prob1-analysis

October 16, 2021

1 Problem 1

Student-Id: 2021-35791

Server: spds005

Name: (Moon JeongHyun) Server Used: Sciserver + Google Colab

For this assignment, I have used two different servers:

- Google Colab: https://colab.research.google.com/?utm_source=scs-index
- Sciserver: https://www.sciserver.org/
- Google Colab is an online Python platform developed by Google in order to share and run code with others. I believe Google Colab uses Python 3.7 rather than 3.8. Google Colab has RAM of about 12-13GB
- While I have planned on only using Google Colab, there were few of the notebooks that took more than an hour to run. As a result, I have used Sciserver to complete those. Sciserver is a science playform built and supported by the Institite for Data Intensive Engineering at the Johns Hopkins University.Sciserver is being continously developed and is up-to-date with most of the model. Sciserver itself may simply be considered as more enhanced version of Google Colab. Sciserver like Colab may be shared with others and may run many different version of Python. Sciserver has few cool features: CasJobs, Compute, Compute Jobs, SciDrive, Skyserver, SkyQuery. I will talk about only Compute and Compute Jobs for others may took too much time. Compute may be considered as a Google Colab with more functions where unlike Google Colab, one can run Python, R, and MatLab. In order to complete this assignment, I have used Compute Jobs where compute jobs basically runs the notebook and download once finished. The image I used was SciServer Essentials 2.0 which can run Python 3.8 (Anaconda 2020.11). R.4.03. TensoreFlow 2.3.0, Pytorch 1.71. In terms of RAM, it is about 40GB.

1.1 Analysis of Problem 1

Before going into the analysis of each of the cases (best, average, worst), I will create a table formed by results of different csv files and also discuss when worst and best cases occur.

Bubble Sort and **Insertion Sort** has a lot of things in common. For both cases, the best scenerio occurs when the list is sorted, and the worst scenerio occurs when the list is reverse-sorted. On the case of the **Seletion Sort**, like the two sorting methods, the best case occurs when the list is

sorted, but for the worst case, it occurs when the first is the largest element and the rest is in order. For **Merge Sort**, similar to the other three, best case occurs when the element is sorted. For the worst case, it occurs in the array where for given array in sorted order, the array will be splitted in such way where even and the odd index are continuously separated until one reach the state of single array. Then, one would combine all obtaining mix of original odd and even index. For **Quick Sort**, best case occurs when partition process always pick the middle element as pivot. In order to do this, I have created a function that will create values where it will check for middle value and do recursive root. Then, save it from backward to obtain the final solution, so that it will always pick the middle element as picot. For the worst case, it happens when partition process always picks the greatest or smallest element as pivot (often tend to be in ascending order - so I have used the sorted array to do this). For the **Counting Sort**, the best case occurs when all elements are same, and the worst case occurs when the data is skewed and range is large. For **Radix Sort**, best case occurs when all elements have same number of digits, and worst case occurs when only one element has significantly large number of digits and the other element has small number of digits.

For all average cases, I have obtained using random number of each length and computed average of the 5 runs.

Following are the time complexity for different sorting algorithms,

Sorting Algorithms	Best	Average	Worst
Bubble	$O(N^2)$	$O(N^2)$	$O(N^2)$
Insertion	$O(N)$	$O(N^2)$	$O(N^2)$
Selection	$O(N^2)$	$O(N^2)$	$O(N^2)$
Merge	$O(Nlog_2N)$	$O(Nlog_2N)$	$O(Nlog_2N)$
Quick	$O(Nlog_2N)$	$O(Nlog_2N)$	$O(N^2)$
Counting	$O(N+k)$	$O(N+k)$	$O(N+k)$
Radix	$O(Nd)$	$O(Nd)$	$O(Nd)$

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
df = pd.read_csv('dataset/best.csv')
df = df.drop(['Unnamed: 0','Radix','Select','Quick'],axis = 1)

df2 = pd.read_csv('dataset/best-r.csv')
df2 = df2.drop(['Unnamed: 0','Length'],axis = 1)

df3 = pd.read_csv('dataset/best-q.csv')
df3 = df3.drop(['Unnamed: 0','Length'],axis = 1)

df4 = pd.read_csv('dataset/best-s.csv')
df4 = df4.drop(['Unnamed: 0','Length'],axis = 1)

