Project1

author: Ke Chen, Xunyu Chu

October 2020

1 Description of the Result

Our algorithms behave as expected for both sorted and unsorted input arrays. For sorted array $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, the output is $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. As for the unsorted array $\{1, 2, 3, 4, 5, -1, -2, -3, -4, -5, 0\}$, the output is a sorted array: $\{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$.
In the meantime, our algorithms can pass the test about array with one element, array with repeated elements, array with all repeated elements, array in descending order, array with all negative inputs, array with mixed negative and positive inputs, array with size $2^k - 1$.

2 Analysis of the Result

2.1 Analysis of the algorithms

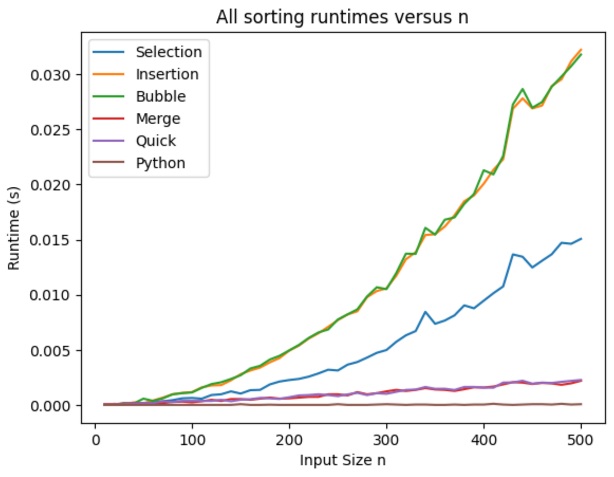


Figure 1: all sorting run-times versus n

Picture 1 displays the plot of all sorting run-times versus n. It is clear that Insertion sort and Bubble sort costs the more time and merge sort and quick costs smaller time.

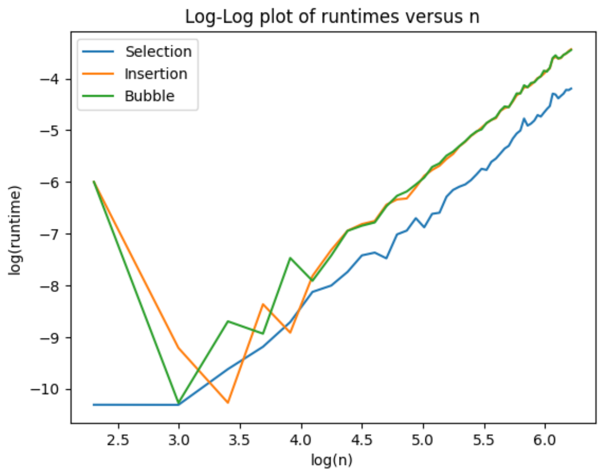


Figure 2: Log-Log plot of run-time versus n

The run-times of selection sort, bubble sort and insertion sort are all $O(n^2)$(We can verify this through Picture 2, the selection sort, insertion sort and bubble sort all have similar slope especially when n becomes bigger). Although selection sort, insertion sort and bubble sort all need to copy the data from caches to registers, and then store the sorted data to caches again, bubble sort and insertion sort need to do more lw and sw operation, which will cost more time than selection sort. As to the algorithms themselves, we believe bubble sort is more easier to understand. Since we only need to compare and swap, but insertion sort need to separate the array into "sorted" and "unsorted" components. Therefore, we consider insertion sort is the worst sorting algorithm.
When n becomes bigger, the Log-Log slope of merge sort and quick sort turn to near 1. which means they are much more quicker than selection sort, insertion sort and bubble sort. However, quick sort can choose a point value k to arrange elements bigger than k and elements smaller than k, and the elements bigger than k and elements do not need to be equal. Besides, quick sort do not require to allocate extra new space to store the split halves. By contrast, merge sort need to spilt the array into two halves and sort and then merge the sorted halves again. Which wastes more time than quick sort. Therefore, we believe quick sort is the the best.

