Solution Document

**Question 1**

1. Filename –Q1\_Insertion.py

Method to run file-

>>python Q1\_Insertion.py

E.g.

Enter the max size 5

The unsorted list [3, 2, 2, 1, 5]

Sorted list [1, 2, 2, 3, 5]

Time 1.839226771955983e-05

1. Filename –Q1\_Quicksort.py

Method to run file-

>>python Q1\_Quicksort.py

E.g.

Enter the max size 5

The unsorted list [-4, 0, 3, -5, -3]

Sorted list [-5, -4, -3, 0, 3]

Time 2.138635781344167e-05

1. Filename – Q1\_RandomQS.py

Method to run file-python RandomQS.py

E.g.

Enter the max size 5

The unsorted list [-3, -2, -4, -5, 2]

Sorted list [-5, -4, -3, -2, 2]

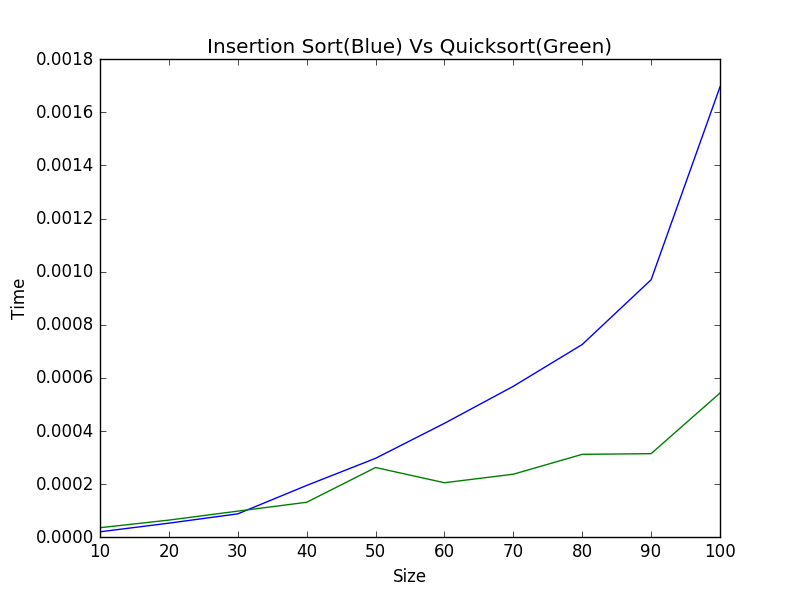
Time 3.8495444064195e-05

**Q1 (A)**

Soln.

Run the file Q1\_ISvsQS.py from command prompt to generate the Time vs Size graph for Insertion vs QuickSort by following the step below:

>>python Q1\_ISvsQS.py



|  |  |  |
| --- | --- | --- |
| **Size** | **InsertionSort(Time)** | **QuickSort(Time)** |
| 10 | 2.05E-05 | 3.59E-05 |
| 20 | 5.30E-05 | 6.46E-05 |
| 30 | 8.81E-05 | 9.84E-05 |
| 40 | 0.000195044 | 1.32E-04 |
| 50 | 0.00029727 | 0.000262624 |
| 60 | 0.00042901 | 2.05E-04 |
| 70 | 0.000568449 | 2.37E-04 |
| 80 | 0.000725853 | 3.12E-04 |
| 90 | 0.000969657 | 3.15E-04 |
| 100 | 0.001695938 | 5.43E-04 |

