Report1

Matthew Keon Hee Lee
Cmpt 225
301283427

Report for Experiment 1

Table of Contents:

Introduction

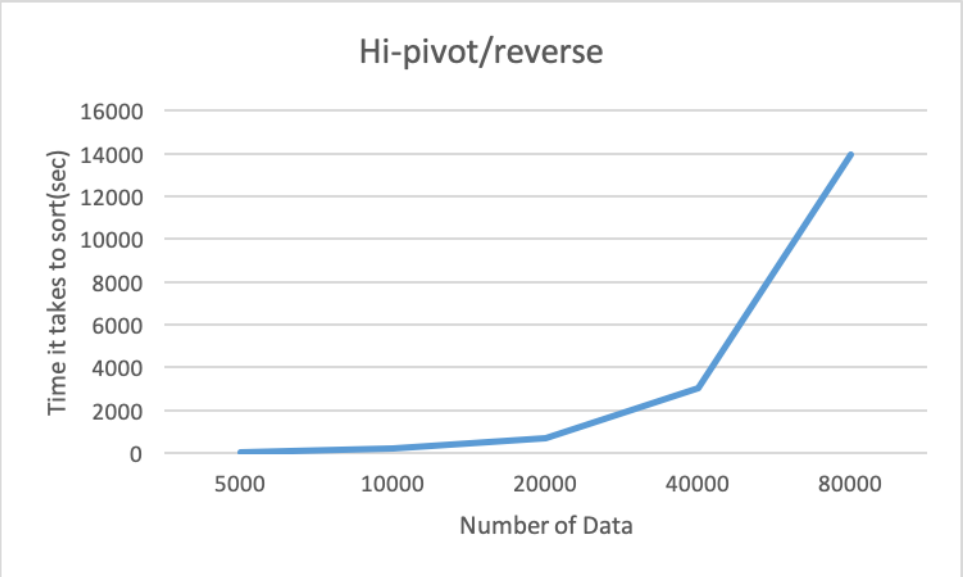
In this experiment, I have use 3 different quick sort methods (include given quick sort) which are Hi-pivot method, mid-pivot method, and random-pivot method. Those methods have same algorithm but have different way of choosing pivot. For inputs, I use to put as reverse order, order(sorted), same element, and random array because I make hypothesis that the order of the list can affect the time of the sorting and maybe same element can affect the sorting, and lastly unsorted(random) array is usual array. I want to see reverse, order (already been sorted), random, and everything same elements have different result. I use 5000,10000,20000,40000, and 80000 data sizes. I choose these sizes because I want to see as size double, I want to see how much time increases.

Reverse Case

The chart of data for unordered:

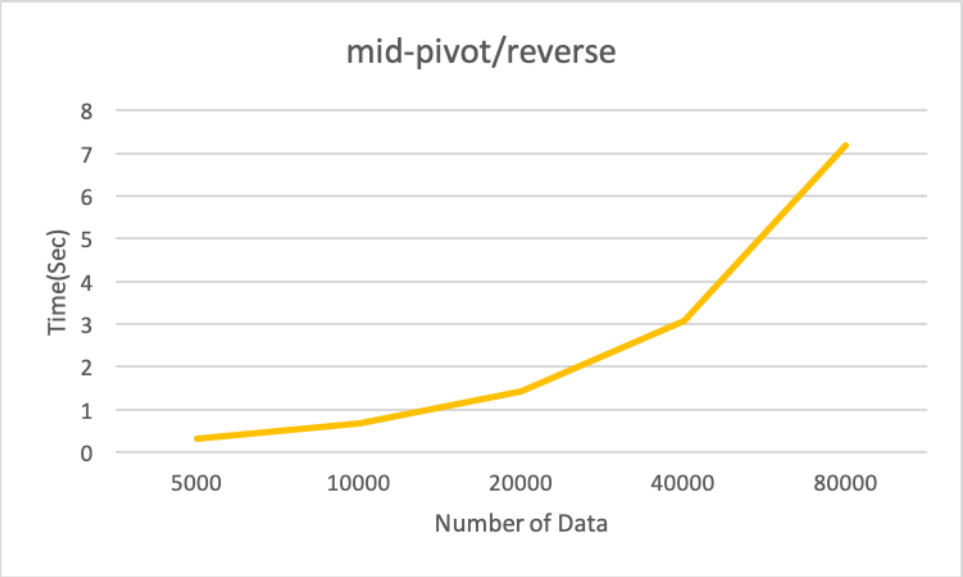
Number	Hi-pivot/reverse	mid-pivot/reverse	random-pivot/reverse
5000	44.9876	0.315	1.1341
10000	177.991	0.6664	2.6879
20000	711.266	1.4323	7.286
40000	3019.05	3.0631	21.0009
80000	13944.2	7.1712	61.6252