2.2 The influence of value n on run-time

Picture 2 displays Log-Log plot of algorithm selection sort, insertion sort and bubble sort. According to Picture 2, when $log(n) < 4$, the $log(run - time)$ of the three algorithms changes violently. Which means we can not get a slope precisely when n is small. While n gets bigger, we can see the Log-Log plot resemble straight lines.
After our discussion, we believe the reason is that we need to copy many data from caches to register. However, when n is small, the computer still need to copy many data, and that amount may way larger than n. This costs a waste of time. In other words, when n is small, the time cost by copy data from caches to registers and then restore data to register compare to the time cost by sort is relatively big, and that will influence the calculation of our run-time of algorithms. However, as n become much bigger, the time cost by copy data from caches to register by sort is relatively smaller, because we now need to cost much more time on sort. And the Log-Log plot can now seen as the run-time of algorithm for sorting array elements. Therefore, the influence of copy data from registers to caches can barely ignore. Therefore, we need to report theoretical run-times of asymptotically large values of n.

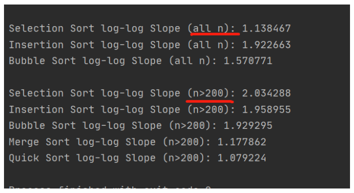
3 Discussion

3.1 Analysis of Data Processing

Data processing is a very important part of experiment. For this experiment, we process the data by computing the average of the data. After discussion, our team think that there will be 2 main reasons to use this method.
Firstly, as we can not ensure that we have the seriously same environment to execute our code, so the difference between different environment will make our result imprecise and we can not have a correct conclusion.
Secondly, as different original data may make the algorithms behave differently. For example, if we have an array $A = \{2, 1, 4, 3, 6, 5\}$, $B = \{6, 5, 4, 3, 2, 1\}$, although both of them are unsorted arrays, we can spend less time to get the final result of A than that of B. So, if we just use either A or B to get the Final Conclusion, it may lead to a wrong answer.
As a result, we use average to decrease the error incurred by the difference of environment and data. And this average will lead to a more precise result.

3.2 The influence From other program

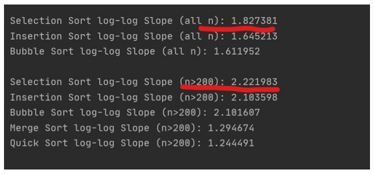
As the computer will **Parallel at the macro level, serial at the micro level**, so when we execute a computationally expensive task in computer, the run-time of the sort algorithm will be longer than before.



```
Selection Sort log-log Slope (all n): 1.138467
Insertion Sort log-log Slope (all n): 1.922663
Bubble Sort log-log Slope (all n): 1.570771

Selection Sort log-log Slope (n>200): 2.034288
Insertion Sort log-log Slope (n>200): 1.958955
Bubble Sort log-log Slope (n>200): 1.929295
Merge Sort log-log Slope (n>200): 1.177862
Quick Sort log-log Slope (n>200): 1.079224
```

Figure 3: no-task in background



```
Selection Sort log-log Slope (all n): 1.827381
Insertion Sort log-log Slope (all n): 1.645213
Bubble Sort log-log Slope (all n): 1.611952

Selection Sort log-log Slope (n>200): 2.221983
Insertion Sort log-log Slope (n>200): 2.103598
Bubble Sort log-log Slope (n>200): 2.101607
Merge Sort log-log Slope (n>200): 1.294674
Quick Sort log-log Slope (n>200): 1.244491
```

Figure 4: have background task

3.3 Discussion about the theoretical time and the experimental time

Our team has proved that we have used scientific method to do the experiment and get a final conclusion. However, we still did not know the difference between the theoretical time and the experimental time. So our team discuss the difference between them and the corresponding situation where we need to use one of them.
For the theoretical time, we get this time without relying on any existing environment and we just regard the time of the single step of the algorithm as constant. So when want to know which algorithm is better, we usually use the theoretical time as we can directly compare the performance of the algorithm without caring about their environment.
For the experimental time, firstly, if we have a custom chip which is designed for a specific algorithm, we can use the experimental time to prove our design. And it will also lead us to think about the advantage and disadvantage of the algorithm, in terms of the actual execution(such as if the memory access is too many times). Secondly, we can also know which computer is better by referring the experimental time.

4 Conclusion

In this project, we implement 5 kinds of sorting algorithm and analyse the execution time of them. In order to get a more precise result, we compute the average number to overcome these effect. Depending on the data and our analysis, we conclude that the Quick sort is the best one among these 5 algorithm and the Insertion sort is the worst on among these 5 algorithm. Furthermore, we also analyse the influence caused by the environment, such as memory access, background tasks. After that, we finally conclude that when we have a large n array, our algorithm will perform like the theoretical result and show the reason of our conclusion.
In conclusion, the best one is Quick Sort and The worst is Insertion Sort and the performance of algorithm will also be affected by the environment.