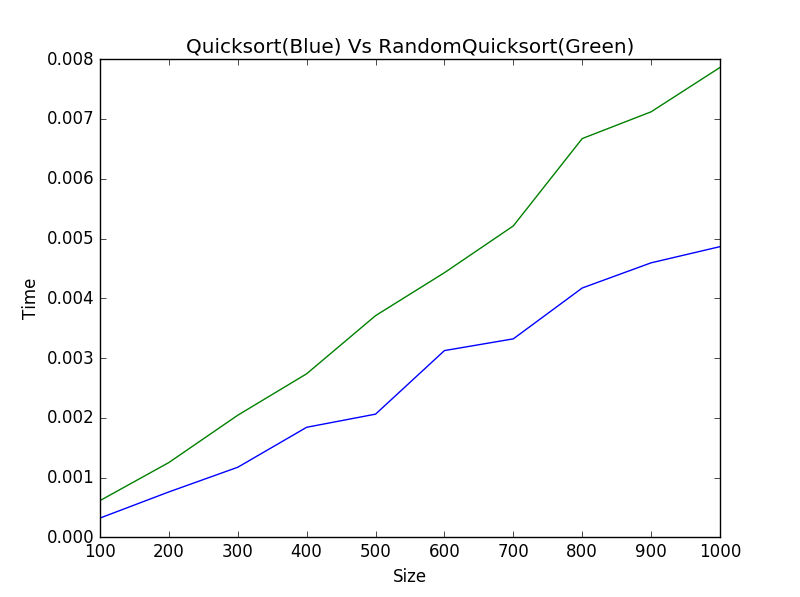
Q1\_A\_Graph.png represents the graph for running time of Insertion and Quick sort. We can observe from the graphs that at the input size of around 30 the Insertion Sort overtakes Quick Sort in terms of time. From this we can conclude that Quick Sort becomes more efficient than Insertion Sort as Size increases.

**Q1 (B)**

Soln.

Run the file Q1\_QSvsRQS.py from command prompt to generate the Time vs Size graph for QuickSort vs RandomQuickSort by following the step below:

>>python Q1\_QSvsRQS.py



|  |  |  |
| --- | --- | --- |
| **Size** | **QuickSort(Time)** | **RandomQuickSort(Time)** |
| 100 | 0.000321223 | 0.000613361 |
| 200 | 0.000758788 | 0.001250674 |
| 300 | 0.001171545 | 0.002042397 |
| 400 | 0.00184051 | 0.002736171 |