Hi pivot reverse graph



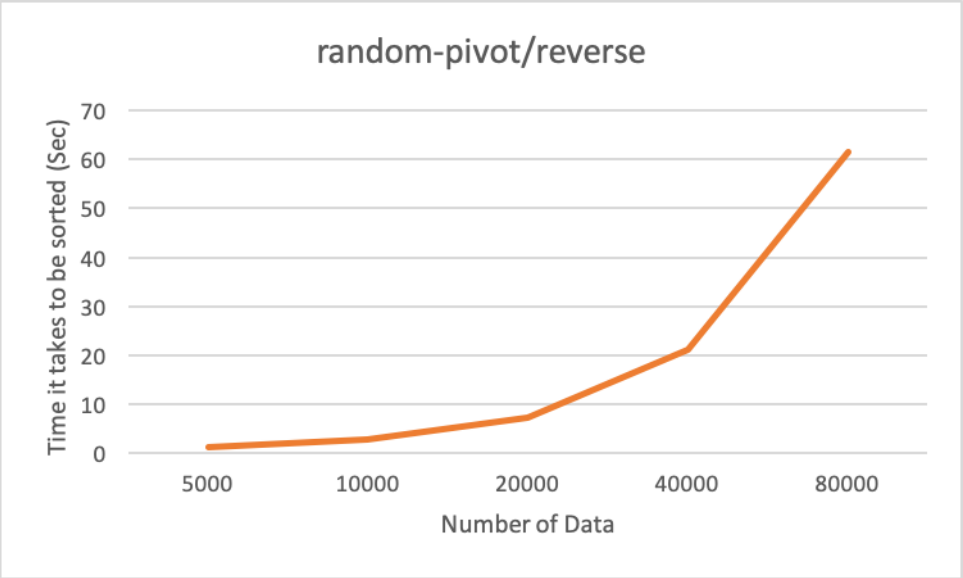
Hi pivot takes most of the time when we reverse insert the item in the list. Since it is choosing lowest number from the list (Hi pivot method and reverse) that is not way of choosing pivot which it nearly takes $O(n^2)$ which as size double, the time approximately increase by quadruple.

Mid pivot reverse graph



Mid pivot takes shortest time from the all other sort methods. From the data, we can see data is about 2 (or bit higher than 2) times bigger when increase the data twice larger. This seems to be better method than Random pivot method which both have O(n log n) for big oh.

Random pivot reverse graph



Random pivot takes quite shorter time than hipivot method but still it is slower than mid pivot method. From the data, we can see data is about 3 (or bit smaller than 3) times bigger when increase the data twice larger.

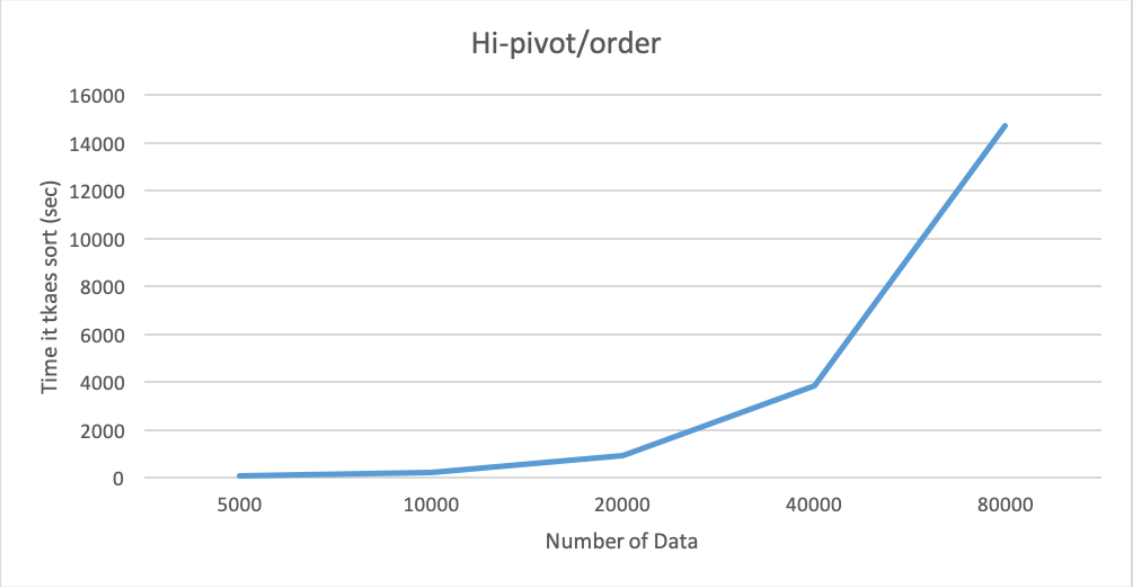
Conclusion for reverse order:
From all of the data, as size double, Hi pivot will quadruple, Mid pivot will double, and Random pivot will triple. I have observed even if it is same sort, choosing the pivot can effect on results.