df = pd.concat([df,df2,df3,df4], axis = 1)
```

```
best = df.set_index('Length')
# convert second to microsecond
best = best * 1000000
best = best.reset_index()
best = best.apply(np.log10)
best = best.set_index('Length')
```

[3]:
```
df = pd.read_csv('dataset/average-b.csv')
df = df.drop(['Unnamed: 0'],axis = 1)
df2 = pd.read_csv('dataset/average-r.csv')
df2 = df2.drop(['Unnamed: 0','Length'],axis = 1)
df3 = pd.read_csv('dataset/average-i.csv')
df3 = df3.drop(['Unnamed: 0','Length'],axis = 1)
df4 = pd.read_csv('dataset/average-m.csv')
df4 = df4.drop(['Unnamed: 0','Length'],axis = 1)
df5 = pd.read_csv('dataset/average-q.csv')
df5 = df5.drop(['Unnamed: 0','Length'],axis = 1)
df6 = pd.read_csv('dataset/average-c.csv')
df6 = df6.drop(['Unnamed: 0','Length'],axis = 1)
df7 = pd.read_csv('dataset/average-s.csv')
df7 = df7.drop(['Unnamed: 0','Length'],axis = 1)

df = pd.concat([df,df2,df3,df4,df5,df6,df7], axis = 1)
average = df.set_index('Length')

# convert second to microsecond
average = average * 1000000
average = average.reset_index()
average = average.apply(np.log10)
average = average.set_index('Length')
```

[4]:
```
df = pd.read_csv('dataset/worst-b.csv')
df = df.drop(['Unnamed: 0'],axis = 1)
df2 = pd.read_csv('dataset/worst-r.csv')
df2 = df2.drop(['Unnamed: 0','Length'],axis = 1)
df3 = pd.read_csv('dataset/worst-i.csv')
df3 = df3.drop(['Unnamed: 0','Length'],axis = 1)
df4 = pd.read_csv('dataset/worst-m.csv')
df4 = df4.drop(['Unnamed: 0','Length'],axis = 1)
df6 = pd.read_csv('dataset/worst-c.csv')
df6 = df6.drop(['Unnamed: 0','Length'],axis = 1)
df7 = pd.read_csv('dataset/worst-s.csv')
df7 = df7.drop(['Unnamed: 0','Length'],axis = 1)
df8 = pd.read_csv('dataset/worst-q.csv')
df8 = df8.drop(['Unnamed: 0','Length'],axis = 1)

df = pd.concat([df,df2,df3,df4,df6,df7,df8], axis = 1)
```

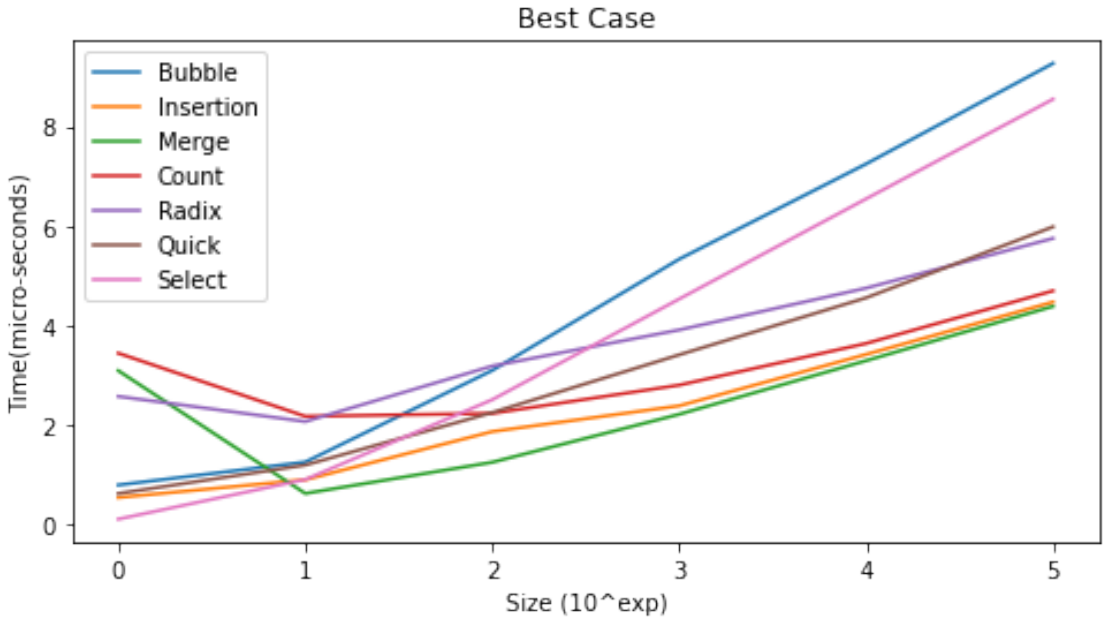
```
worst = df.set_index('Length')

