Computer Science 717 – (2022)

Programming Assignment 1

Due: Sunday, March 27 2022 (11:59pm)

1 Implementing selection sort, insertion sort, merge sort and quick sort in ascending order (6 pts)

Write a program with required components:

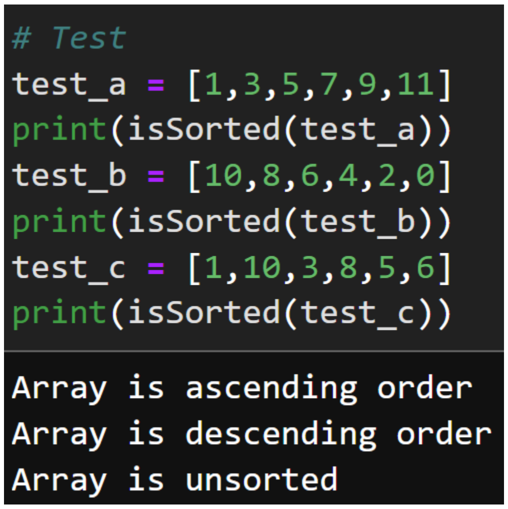
1. Given an array A of n real values, verify that A is sorted or unsorted in ascending order. What is the running time complexity of your verifying algorithm in term of big-O notation in n? Explain your answer.

ANSWER:

For the python program, please refer to the program file Part I: 1.1 Verify algorithm.

When the length of the array A is "n", the first "if" statement will compare all elements in A, for a total of "n-1" times. If the condition is met, it will return, otherwise the next " elif" statement will be executed. The "elif" statement also compares all elements in A, for a total of "n-1" times. If the condition is met, it will return, otherwise the last "else" statement will be executed. So the time complexity of this algorithm is O(n).

The figure below shows the three different arrays used to test the verify algorithm and the output validation results.



```
# Test
test_a = [1,3,5,7,9,11]
print(isSorted(test_a))
test_b = [10,8,6,4,2,0]
print(isSorted(test_b))
test_c = [1,10,3,8,5,6]
print(isSorted(test_c))
```

```
Array is ascending order
Array is descending order
Array is unsorted
```

2. Implement sorting algorithms: selection sort, insertion sort, merge sort and quick sort. For each sorting algorithms, output the number of comparisons used.

3. Given a parameter n, generate an array of n random real values in [0, 1]. Run the 4 implemented sorting algorithms. For each algorithm, you need to (1) verify the sorted

output, (2) report the number of comparisons, and (3) report the running time in milliseconds.

ANSWER:

I combined question 2 and question 3 together to complete.

For the python program, please refer to the program file:

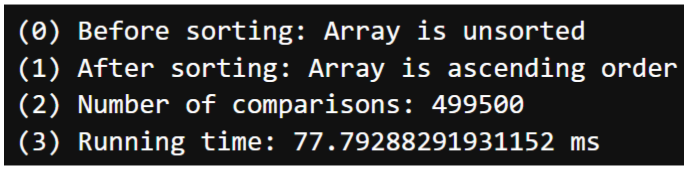
Part I: 2.1 Selection Sort

Part I: 2.2 Insertion Sort

Part I: 2.3 Merge Sort

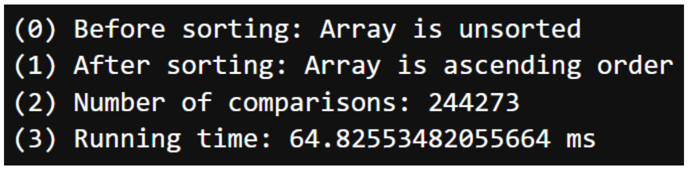
Part I: 2.4 Quick Sort

Regarding the parameter n in question 3, I assume n=1000 in the program to facilitate testing and output results. Here are the results of the four algorithm outputs. (Note: Each time you rerun the program, the output will be slightly different, but within a relatively reasonable range.)