Order Case

The chart of order:

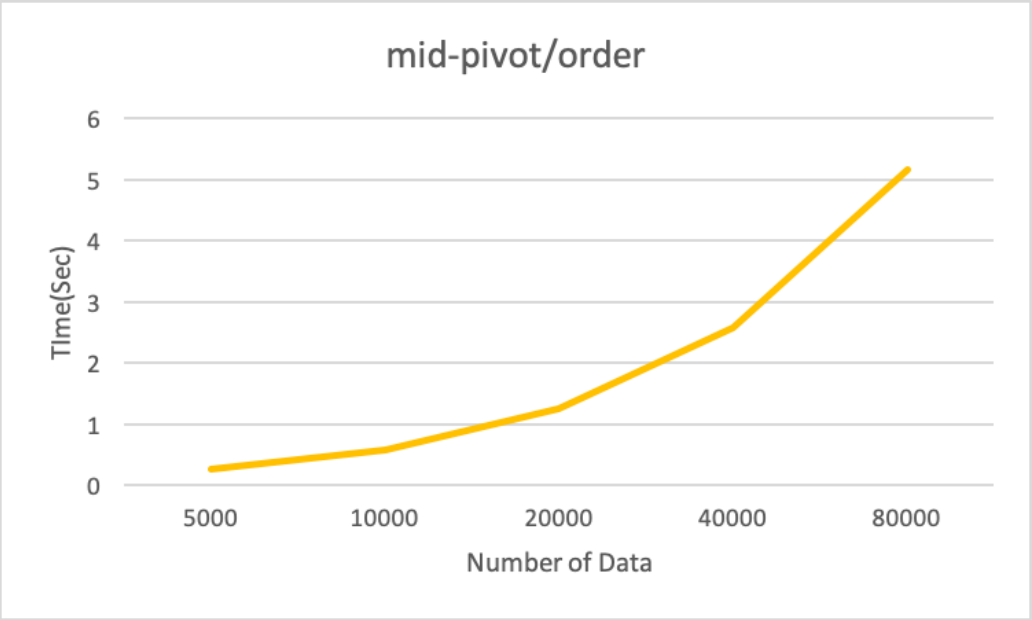
Number	Hi-pivot/order	mid-pivot/order	random-pivot/order
5000	57.0008	0.2551	1.0793
10000	232.487	0.574	2.906
20000	929.633	1.2457	7.5902
40000	3836.6	2.5857	21.0761
80000	14724.1	5.1731	56.7185

Hi pivot order graph



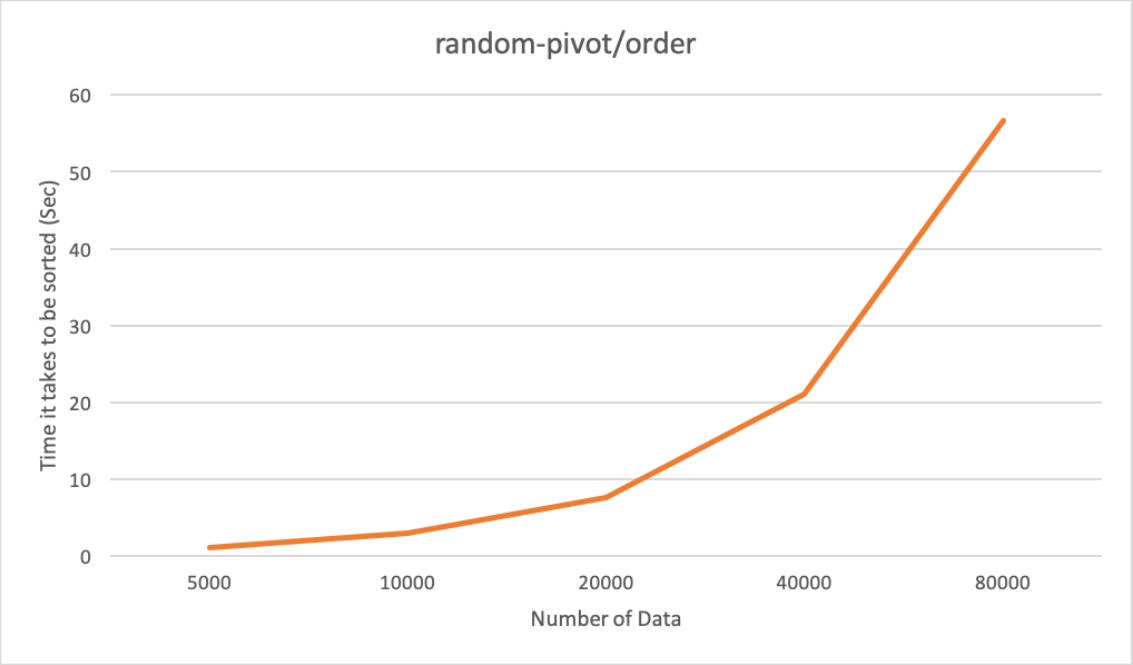
Surprisingly, hi pivot order takes more time than hi pivot reverse. I can make assumption because if we choosing hi pivot from order, we are choosing highest number and number is in order which we need to shift the number even it is already sorted.

mid pivot order graph



Another interesting thing is mid-pivot order is faster than mid pivot reverse. Since the difference between each other is whether it is order or reverse. I can assume this is just how much computer is good at that point (Apple Mac is faster than the lab computer in sfu) since it is very little time.

