CPSC 319 Assignment 2: Comparison of four Sorting Algorithm

Name:Shanzi Ye

Instructor:Bajwa, Sohaib

Submission Date: July,21,2022

Question1： The data collected (use tables and graphs to help illustrate this).

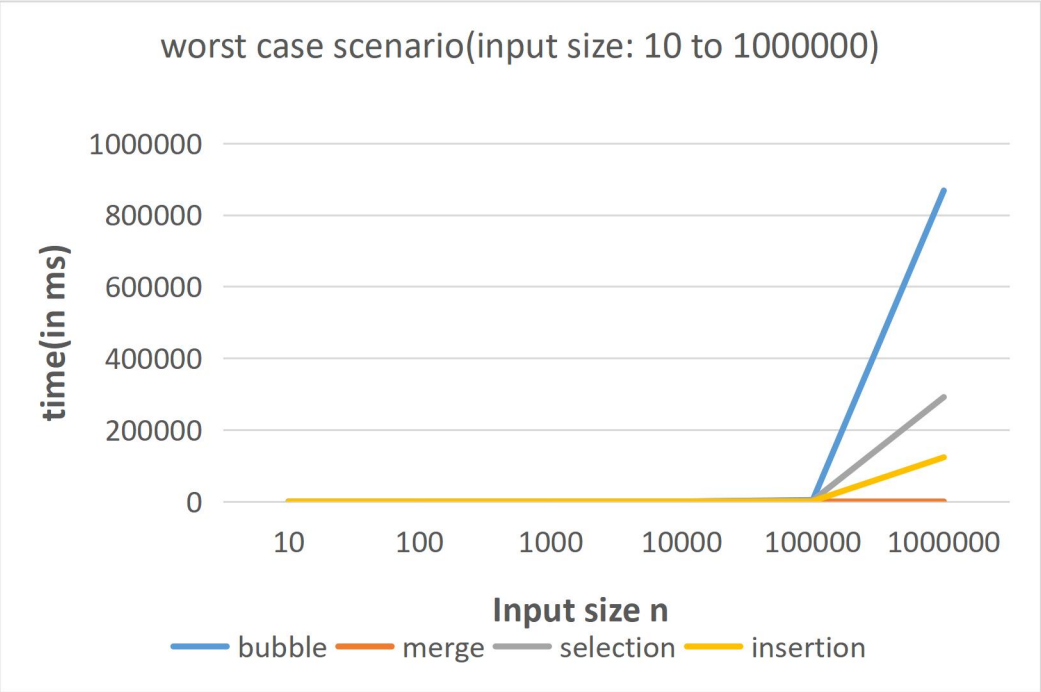
1.Worst case scenario:

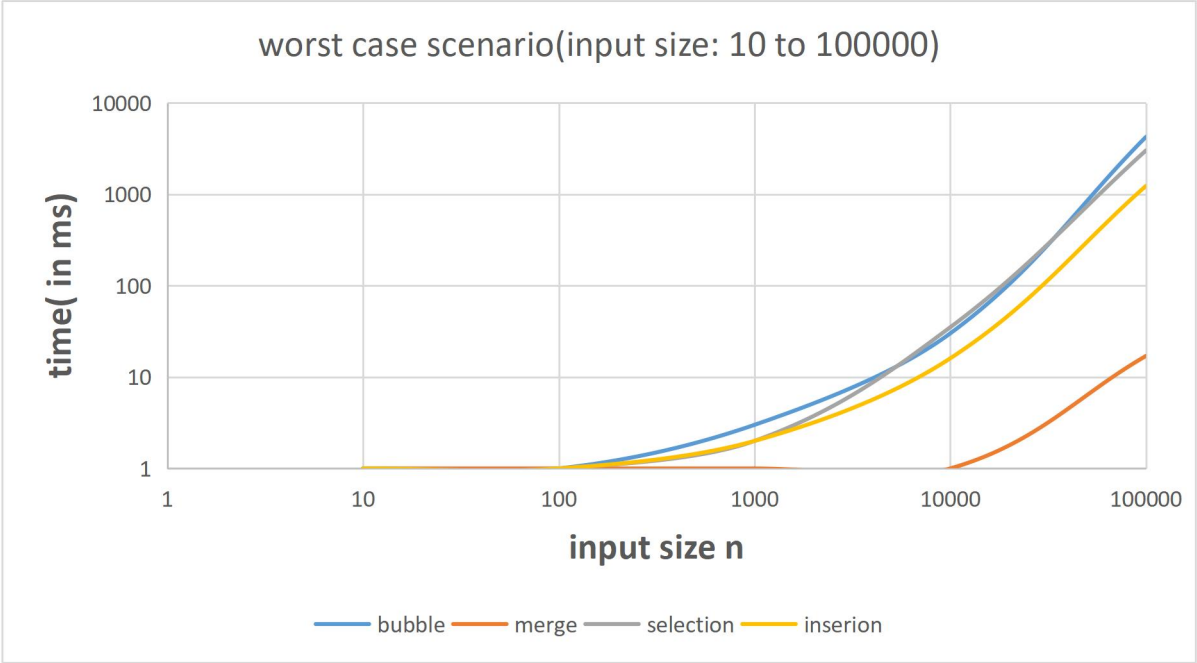
The worst case scenario happens when the user inputs “descending” in the beginning.