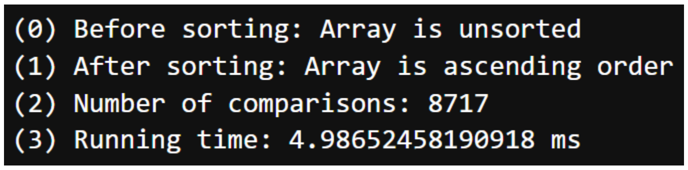
```
(0) Before sorting: Array is unsorted
(1) After sorting: Array is ascending order
(2) Number of comparisons: 499500
(3) Running time: 77.79288291931152 ms
```

Part I: 2.1 Selection Sort



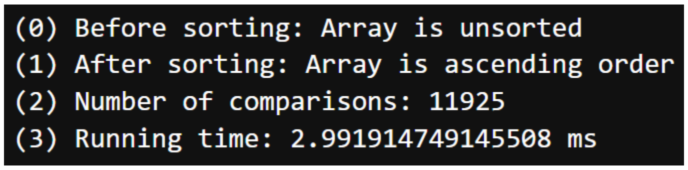
```
(0) Before sorting: Array is unsorted
(1) After sorting: Array is ascending order
(2) Number of comparisons: 244273
(3) Running time: 64.82553482055664 ms
```

Part I: 2.2 Insertion Sort



```
(0) Before sorting: Array is unsorted
(1) After sorting: Array is ascending order
(2) Number of comparisons: 8717
(3) Running time: 4.98652458190918 ms
```

Part I: 2.3 Merge Sort



```
(0) Before sorting: Array is unsorted
(1) After sorting: Array is ascending order
(2) Number of comparisons: 11925
(3) Running time: 2.991914749145508 ms
```

Part I: 2.4 Quick Sort

It can be seen from the above results that, under the same data size, selection sort requires the most comparisons and has the longest running time, followed by insertion sort. In contrast, merge sort and quick sort require fewer comparisons and running time.

Using your written program to execute the following tasks.

1. We test the efficiency and accuracy of our implementations as follows. For n ∈ {10^3, 10^4, . . . , 10^6}, generate an array of n random values in [0, 1]. Run the implementation of 4 sorting algorithms above, and for each algorithm, return the number of comparisons and the running time.

ANSWER:

For the python program, please refer to the program file:

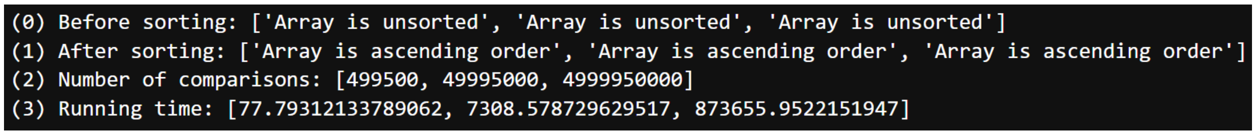
Part II: 1.1 Selection Sort

Part II: 1.2 Insertion Sort

Part II: 1.3 Merge Sort

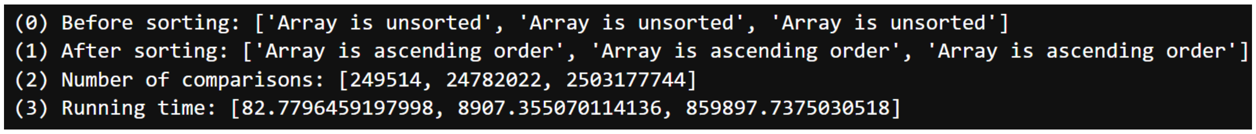
Part II: 1.4 Quick Sort

The time complexity of selection sort and insertion sort are both O(n^2), which means that as the data size increases, the running time required for both algorithms will increase exponentially. In the process of actually running the program, I found that due to the limitation of computer performance, for selection sort and insertion sort, I had to set n up to 10^5, otherwise the system would risk crashing, therefore, you can see there are only three output results for selection sort and insertion sort. Here are the results of the four algorithm outputs.