random pivot order graph



Similar reason to mid pivot/ order

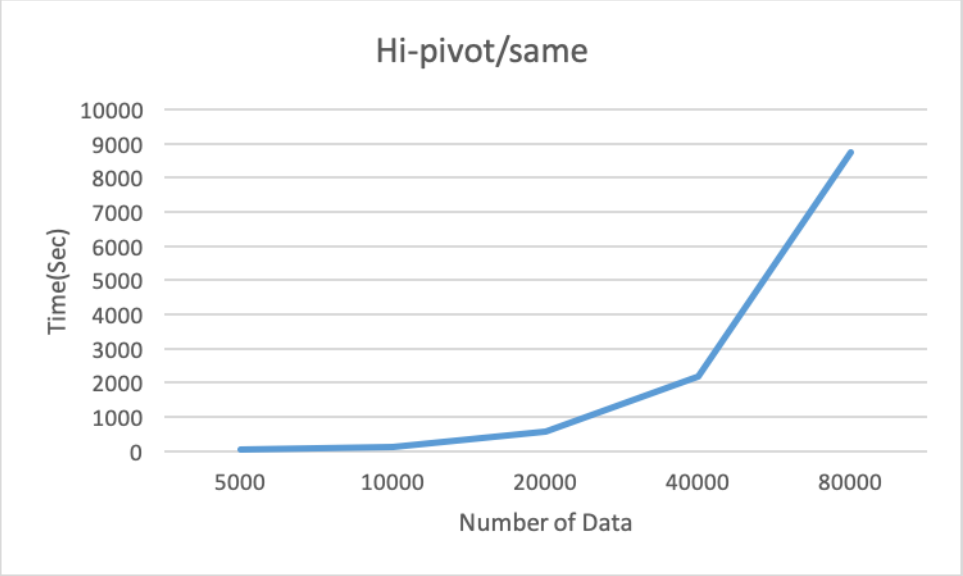
Conclusion for order:
From this, I hypothesis that order and reverse order have big difference, however, I found there are not big difference between order and reverse order.

Same Case

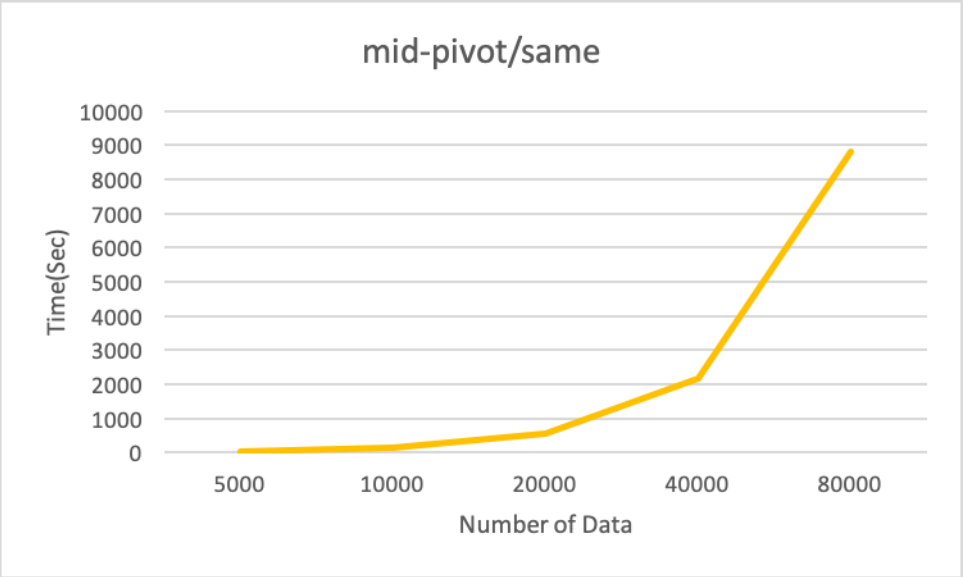
The chart of same element:

Number	Hi-pivot/same	mid-pivot/same	random-pivot/same
5000	34.2101	32.9637	33.1883
10000	137.892	137.643	137.824
20000	553.631	563.068	563.144
40000	2167.95	2144.72	2180.07
80000	8728.63	8712.22	8811.4