| 500 | 0.002060362 | 0.003708822 |
| 600 | 0.003124119 | 0.004427832 |
| 700 | 0.003320446 | 0.005211 |
| 800 | 0.004171623 | 0.006672544 |
| 900 | 0.004592934 | 0.007120374 |
| 1000 | 0.004863685493932901 | 0.007862908 |

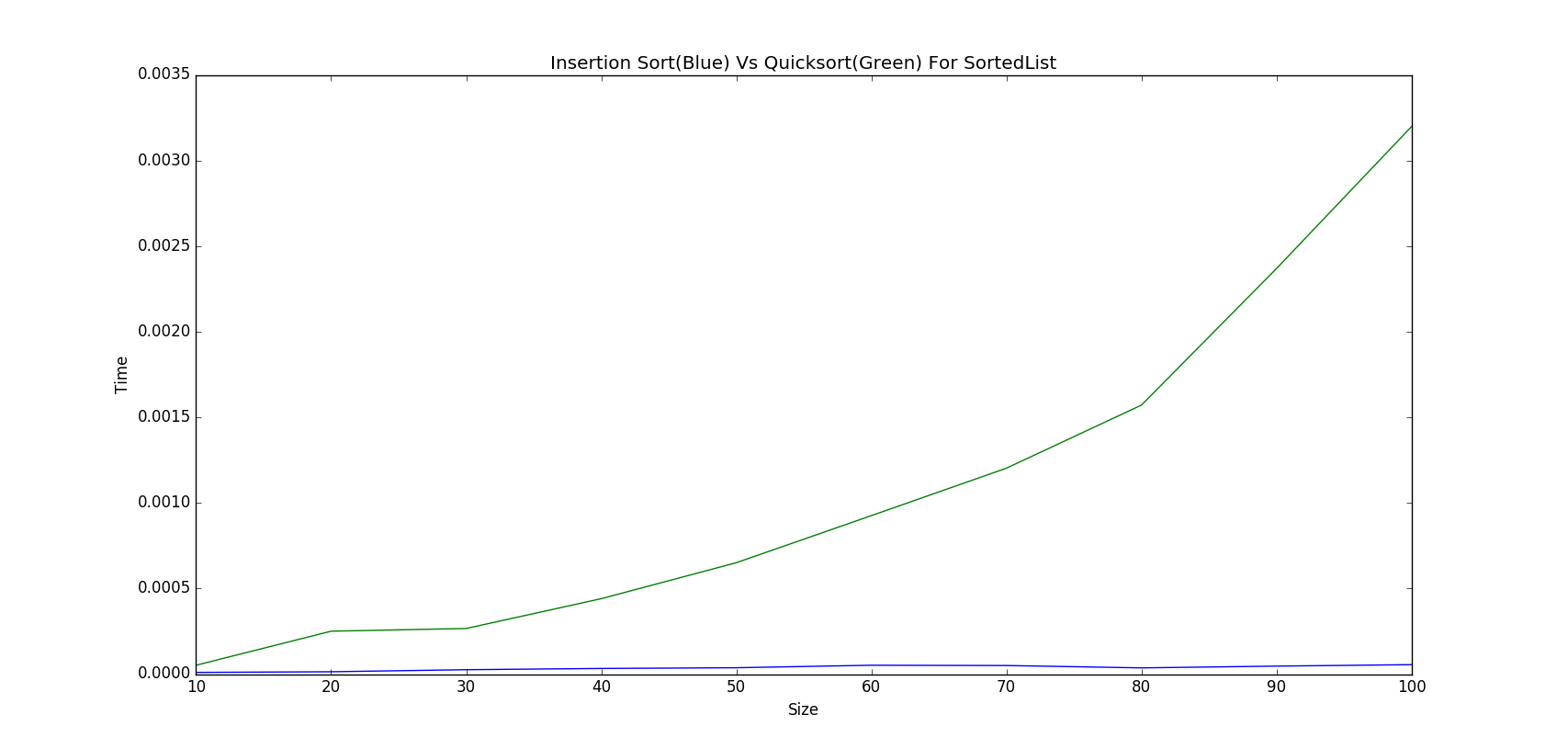
Q1\_B\_Graph.png represents the graph for running time of Quick sort and Randomize Quick sort. We can interpret from the graphs that running time of both the algorithms show somewhat similar behavior with Quick sort having less running time as compared to RandomQuicksort with increase in Size due to overhead of calculating Random Pivot in RandomQuickSort . Therefore both of the algorithms seem to be feasible for a random list as input though Quicksort seems efficient over Random Sort.