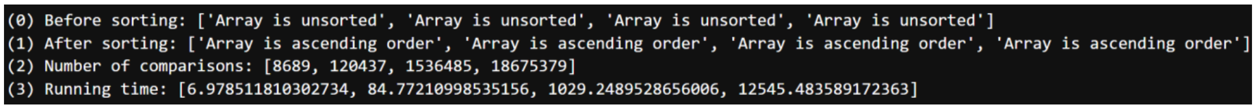
```
(0) Before sorting: ['Array is unsorted', 'Array is unsorted', 'Array is unsorted']
(1) After sorting: ['Array is ascending order', 'Array is ascending order', 'Array is ascending order']
(2) Number of comparisons: [499500, 49995000, 4999950000]
(3) Running time: [77.79312133789062, 7308.578729629517, 873655.9522151947]
```

Part II: 1.1 Selection Sort



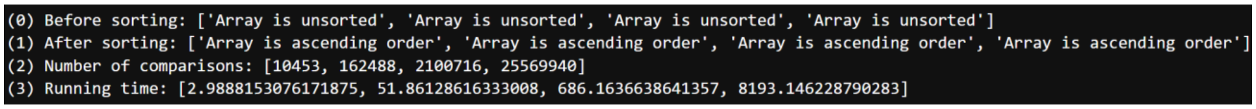
```
(0) Before sorting: ['Array is unsorted', 'Array is unsorted', 'Array is unsorted']
(1) After sorting: ['Array is ascending order', 'Array is ascending order', 'Array is ascending order']
(2) Number of comparisons: [249514, 24782022, 2503177744]
(3) Running time: [82.7796459197998, 8907.355070114136, 859897.7375030518]
```

Part II: 1.2 Insertion Sort



```
(0) Before sorting: ['Array is unsorted', 'Array is unsorted', 'Array is unsorted', 'Array is unsorted']
(1) After sorting: ['Array is ascending order', 'Array is ascending order', 'Array is ascending order', 'Array is ascending order']
(2) Number of comparisons: [8689, 120437, 1536485, 18675379]
(3) Running time: [6.978511810302734, 84.77210998535156, 1029.2489528656006, 12545.483589172363]
```

Part II: 1.3 Merge Sort



```
(0) Before sorting: ['Array is unsorted', 'Array is unsorted', 'Array is unsorted', 'Array is unsorted']
(1) After sorting: ['Array is ascending order', 'Array is ascending order', 'Array is ascending order', 'Array is ascending order']
(2) Number of comparisons: [10453, 162488, 2100716, 25569940]
(3) Running time: [2.9888153076171875, 51.86128616333008, 686.1636638641357, 8193.146228790283]
```

Part II: 1.4 Quick Sort

It can be seen from the above results that even if n=10^6 has been removed, the average running time and number of comparisons of selection sort and insertion sort is very exaggerated. This also illustrates the importance of a good algorithm for computational efficiency.

2. Repeat Step 1 10 times to get the average of number of comparisons and the average running time for each sorting algorithm.

ANSWER:

For the python program, please refer to the program file:

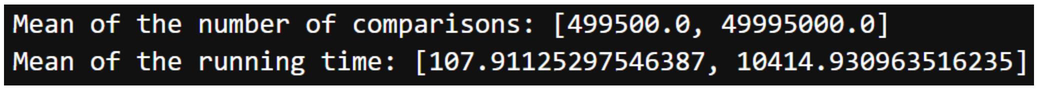
Part II: 2.1 Selection Sort

Part II: 2.2 Insertion Sort

Part II: 2.3 Merge Sort

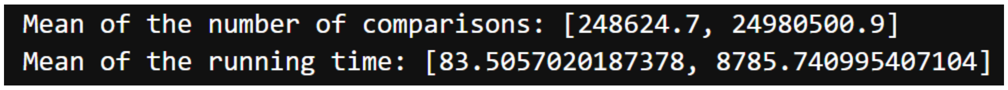
Part II: 2.4 Quick Sort

In the case of repeating 10 times, for selection sort and insertion sort, the amount of computation brought by n=10^5 is still unbearable for my computer, so I have to remove n=10^5 and only keep n=10^3 and n=10^4 to run the next program.