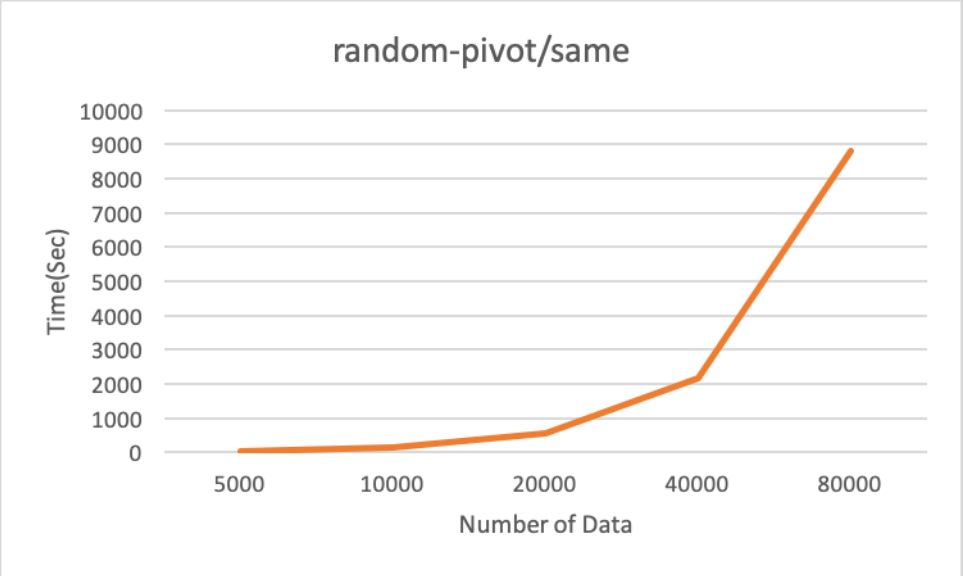
Hi pivot Same graph



mid pivot same graph



random pivot same



Conclusion for same:

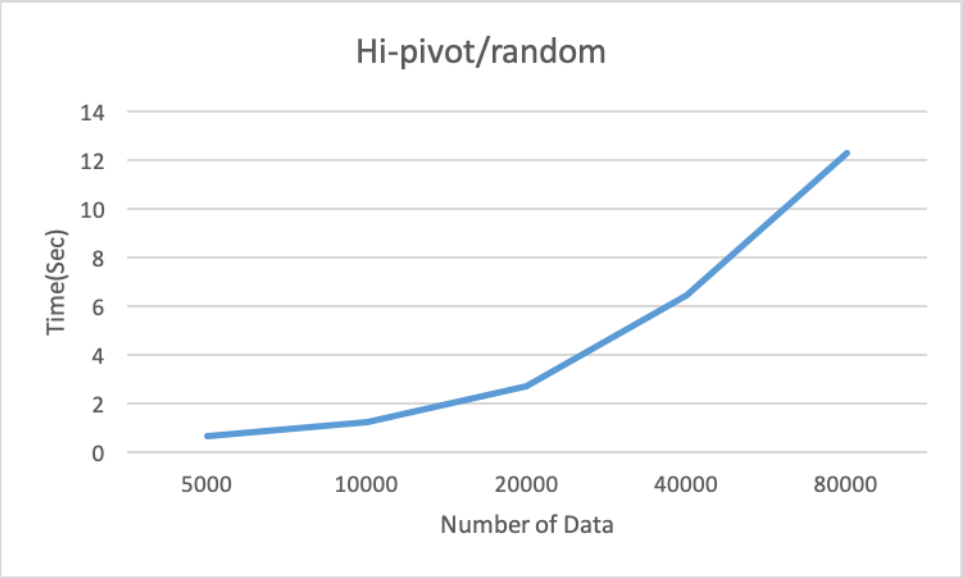
Different from the other two experiments above I done, all the methods seem to have same time and same big-oh. This implies the choosing any pivot would not have big difference than 3 other methods. Moreover, the c++ sort method and Insertion Sort is much faster than the quick sort in this case.

Random Case

The chart of random:

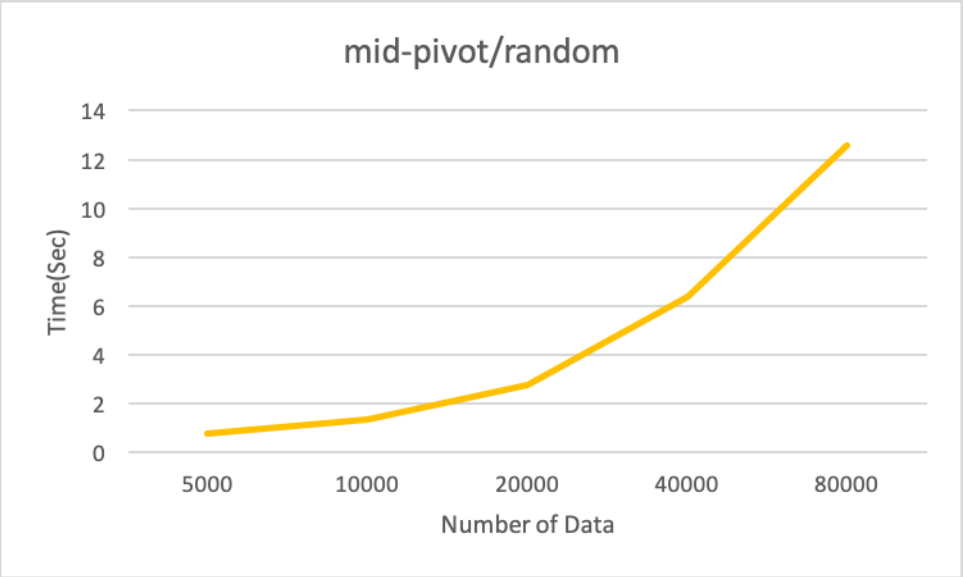
Number	Hi-pivot/random	mid-pivot/random	random-pivot/random
5000	0.6795	0.7471	1.3476
10000	1.2613	1.3313	3.0508
20000	2.69	2.7456	7.7651
40000	6.4339	6.3619	22.4128
80000	12.2816	12.5967	55.2631

