number of comparisons done in selection sort ",numComparisons)
print("Time for running Selection sort ",stop-start)


#bubblesort
start = timeit.default_timer()
numComparisons=0
numSwaps=0
```

```
temp=0

for i in range(len(arr3)):
    for j in range(len(arr3)-i-1):
        numComparisons=numComparisons+1
        if arr3[j]>arr3[j+1]:
            temp=arr3[j]
            arr3[j]=arr3[j+1]
            arr3[j+1]=temp
            numSwaps=numSwaps+1
stop=timeit.default_timer()

swapBubble.append(numSwaps)
compareBubble.append(numComparisons)
timeBubble.append(stop-start)

print("Number of swaps done in bubble sort ",numSwaps)
print("Number of comparisons done in bubble sort ",numComparisons)
print("Time for running bubble sort ",stop-start)
```

```
The unsorted array is  [245735858, 486477523, 970767390, 490948199,
984604767]
Number of swaps done in insertion sort  4
Number of comparisons done in insertion sort  1
Time for running insertion sort  0.0001140000003942987
Number of swaps done in selection sort  5
Number of comparisons done in selection sort  15
Time for running Selection sort  0.00018259999887959566
Number of swaps done in bubble sort  1
Number of comparisons done in bubble sort  10
Time for running bubble sort  0.00019049999900744297
```

```
import numpy as np
import matplotlib.pyplot as plt


data = {'Insertion Sort':timeInsertion[0], 'Selection
Sort':timeSelection[0], 'Bubble sort':timeBubble[0],}

algorithm = list(data.keys())
time = list(data.values())

fig = plt.figure(figsize = (10, 5))


plt.bar(algorithm, time, color ='maroon',
        width = 0.4)

plt.xlabel("Different Algorithms")
```

```
plt.ylabel("Time taken")
plt.title("Time taken to run different algorithms")
plt.show()
```



Report

The efficiency of the algoriths can be explained in two ways. Based on the number of swaps, comparisions, shifts and it can also be explained by looking directly at the time taken to run the algorithm

Based on number of operations

Operations typically comprise of 3 steps.

1) Shifting 2) Swapping 3) comparing

If the time taken to do compariing is x, then typically the time taken for shifting is 1.5x and time taken for swapping is 2.5x.

In bubble sort there is a lot of swapping and comparing and hence it is the slowest among the 3. In selection sort, it has comparatively fewer number of swaps than bubble sort and almost the same as insertion sort but has more comparisions and shifting hence it is the 2nd fastest algorithm In insertion sort there is the least number of swapping, comparision and shifting and hece it is the fastest algorithm

Based on time taken