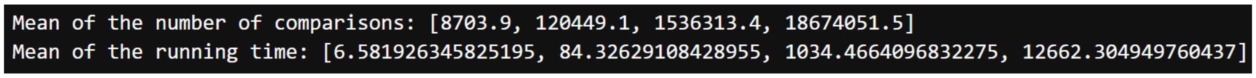
```
Mean of the number of comparisons: [499500.0, 49995000.0]
Mean of the running time: [107.91125297546387, 10414.930963516235]
```

Part II: 2.1 Selection Sort



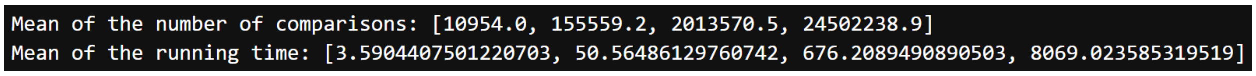
```
Mean of the number of comparisons: [248624.7, 24980500.9]
Mean of the running time: [83.5057020187378, 8785.740995407104]
```

Part II: 2.2 Insertion Sort



```
Mean of the number of comparisons: [8703.9, 120449.1, 1536313.4, 18674051.5]
Mean of the running time: [6.581926345825195, 84.32629108428955, 1034.4664096832275, 12662.304949760437]
```

Part II: 2.3 Merge Sort



```
Mean of the number of comparisons: [10954.0, 155559.2, 2013570.5, 24502238.9]
Mean of the running time: [3.5904407501220703, 50.56486129760742, 676.2089490890503, 8069.023585319519]
```

Part II: 2.4 Quick Sort

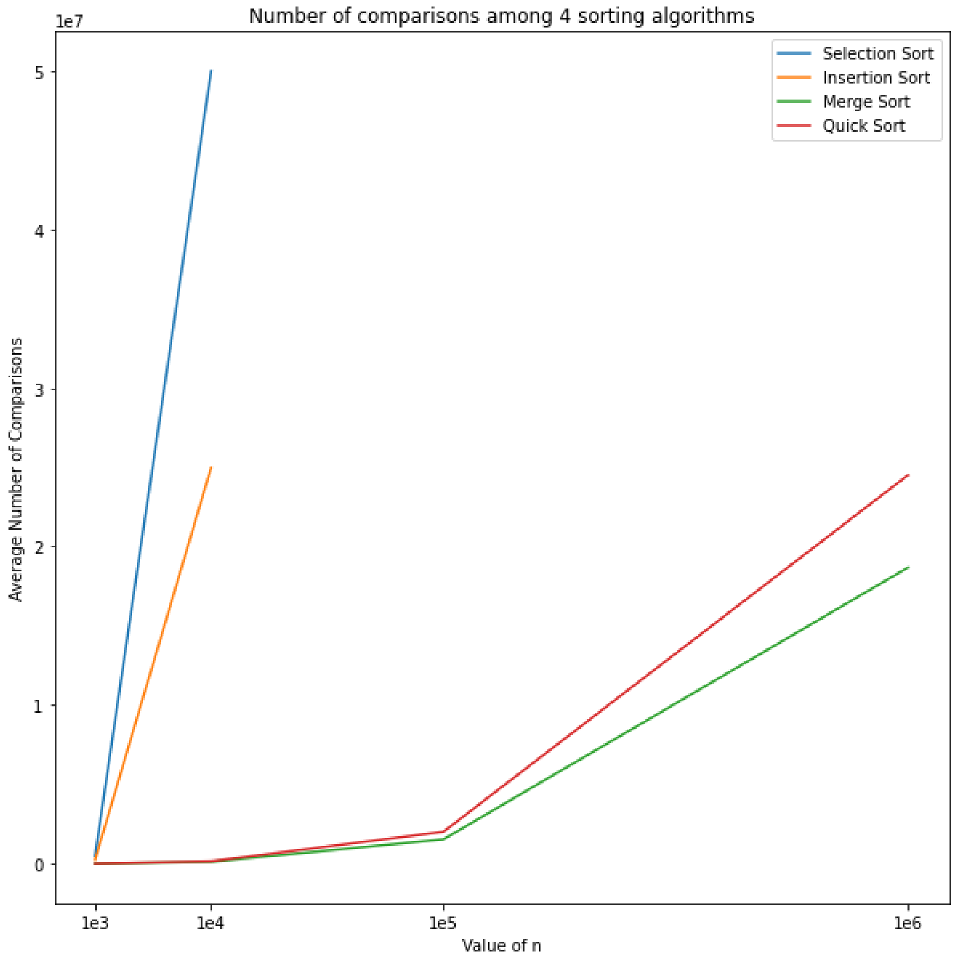
3. Visualize the difference in term of number of comparisons and running time among 4 sorting algorithms. Plot the first graph where the x-axis is the value of n and the y-axis is the average number of comparisons. Plot the second graph where the x-axis is the value of n and the y-axis is the average running time. Note that each graph must have 4 lines, each line with different color corresponding to different algorithm. Explain your findings in both theoretical and practical aspects.

