Name: Krithika Swaminthan
Reg. no.: 205001057

DAA Lab – Assignment 1

Bubble Sort

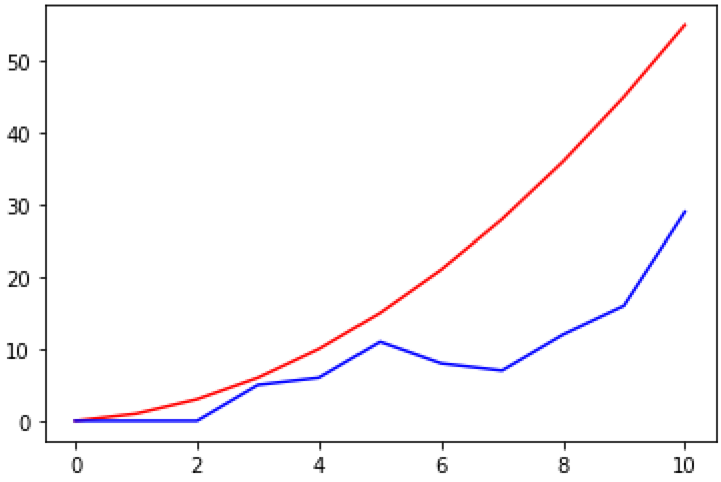
1. Measure the number of Comparisons and swap for the program and plot a chart.

Code:

```
import random
from matplotlib import pyplot as plt
import numpy as np

#function for bubble sort
def bubbleSort(arr):
      n = len(arr)
      comp = 0
      swap = 0
      for i in range(n):
            for j in range(0, n-i-1):
                  comp += 1
                  if arr[j] > arr[j+1] :
                        swap += 1
                        arr[j], arr[j+1] = arr[j+1], arr[j]
      return comp, swap

n = 10
comps, swaps, nums = [], [], []
for i in range(n+1):
      nums.append(i)
      arr = []
      for j in range(i+1):
            arr.append(random.randint(1,100))

      #print("Original array:",arr)
      #calling bubble sort
      comp, swap = bubbleSort(arr)
      #print("Sorted array is:",arr)

      #print("No. of comparisons:",comp)
      comps.append(comp)

      #print("No. of swaps:",swap)
      swaps.append(swap)

xpoints = np.array(nums)
ypoints = np.array(comps)
zpoints = np.array(swaps)
```

```
plt.plot(xpoints,ypoints,color='r')
plt.plot(xpoints,zpoints,color='b')
plt.show()
```

Output:

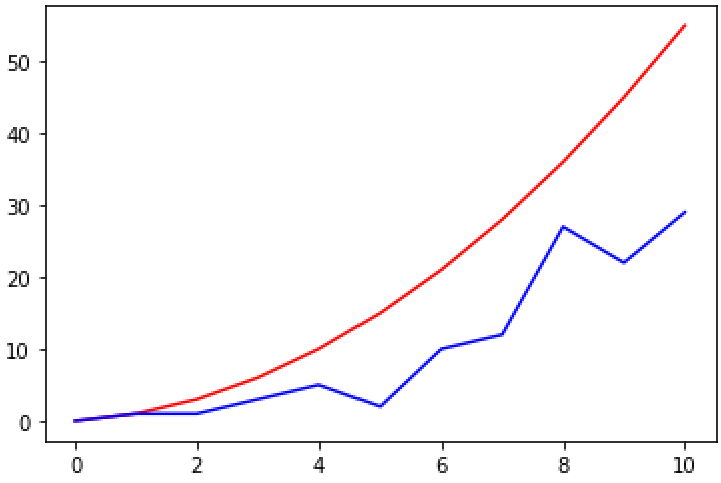


2. Modify the program to have best case Efficiency.

Code:

```
import random
from matplotlib import pyplot as plt
import numpy as np

#function for bubble sort
def bubbleSort(arr):
    n = len(arr)
    comp = 0
    swap = 0

    #modifying the code to accommodate best case efficiency
    flag = 0
    for i in range(n-1):
        if arr[i] > arr[i+1]:
            flag = 1
    if flag == 0:
        return n-1, 0


    for i in range(n):
        for j in range(0, n-i-1):
            comp += 1
            if arr[j] > arr[j+1] :
                swap += 1
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return comp, swap

n = 10
comps, swaps, nums = [], [], []
for i in range(n+1):
    nums.append(i)
    arr = []
    for j in range(i+1):
        arr.append(random.randint(1,100))

    #print("Original array:",arr)
    #calling bubble sort
    comp, swap = bubbleSort(arr)
    #print("Sorted array is:",arr)

    #print("No. of comparisons:",comp)
    comps.append(comp)

    #print("No. of swaps:",swap)
    swaps.append(swap)

xpoints = np.array(nums)
ypoints = np.array(comps)
zpoints = np.array(swaps)

plt.plot(xpoints,ypoints,color='r')
plt.plot(xpoints,zpoints,color='b')
plt.show()
```

Output:



Name: Krithika Swaminthan
Reg. no.: 205001057

3. Check whether the algorithm has the 2 properties.

- The algorithm has well-defined inputs and reaches termination after a finite number of steps.

4. Implement recursive bubble sort.

Code:

```
import random
from matplotlib import pyplot as plt
import numpy as np

def bubbleSortRecursive(arr,newarr):
      n = len(arr)

      for i in range(n-1):
            if arr[i] > arr[i + 1]:
                  arr[i], arr[i+1] = arr[i+1], arr[i]

      newarr.insert(0,arr[-1])

      # base case
      if n == 1:
            return newarr
      # largest element is fixed, recur for remaining array
            return bubbleSortRecursive(arr[:-1],newarr)

#driver code for recursive bubble sort
n = 10
for i in range(n+1):
arr = []
for j in range(i+1):
arr.append(random.randint(1,100))

print("Original array:",arr)

newarr = bubbleSortRecursive(arr,[])

print("Sorted array:",newarr)
```

Output:

```
Original array: [26]
Sorted array: [26]
Original array: [32, 13]
Sorted array: [13, 32]
```

Name: Krithika Swaminthan
Reg. no.: 205001057

```
Original array: [45, 33, 17]
Sorted array: [17, 33, 45]
Original array: [17, 48, 6, 27]
Sorted array: [6, 17, 27, 48]
Original array: [88, 72, 55, 68, 90]
Sorted array: [55, 68, 72, 88, 90]
Original array: [64, 57, 50, 61, 98, 23]
Sorted array: [23, 50, 57, 61, 64, 98]
Original array: [6, 92, 3, 21, 95, 100, 24]
Sorted array: [3, 6, 21, 24, 92, 95, 100]
Original array: [43, 97, 23, 59, 67, 8, 20, 76]
Sorted array: [8, 20, 23, 43, 59, 67, 76, 97]
Original array: [96, 97, 53, 94, 6, 2, 1, 21, 88]
Sorted array: [1, 2, 6, 21, 53, 88, 94, 96, 97]
Original array: [57, 1, 53, 33, 32, 35, 14, 9, 35, 25]
Sorted array: [1, 9, 14, 25, 32, 33, 35, 35, 53, 57]
Original array: [39, 77, 86, 57, 93, 95, 33, 43, 26, 24, 75]
Sorted array: [24, 26, 33, 39, 43, 57, 75, 77, 86, 93, 95]
```

Name: Krithika Swaminthan
Reg. no.: 205001057

Insertion Sort

1. Measure the number of Comparisons and swap for the program and plot a chart.

Code:

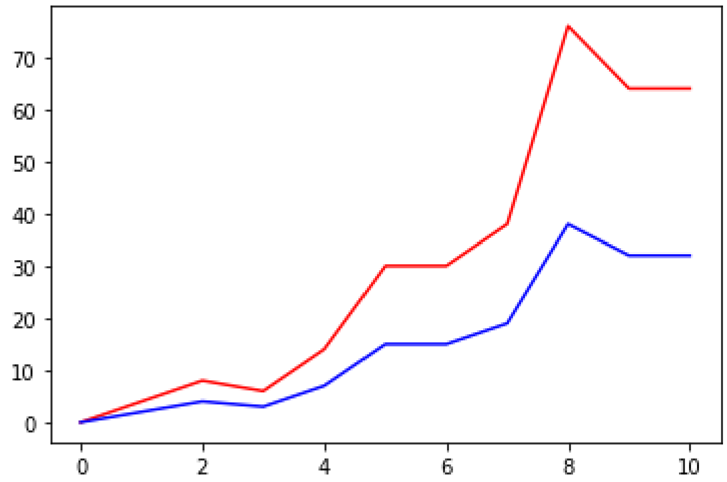
```
import random
from matplotlib import pyplot as plt
import numpy as np

#function for insertion sort
def insertionSort(arr):
    n = len(arr)
    comp, swap = 0, 0
    for i in range(1,len(arr)):
        el = arr[i]
        j = i-1
        while (j>=0 and el<arr[j]):
            comp += 2
            arr[j+1] = arr[j]
            j-=1
            swap += 1
        comp += 2
        arr[j+1] = el
        swap += 1
    return comp, swap

n = 10
comps, swaps, nums = [], [], []
for i in range(n+1):
    nums.append(i)
    arr = []
    for j in range(i+1):
        arr.append(random.randint(1,100))

    print("Original array:",arr)
    comp, swap = insertionSort(arr)
    print("Sorted array is:",arr)

    print("No. of comparisons:",comp)
    comps.append(comp)

    print("No. of swaps:",swap)
    swaps.append(swap)

xpoints = np.array(nums)
ypoints = np.array(comps)
zpoints = np.array(swaps)
```

```
plt.plot(xpoints,ypoints,color='r')
plt.plot(xpoints,zpoints,color='b')
plt.show()
```

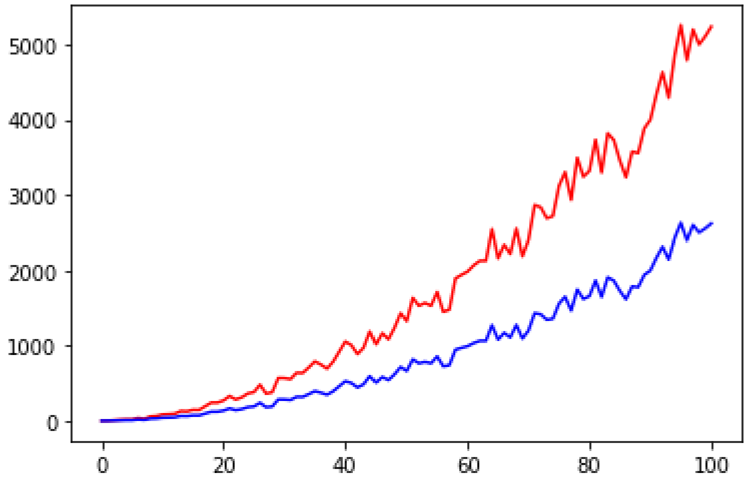
Output:

```
Original array: [45]
Sorted array is: [45]
No. of comparisons: 0
No. of swaps: 0
Original array: [90, 27]
Sorted array is: [27, 90]
No. of comparisons: 4
No. of swaps: 2
Original array: [100, 8, 84]
Sorted array is: [8, 84, 100]
No. of comparisons: 8
No. of swaps: 4
Original array: [15, 76, 82, 86]
Sorted array is: [15, 76, 82, 86]
No. of comparisons: 6
No. of swaps: 3
Original array: [90, 1, 8, 46, 99]
Sorted array is: [1, 8, 46, 90, 99]
No. of comparisons: 14
No. of swaps: 7
Original array: [59, 62, 37, 89, 20, 33]
Sorted array is: [20, 33, 37, 59, 62, 89]
No. of comparisons: 30
No. of swaps: 15
Original array: [53, 18, 23, 63, 78, 74, 16]
Sorted array is: [16, 18, 23, 53, 63, 74, 78]
No. of comparisons: 30
No. of swaps: 15
Original array: [78, 10, 30, 10, 11, 52, 16, 34]
Sorted array is: [10, 10, 11, 16, 30, 34, 52, 78]
No. of comparisons: 38
No. of swaps: 19
Original array: [54, 85, 72, 51, 77, 27, 32, 21, 14]
Sorted array is: [14, 21, 27, 32, 51, 54, 72, 77, 85]
No. of comparisons: 76
No. of swaps: 38
Original array: [78, 38, 67, 82, 24, 5, 1, 55, 94, 77]
Sorted array is: [1, 5, 24, 38, 55, 67, 77, 78, 82, 94]
No. of comparisons: 64
No. of swaps: 32
Original array: [54, 32, 8, 86, 27, 21, 82, 64, 95, 75, 40]
Sorted array is: [8, 21, 27, 32, 40, 54, 64, 75, 82, 86, 95]
No. of comparisons: 64
No. of swaps: 32
```