Hi-pivot random graph



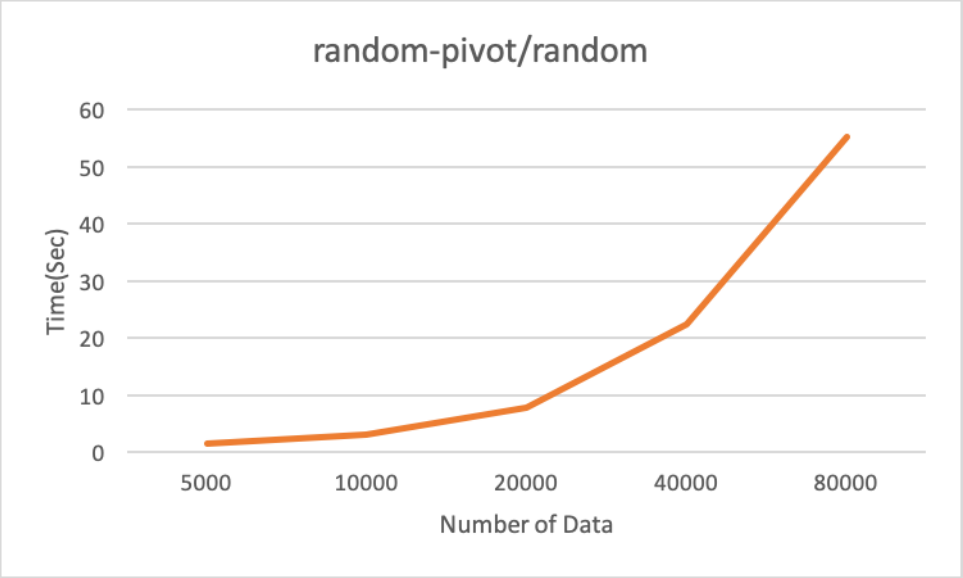
In this experiment, hi-pivot method is fastest which I was keep thinking this method is lazy and slowest, but it is actually best method for general case.

Mid-pivot random graph



It is interesting that it has similar time to hi pivot method.

Random-pivot graph



It is one of the slowest method in this experiment.

Conclusion for random:

As time double, all the method approximately doubles the time. However, Hi pivot method and Mid pivot method will take similar time and Random pivot would take more time than other two methods.

Conclusion

Conclusion for Report1:

Generally, the worst case for all quick sort method is when all the elements are same in the list. In reverse and ordered array, we need to use mid pivot method and in random array case, we need to use either mid pivot method or hi pivot method.

Report2

Matthew Keon Hee Lee
Cmpt 225
301283427

Report for Experiment 2

Table of Contents:

Introduction

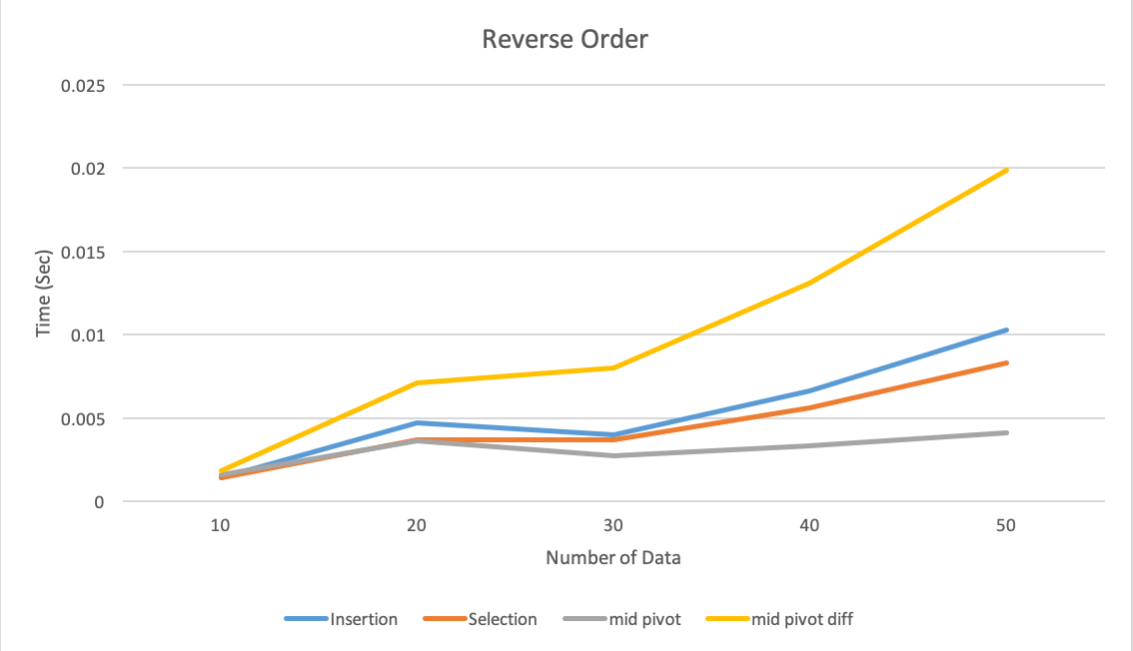
In this experiment, I have used Insertion and Selection Sort for simple sort and I had use two different versions to see what method is better for certain inputs. For my case, I was using mac due to COVID19. The better computer makes the algorithm works faster. This makes me to use very little input so I can find the S where the graph intersects. Same to the report 1, I use reverse, order, same element, and random array if I could see the difference of intersection for each cases.