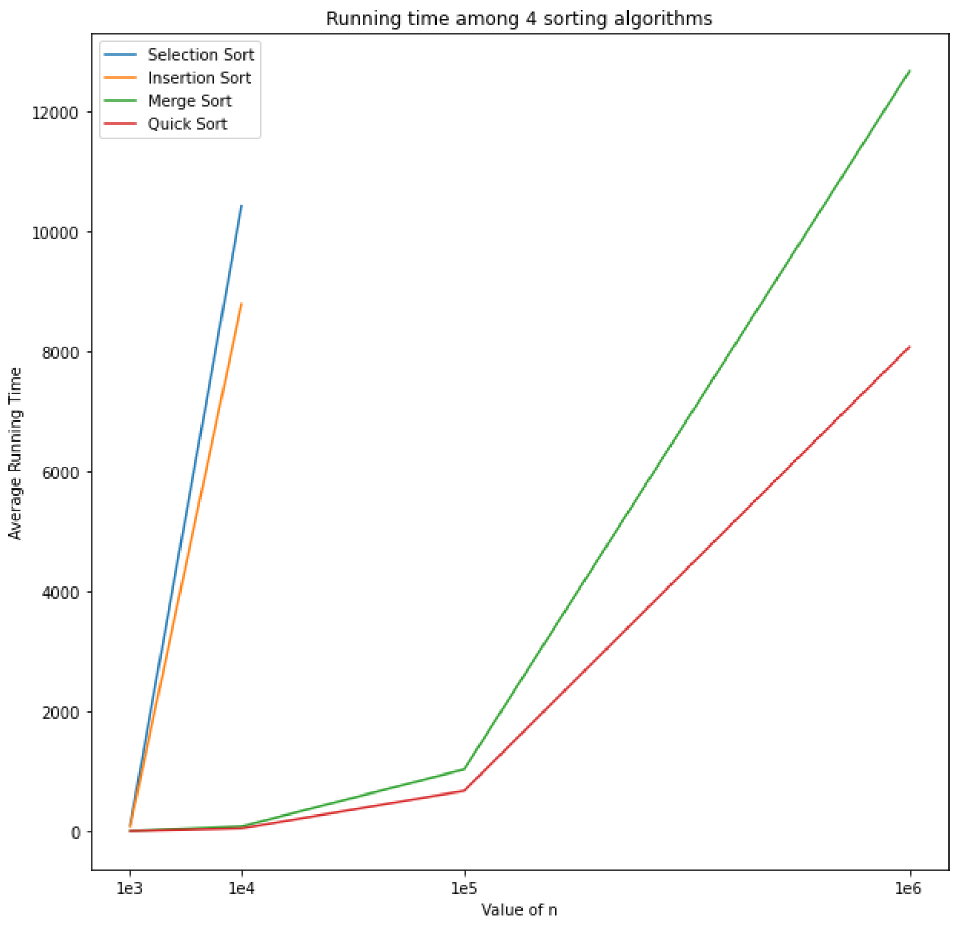
ANSWER:

For the python program, please refer to the program file:

Part II: 3.1 Visualization of the Average Number of Comparisons

Part II: 3.2 Visualization of the Average Running Time





In the above two figures, I have made some adjustments to the coordinate scale of the x-axis to facilitate viewing.

As we can see from the two figures above, selection sort has the highest number of comparisons and average running time, followed by insertion sort. When the amount of data has not yet reached 10^4, the number of comparisons and the average running time of these two algorithms have shown an exponential increase. In comparison, merge sort and quick sort perform very well. The above practical results also verify the theoretical time complexity of these four sorting algorithms.

