prob2-analysis

October 16, 2021

1 Problem 2

Student-Id: 2021-35791

Server: spds005

Name: (Moon JeongHyun) Server Used: Google Colab

1.1 Analysis of Problem 2

Before going into the analysis, I will create a table formed by results of different csv files for sort and unsorted like that of problem 1.

Like problem 1, I will create a table that illustrates complexity of the following searching algorithm

Searching Algorithms	Linear	Binary
Sorted	$O(N)$	$O(log_2 N)$
Unsorted	$O(N)$	$O(Nlog_2 N)$

In order to do this problem, for each length given, I have created a random number from 0 to 999 inclusive and run 5 different iteration and printed out the average. Because the binary search is assumed list to be sorted, I have made minor modification from the class code so that it will sort first before it goes to search

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[5]:
```
df = pd.read_csv('dataset/sort.csv')
df = df.drop(['Unnamed: 0'],axis = 1)

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

[4]:
```
df = pd.read_csv('dataset/unsort.csv')
df = df.drop(['Unnamed: 0'],axis = 1)

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

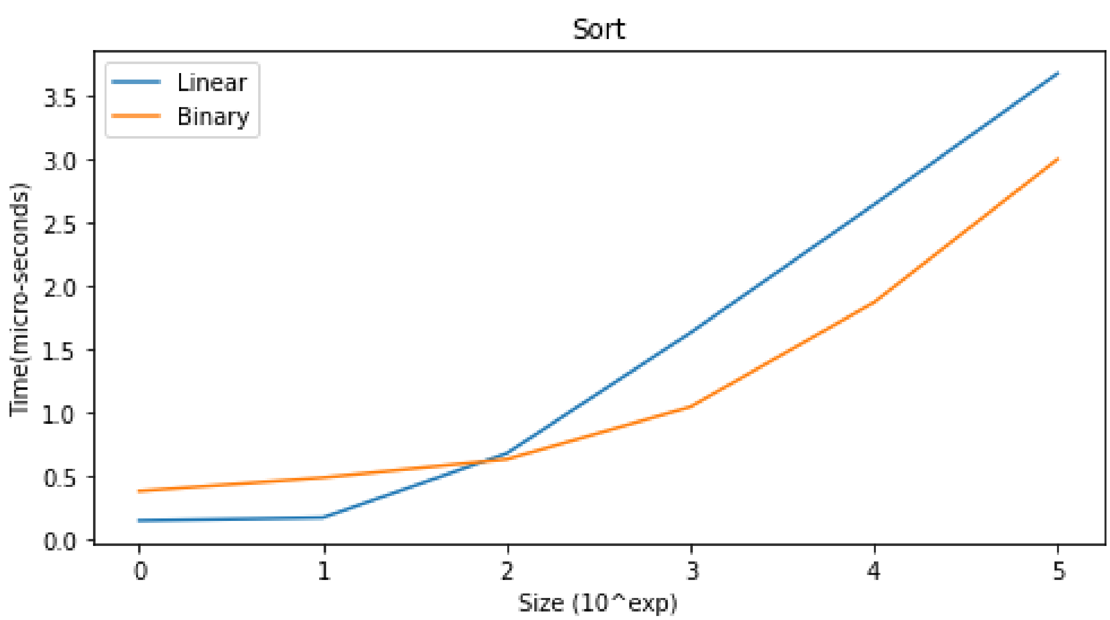
[4]:

Length	Linear	Binary
0.0	0.139879	0.240549
1.0	0.170262	0.387390
2.0	0.695482	0.668386
3.0	1.655906	1.502700
4.0	2.656960	2.556881
5.0	3.672709	3.629320

1.1.1 Sort

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

[6]: Text(0.5, 1.0, 'Sort')



Like problem 1, 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: - $O(log_2N)$ -> $O(N)$

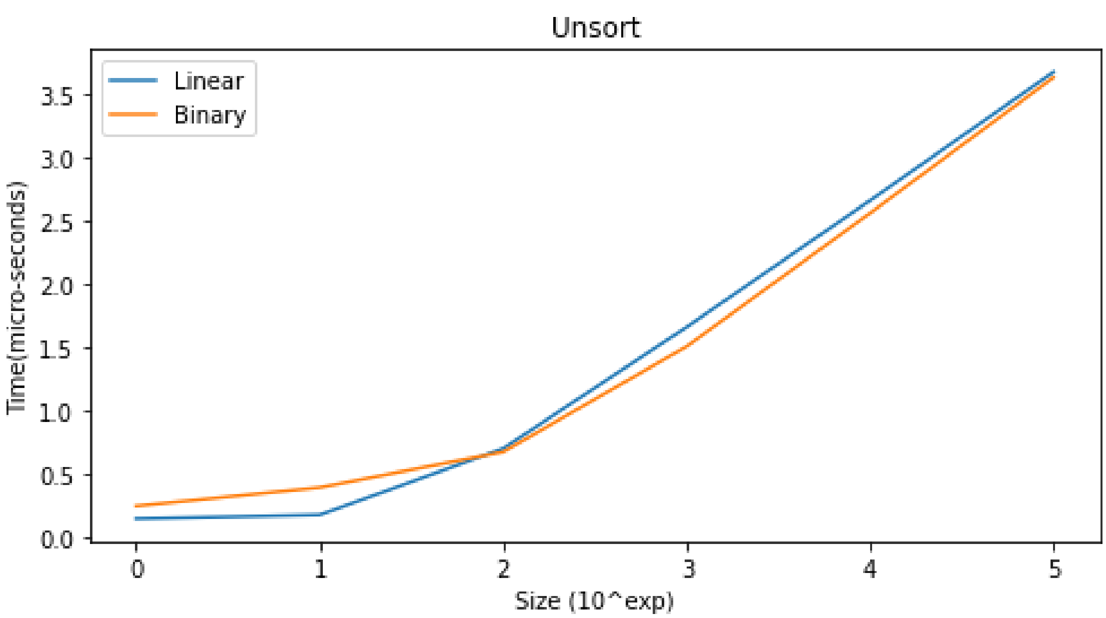
The equivalent order of above is Binary -> Linear.

As expected, as the size of the array increases, the binary search will outperform linear search.

1.1.2 Unsort

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

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



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: - $O(Nlog_2N)$ -> $O(N)$

The equivalent order of above is Binary -> Linear.

As expected binary search performed better than that of linear. This example really looked similar to that of the Merge and Insertion in the best case scenerio