Reverse Case

Chart for Reverse

Number	Insertion	Selection	mid pivot	mid pivot diff
10	0.0014	0.0014	0.0016	0.0018
20	0.0047	0.0037	0.0036	0.0071
30	0.004	0.0037	0.0027	0.008
40	0.0066	0.0056	0.0033	0.0131
50	0.0103	0.0083	0.0041	0.0199

Graph for Reverse



When the data is at 10, selection sort and insertion sort are slightly faster than mid-pivot quick sort. However, at 20, mid pivot quick sort gets faster than two simple sorts and it is faster afterwards. If data is smaller than 20, it is better to use simple sorts and if data is bigger than 20, it is better to use mid pivot quick sort. To prove my conclusion is correct, I use large number of data (10000). Then I found my hypothesis is correct.

```
SIZE:10000
C++ sort: 0.0278
Insertion Sort: 174.966
Quicksort: mid pivot: 0.6886
Selection Sort: 139.857
Quicksort different alg: 396.528
```

Order Case

Chart for Order

Number	Insertion	Selection	mid pivot	mid pivot diff
10	0.0009	0.001	0.0015	0.0011
20	0.0008	0.002	0.0014	0.0022
30	0.0021	0.0049	0.0032	0.0041
40	0.0027	0.0065	0.0042	0.005
50	0.0009	0.0067	0.0038	0.0048

Graph of order

Insertion sort was always faster till 50 of data and depend on the data above, we can make hypothesis that the insertion sort will always faster than the mid pivot quick sort. So when I test with 10000 size of data for this case, it was nearly 20 times faster and this proof that my prediction was right. For selection sort, it was faster than mid pivot quick sort when it is around 10, however, it became slower afterward. So when I test with 10000 size of data, selection sort was slow by about 213times (132.651/ 0.6239). For ordered case, the best method is insertion sort which this experiment clearly shows at ordered cases (best case for insertion sort, $\Theta(n)$), it is faster than quicksort.

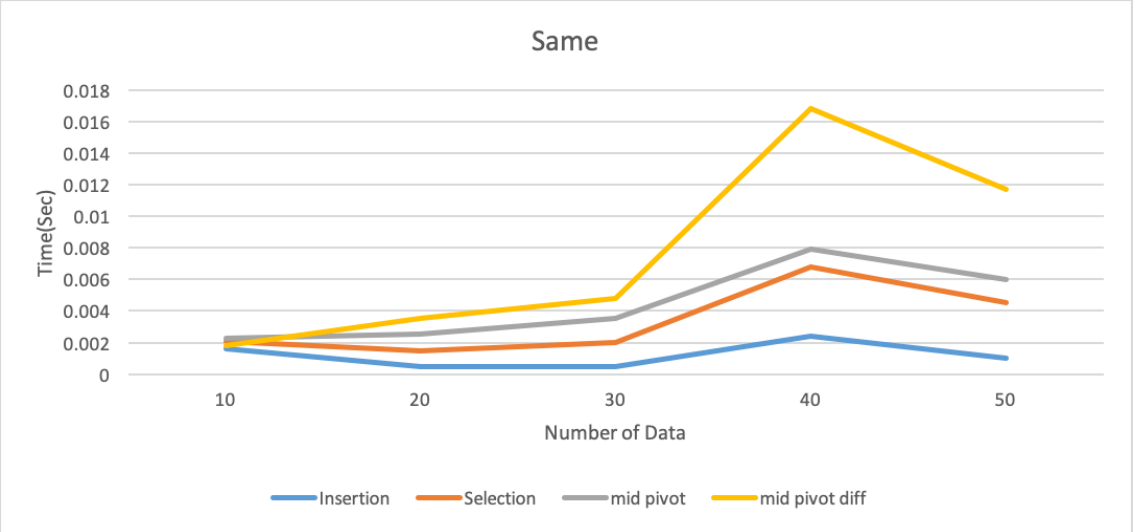
```
SIZE:10000
C++ sort: 0.0134
Insertion Sort: 0.0513
Quicksort: mid pivot: 0.6239
Selection Sort: 132.651
Quicksort different alg: 1.1375
```