- Selection Sort: $O(n^2)$

- Insertion Sort: $O(n^2)$

- Merge Sort: O(nlogn)

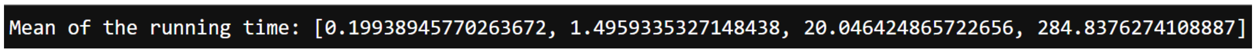
- Quick Sort: O(nlogn)

4. Compare the running time between your implementations with the built-in sort() function of the used programming language. Explain your finding.

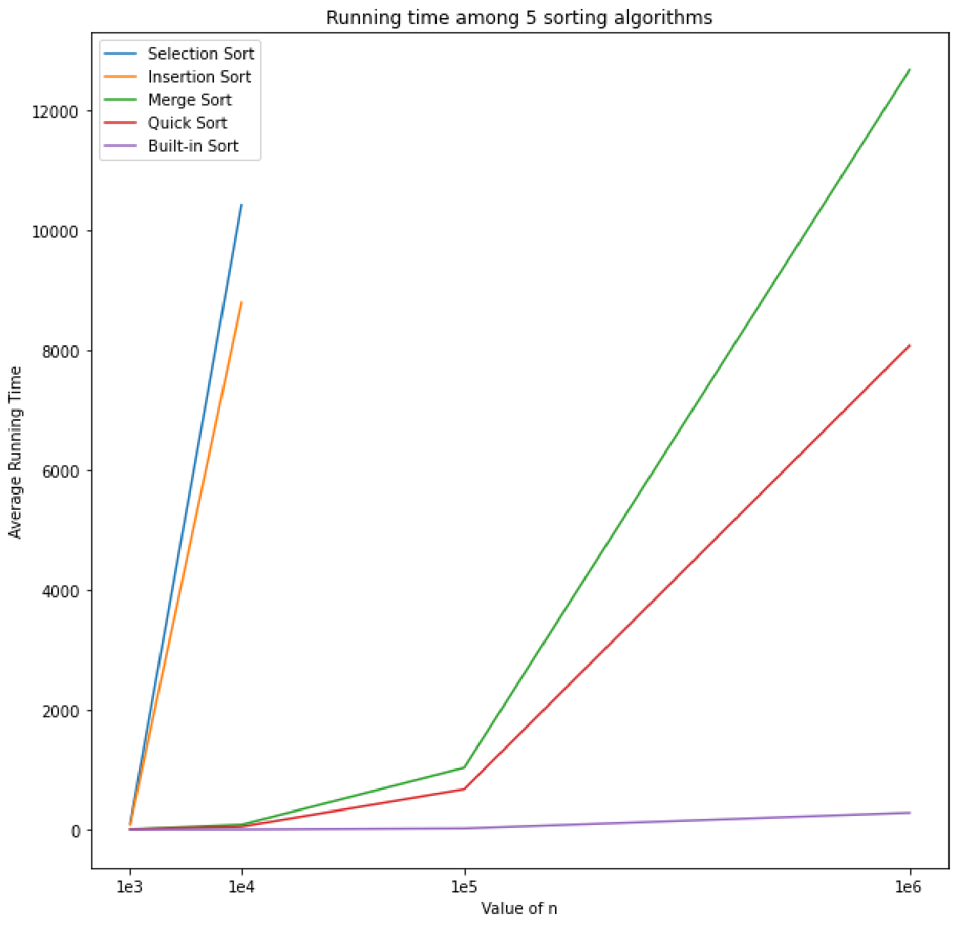
ANSWER:

For the python program, please refer to the program file:

Part II: 4.1 Built-in sort() function



```
Mean of the running time: [0.1993894577026367, 1.4959335327148438, 20.046424865722656, 284.8376274108887]
```



As shown in the above figure, python's built-in sorting algorithm is excellent, and its running time does not increase significantly even in the case of very large data volumes. After querying the Wiki, I learned that the built-in sorting algorithm of python is Timsort, which is a hybrid stable sorting algorithm derived from merge sort and insertion sort.