# convert second to microsecond
worst = worst * 1000000
worst = worst.reset_index()
worst = worst.apply(np.log10)
worst = worst.set_index('Length')
```

1.1.1 Best Case

[5]:
```
fig, axs = plt.subplots(figsize=(8, 4))
best.plot(ax=axs)
axs.set_ylabel("Time(micro-seconds)")
axs.set_xlabel("Size (10^exp)")
axs.set_title('Best Case')
```

[5]:
```
Text(0.5, 1.0, 'Best Case')
```



Because the graph of 1,10,100,1000,10000,100000 makes it hard to see values in entirety, I have also made the length to be of base 10. Ideally speaking, when only considering the time-complexity of different functions, it was expected to perform in the following manner: - $O(Nlog_2N)$ -> $O(N)$ -> $O(N+k)$ -> $O(Nd)$ -> $O(N^2)$

The equivalent order of above is Merge ~ Quick -> Insertion -> Counting -> Radix -> Bubble ~ Selection.

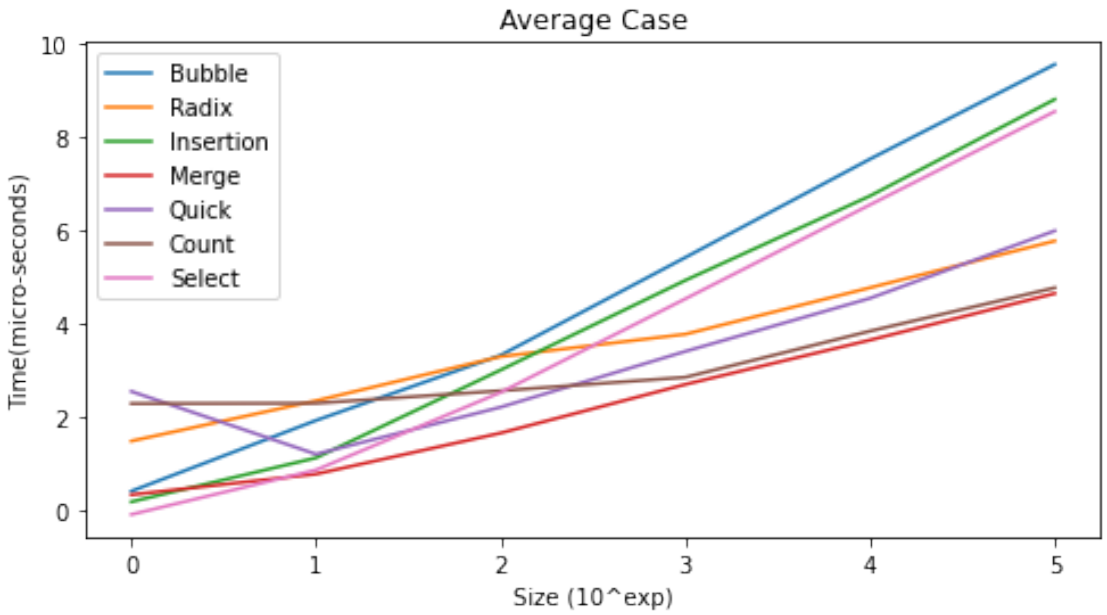
Looking at the result, merge sort has performed very well as expected. Interesting enough Quick Sort has not performed as well as I would have expected. In terms of its location, it has performed

worse than that of Radix Sort but better than those of $O(N^2)$. Other than the Quick Sort, insertion, count, and radix sort has performed as expected. In terms of $O(N^2)$, selection sort has better performance than the bubble sort.

1.1.2 Average Case

[6]:
```
fig, axs = plt.subplots(figsize=(8, 4))
average.plot(ax=axs)
axs.set_ylabel("Time(micro-seconds)")
axs.set_xlabel("Size (10^exp)")
axs.set_title('Average Case')
```

[6]:
```
Text(0.5, 1.0, 'Average Case')
```



Like the best case, I have also made the length to be of base 10. Ideally speaking, when only considering the time-complexity of different functions, it was expected to perform in the following manner: - $O(Nlog_2N)$ -> $O(N+k)$ -> $O(Nd)$ -> $O(N^2)$

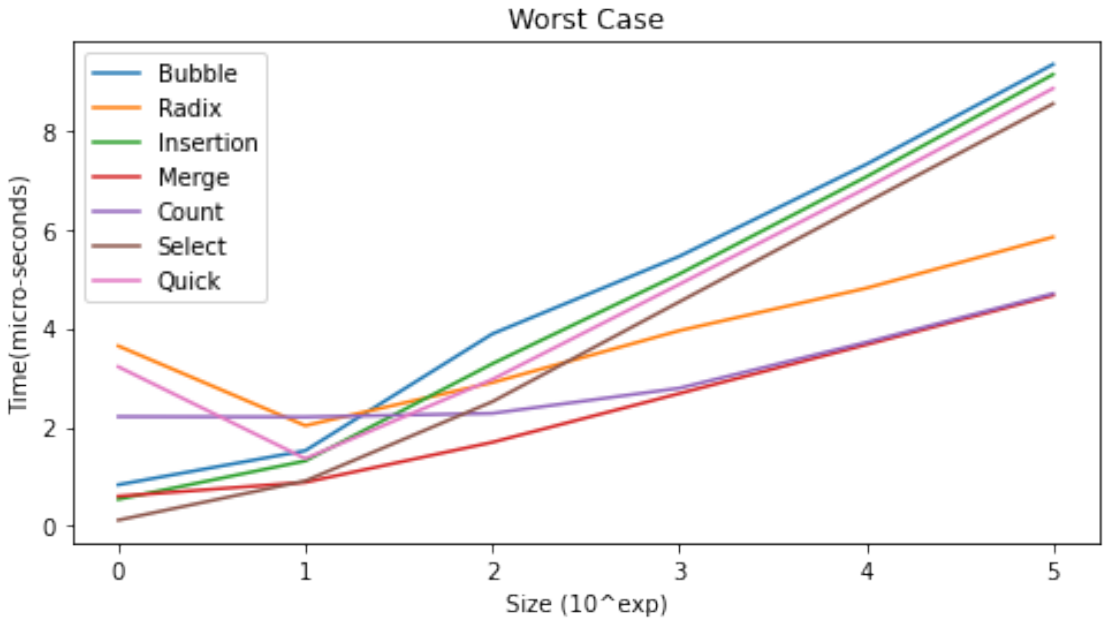
The equivalent order of above is Merge ~ Quick -> Counting -> Radix -> Bubble ~ Selection ~ Insertion.

Just like best case, as expected merge sort has performed the best. Again for the quick sort, it has performed worse than Radix but better than those of $O(N^2)$. Since insertion sort is no longer the case of $O(N)$ as expected from time complexity, count sort did next well after merge sort. After the count sort, the Radix Sort performs the best. In terms of the three $O(N^2)$, Selection sort performed the best, followed by insertion, then, bubble.