**Q1 (C)**

Soln.

Run the file Q1\_QSvsRQS.py from command prompt to generate the Time vs Size graph for QuickSort vs RandomQuickSort by following the step below:

>>python Q1\_QSvsISforSortedList.py



|  |  |  |
| --- | --- | --- |
| **Size** | **InsertionSort(Time)** | **QuickSort(Time)** |
| 10 | 9.84E-06 | 5.18E-05 |
| 20 | 1.37E-05 | 0.000250648 |
| 30 | 2.57E-05 | 0.000266474 |
| 40 | 3.29E-05 | 0.000440559 |
| 50 | 3.72E-05 | 0.000650573 |
| 60 | 5.18E-05 | 0.000925602 |
| 70 | 5.00E-05 | 0.001202769 |
| 80 | 3.64E-05 | 0.00157147 |
| 90 | 4.66E-05 | 0.002369181 |
| 100 | 5.52E-05 | 0.003198544 |

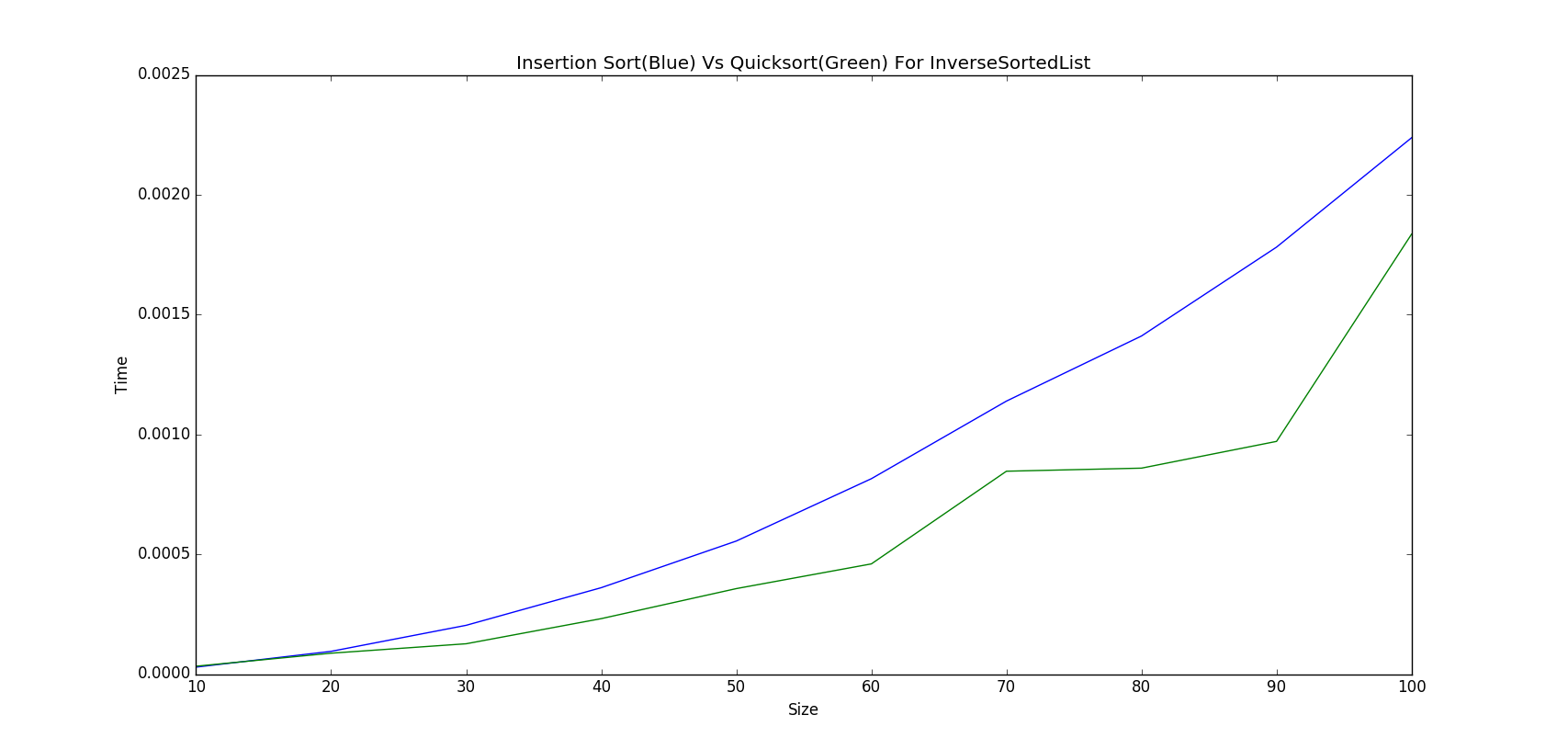
Q1\_C\_Graph.png represents the graph for running time of Insertion and Quick sort for a sorted input list. We can see from the graphs that the two graphs are deviating from each other with running time of Quick Sort increasing exponentially with increase in input size whereas Insertion sort shows somewhat similar running times with increase in input size for a sorted list. Thus we can conclude that Insertion Sort is better than Quick Sort for an already sorted list.

**Q1 (D)**

Soln.

Run the file Q1\_ISvsQS\_ReverseSort.py from command prompt to generate the Time vs Size graph for QuickSort vs RandomQuickSort by following the step below:

>>python Q1\_\_ISvsQS\_ReverseSort.py



|  |  |  |
| --- | --- | --- |
| **Size** | **InsertionSort(Time)** | **QuickSort(Time)** |
| 10 | 2.91E-05 | 3.29E-05 |
| 20 | 9.50E-05 | 8.73E-05 |
| 30 | 0.000203598 | 0.000126607 |
| 40 | 0.000360574 | 0.0002314 |
| 50 | 0.00055519 | 0.000356724 |
| 60 | 0.000815248 | 0.000459807 |
| 70 | 0.001139037 | 0.000846472 |
| 80 | 0.001410644 | 0.000859304 |
| 90 | 0.001781484 | 0.000970941 |
| 100 | 0.002237868 | 0.001835805 |

Q1\_D\_Graph.png represents the graph for running time of Insertion and Quick sort for an inversely Sorted list. We can observe from the graph that the Insertion Sort overtakes Quick Sort in terms of running time with increase in input size for an inversely sorted list. Thus we can conclude that Quick Sort works more efficiently than Insertion Sort for an Inversely sorted list with increase in input size.

**Question 2**

Filename –Greedy.py

Method to run file-

>>python Greedy.py

e.g. enter number of categories:5

2

2

1

1

1

[2, 2, 1, 1, 1]

The GQ is : 2

Expected output – Maximum Greed Quotient