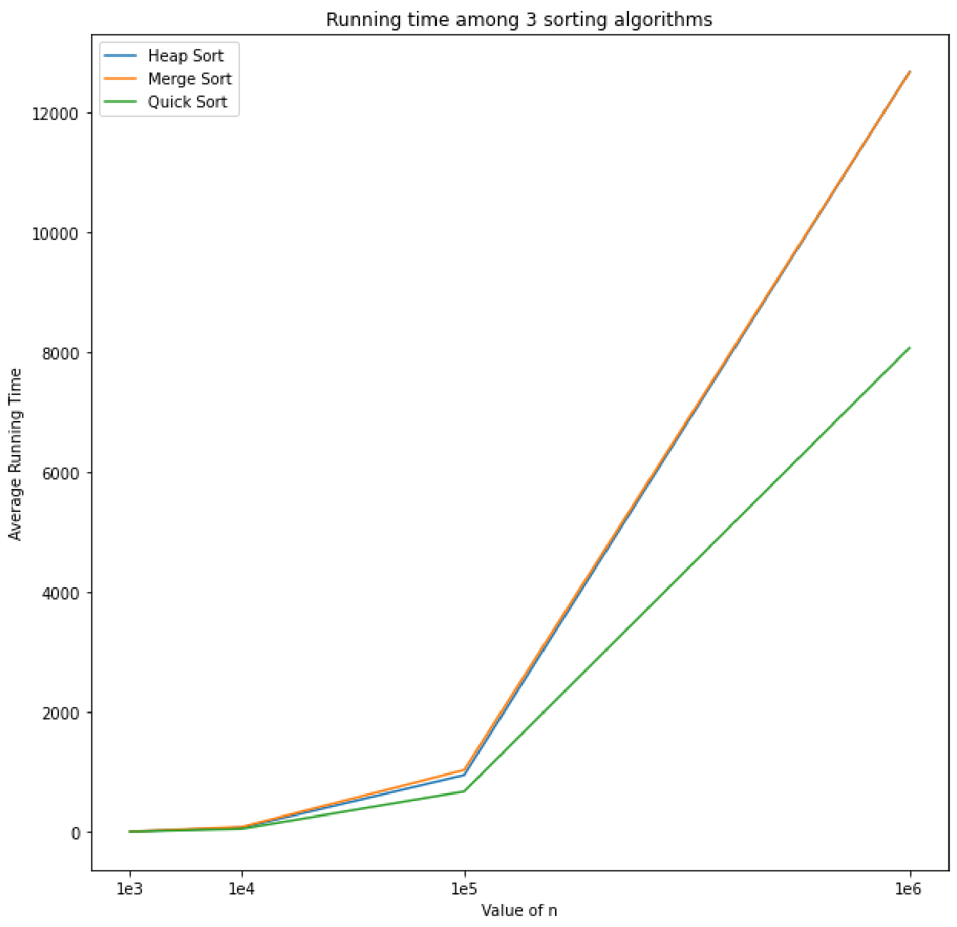
2 Implementing heap sort in ascending order (2 pts)

Write a program that implements heap sort in ascending order and uses your heap sort to sort an array of n random values in [0, 1] for n ∈ $\{10^3, 10^4, . . . , 10^6\}$. Similar to Task 1, we visualize the difference in running time between heap sort, merge sort and quick sort by plotting a graph where the x-axis is the value of n and the y-axis is the average time of 10 runs. The graph must have 3 lines, each line with different color corresponding to different algorithm. Explain your findings in both theoretical and practical aspects.

ANSWER:

For the python program, please refer to the program file: Task 2

```
Mean of the running time: [4.985976219177246, 71.608567237854, 951.5005350112915, 12581.477403640747]
```



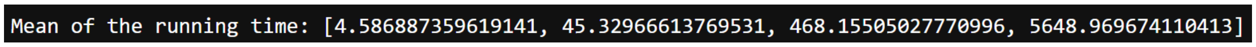
As shown in the figure above, the average running time required by the heap sort, merge sort, and quick sort algorithms are not much different. Relatively speaking, quick sort requires less running time, but both heap sort and quick sort are unstable. The above practical results also verify that the theoretical time complexity of these three sorting algorithms is the same O(nlogn).