1.1.3 Worst Case

[7]:
```
fig, axs = plt.subplots(figsize=(8, 4))
worst.plot(ax=axs)
axs.set_ylabel("Time(micro-seconds)")
axs.set_xlabel("Size (10^exp)")
axs.set_title('Worst Case')
```

[7]:
```
Text(0.5, 1.0, 'Worst Case')
```



Like before, I have also made the length to be of base 10. Ideally speaking, when only considering the time-complexity of different functions, it was expected to perform in the following manner: - $O(Nlog_2N)$ -> $O(N+k)$ -> $O(Nd)$ -> $O(N^2)$

The equivalent order of above is Merge -> Counting -> Radix -> Bubble ~ Selection ~ Insertion ~ Quick.

As expected, merge has performed the best, and was followed by counting sort and radix sort. Since Quick Sort is no longer $O(Nlog_2N)$,it now performed far warse then Radix Sort comparing to Average and Best case. In terms of the four $O(N^2)$, Selection performed the best, followed by Quick, Insertion, and Bubble.

1.2 Finding and Possible Guesses

Except for the quick sort, all the other algorithms has performed in an expected order. I may think of few reasoning behind why quick sort did not do so well on the average and best cases. As I was reading Wikipedia for the quick sort, often for the partition modeling, the performance gets worse as the same number increases. Since the range of our value was smaller than that of 10^5, that may explain why the Quick Sort has not performed so well. If I made the Quick Sort to have unique

numbers, it may have been possible for quick sort to perform as equally good or similar to that of Merge Sort.

Based on the results, it would be nice for one to use Merge Sort over all of the other algorithm, for it has best performance in all of its cases. If not merge sort, count sort would be other good sorting method to use. Definitely, it would be unwise for one to use bubble sort because among the family of $O(N^2)$ and overall, bubble sort had the worst performance. If one need to choose from $O(N^2)$, selection sort would be the way to go