Same Case

Chart of Same

Number	Insertion	Selection	mid pivot	mid pivot diff
10	0.0016	0.0021	0.0023	0.0018
20	0.0005	0.0015	0.0025	0.0035
30	0.0005	0.002	0.0035	0.0048
40	0.0024	0.0068	0.0079	0.0168
50	0.001	0.0045	0.006	0.0117

Graph of Same



Insertion sort was always faster till 50 of data and depend on the data above, we can make hypothesis that the insertion sort will always faster than the mid pivot quick sort. So when I test with 10000 size of data for this case, it was nearly 2675 times faster and this proof that my prediction was right. For selection sort, it was faster than mid pivot quick sort till 50 so I was predicting that this might be similar case as insertion sort. However, when I test with 10000 size of data for this case, the time was pretty similar to the mid pivot quick sort which this proves my prediction is wrong. To conclude, when all the elements are same in array, we need to use insertion sort.

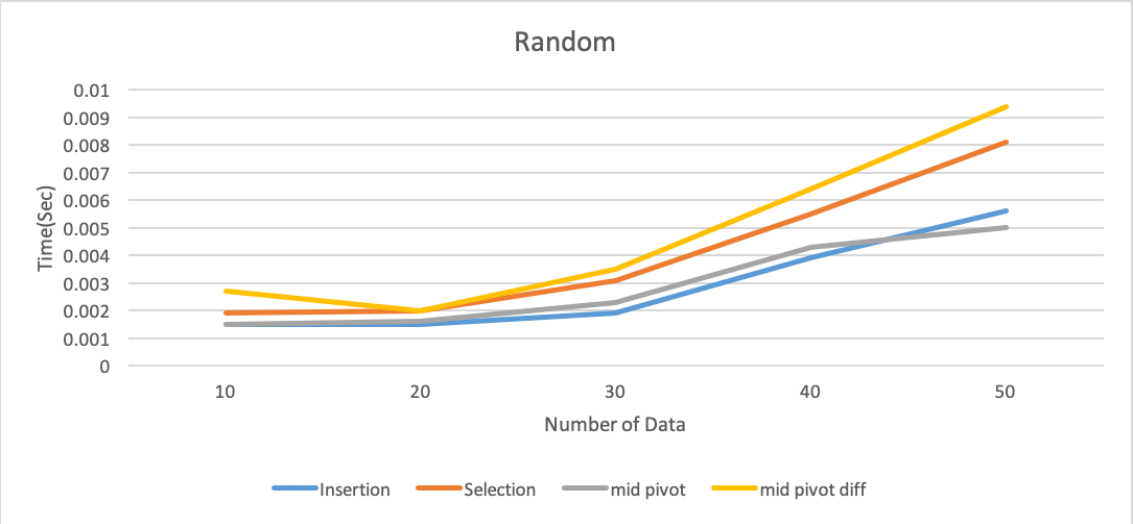
```
SIZE:10000
C++ sort: 0.013
Insertion Sort: 0.0503
Quicksort: mid pivot: 134.565
Selection Sort: 134.161
Quicksort different alg: 392.674
```

Random Case

Chart of Random

Number	Insertion	Selection	mid pivot	mid pivot diff
10	0.0015	0.0019	0.0015	0.0027
20	0.0015	0.002	0.0016	0.002
30	0.0019	0.0031	0.0023	0.0035
40	0.0039	0.0055	0.0043	0.0064
50	0.0056	0.0081	0.005	0.0094

Graph of Random



Insertion sort is faster than mid pivot quick sort until data size 40. However, it became slower after data size 40. So I test when size is 10000, mid pivot quick sort is faster than Insertion sort by 63times. For Selection sort, selection sort seems faster than quicksort with different algorithm. However, when I test with size 10000, selection sort is slower than the quicksort with different algorithm. To conclude, when its random array, when size is under 40, it is better to use insertion sort but when it is over 40, it is better to use mid pivot quick sort to sort the array of data.

```
SIZE:10000
C++ sort: 0.5712
Insertion Sort: 88.7345
Quicksort: mid pivot: 1.3726
Selection Sort: 138.881
Quicksort different alg: 96.2991
```

Additional Source

Explanation for additional Source I use

I have implemented selection sort since I do not want depend my conclusion by comparing one simple sort. I believe my report get more precise by using data of selection sort.

I have implemented quick sort with different variant since I believe different variant could be faster for some cases. However, my prediction was wrong and in most case, the way I had implement the quick sort was slower on every case than usual mid pivot quick sort method.

Conclusion

Conclusion for report2

In our assignment, we want to find the point S is either same or not same for all the cases. Then we figure it out that point S is not same for all cases. Furthermore, for every cases, there exist fastest method or best method to implement the code. In some cases, Insertion sort is faster than Quick sort. This contradicts our knowledge before testing since we believe quick sort would be faster in most cases than the insertion sort since the big oh of insertion sort is $O(n^2)$ where quick sort is O(nlogn).