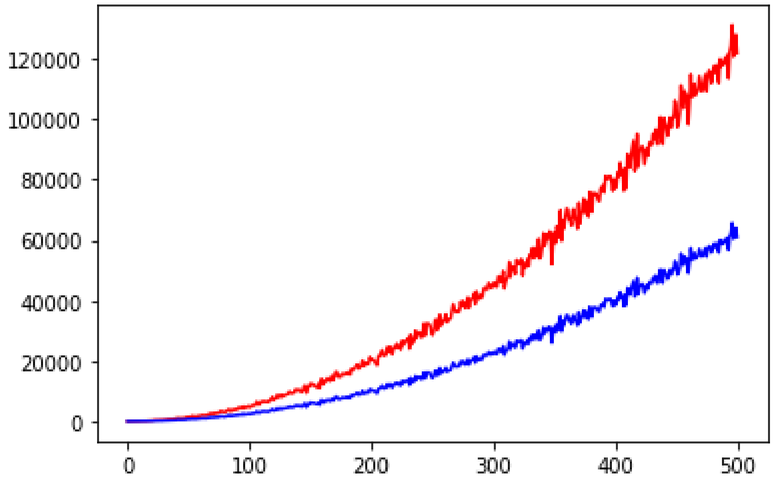
Name: Krithika Swaminthan
Reg. no.: 205001057



For n = 100,



For n = 500,



Name: Krithika Swaminthan
Reg. no.: 205001057

2. Modify the program to have best case Efficiency. Check whether the algorithm has the 2 properties.

Code:

```
import random
from matplotlib import pyplot as plt
import numpy as np

#function for insertion sort
def insertionSort(arr):
      n = len(arr)
      comp, swap = 0, 0

      #modifying the code to accommodate best case efficiency
      flag = 0
      for i in range(n-1):
            if arr[i] > arr[i+1]:
                  flag = 1
      if flag == 0:
            return n-1, 0
      for i in range(1,len(arr)):
            el = arr[i]
            j = i-1
            while (j>=0 and el<arr[j]):
                  comp += 2
                  arr[j+1] = arr[j]
                  j-=1
                  swap += 1
            comp += 2
            arr[j+1] = el
            swap += 1
      return comp, swap

n = 10
for i in range(n+1):
      arr = []
      for j in range(i+1):
            arr.append(random.randint(1,100))

      print("Original array:",arr)
      comp, swap = insertionSort(arr)
      print("Sorted array is:",arr)

      print("No. of comparisons:",comp)
      print("No. of swaps:",swap)
```

- The algorithm has well-defined inputs and reaches termination after a finite number of steps.