3 Implementing radix sort in ascending order (2 pts)

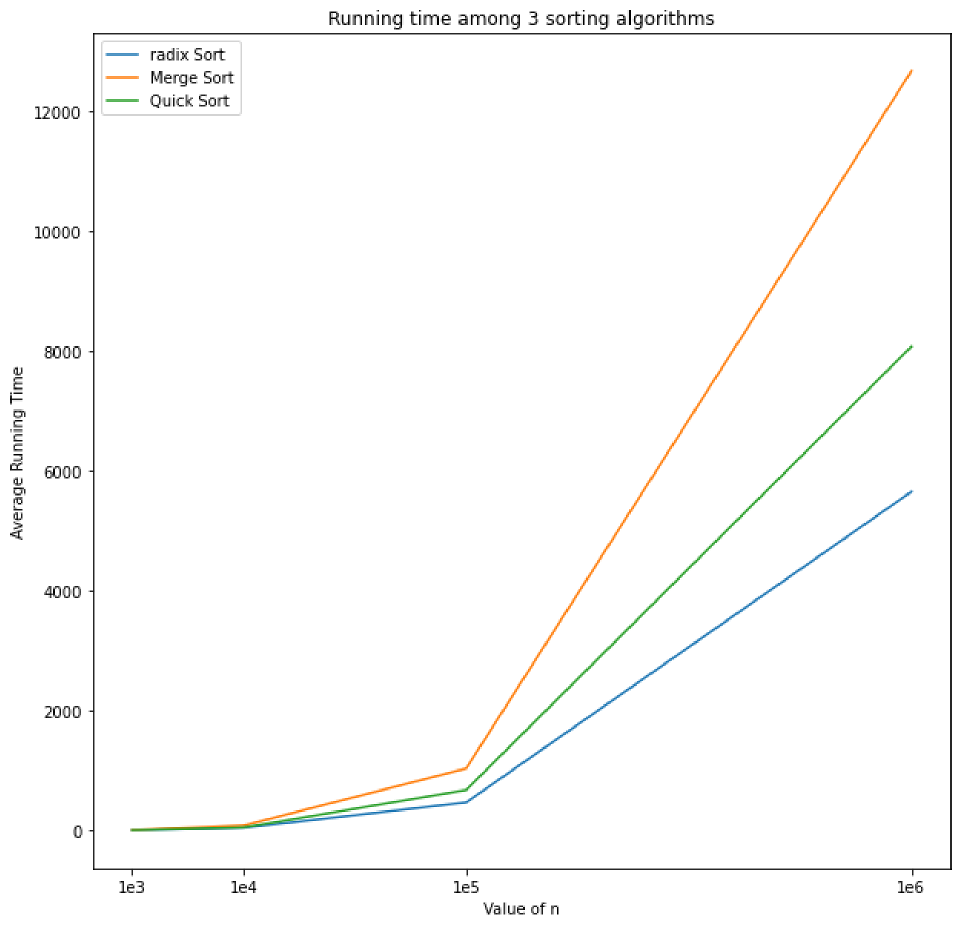
Write a program that implements the radix sort in ascending order and uses your radix sort to sort an array of n random integers in [0, 10^9] for n ∈ {10^3, 10^4, . . . , 10^6}. Similar to Task 1, we visualize the difference in running time between radix sort, merge sort and quick sort by plotting a graph where the x-axis is the value of n and the y-axis is the average time of 10 runs. The graph must have 3 lines, each line with different color corresponding to different algorithm. Explain your findings in both theoretical and practical aspects.

ANSWER:

For the python program, please refer to the program file: Task 3



```
Mean of the running time: [4.586887359619141, 45.32966613769531, 468.15505027770996, 5648.969674110413]
```



As shown in the above figure, among the three sorting algorithms, merge sort requires the longest average running time, followed by quick sort, and radix sort requires the least time. At the same time, both radix sort and merge sort are stable. The time complexity of radix sort is O(n*k), which is theoretically slightly better than O(nlogn) of merge sort and quick sort, which is also proved from practical results.

Submission Procedure

Submit your solutions (a pdf file and a programming file) to Canvas.