Data collected for the worst case scenario:

algo \ size	10	100	1000	10000	100000	1000000
bubble sort	0 ms	0 ms	3 ms	30 ms	4232 ms	867920 ms
merge sort	0 ms	0 ms	0 ms	1 ms	17 ms	84 ms
selection sort	0 ms	1 ms	2 ms	35 ms	3009 ms	291456 ms
insertion sort	0 ms	0 ms	2 ms	16 ms	1231 ms	123333 ms

In order to help reader visualize the difference between these four algorithms, I made two graphs with different y and x axis scales):



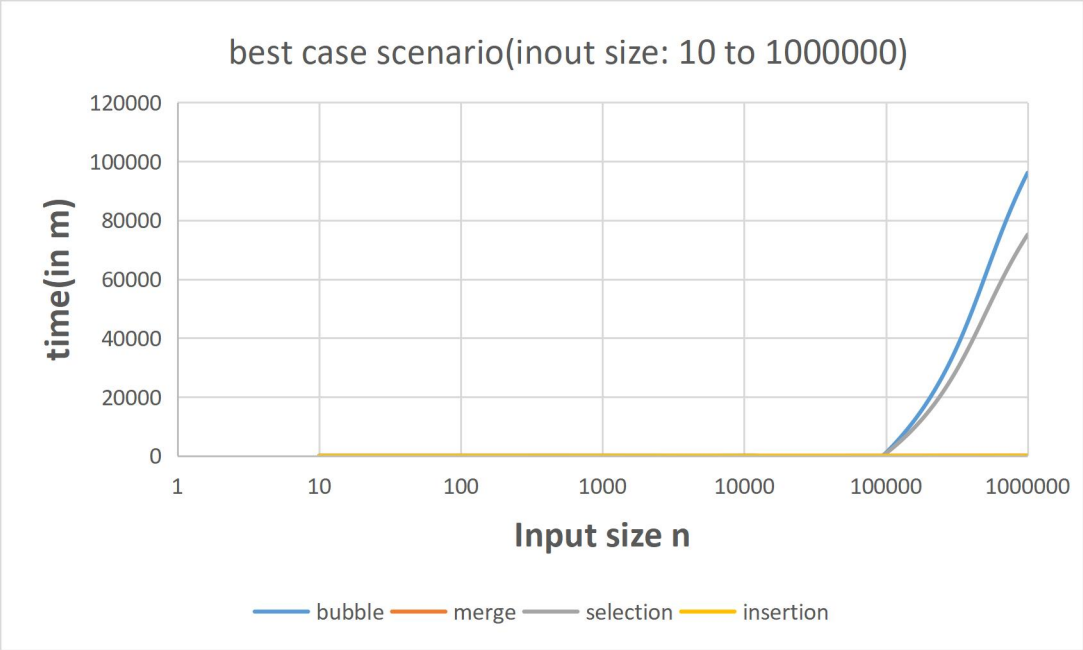


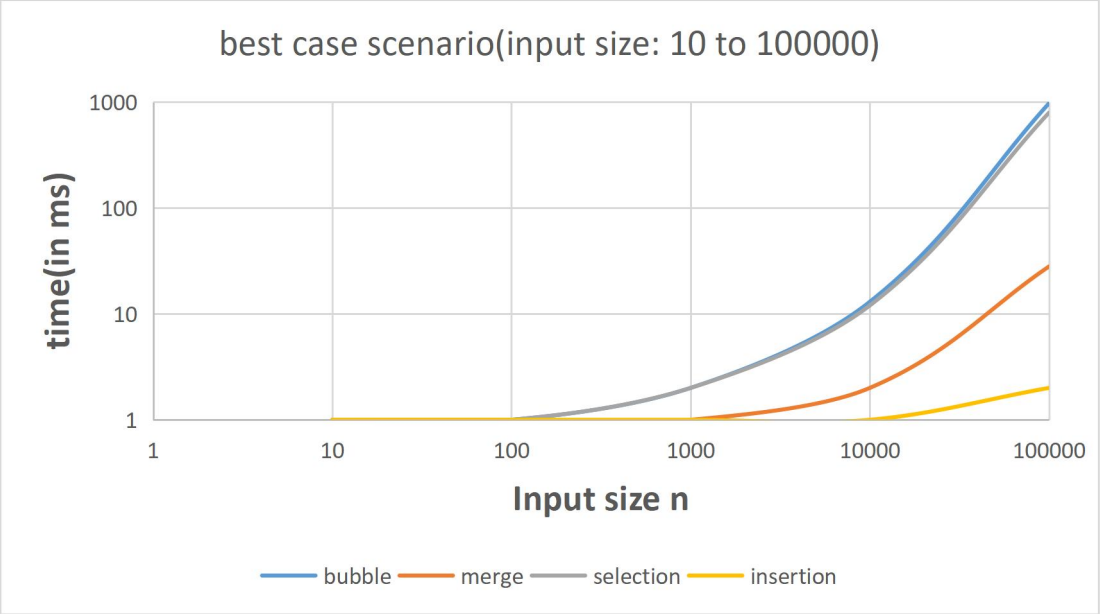
2.Best case scenario:

The best case scenario happens when the user inputs “ascending” in the beginning.

Data collected for best case scenario:

algo \ size	10	100	1000	10000	100000	1000000
bubble sort	0 ms	0 ms	2 ms	13 ms	972 ms	96107 ms
merge sort	0 ms	0 ms	0 ms	2 ms	28 ms	82 ms
selection sort	0 ms	0 ms	2 ms	12 ms	792 ms	75105 ms
insertion sort	0 ms	0 ms	0 ms	1 ms	2 ms	4 ms



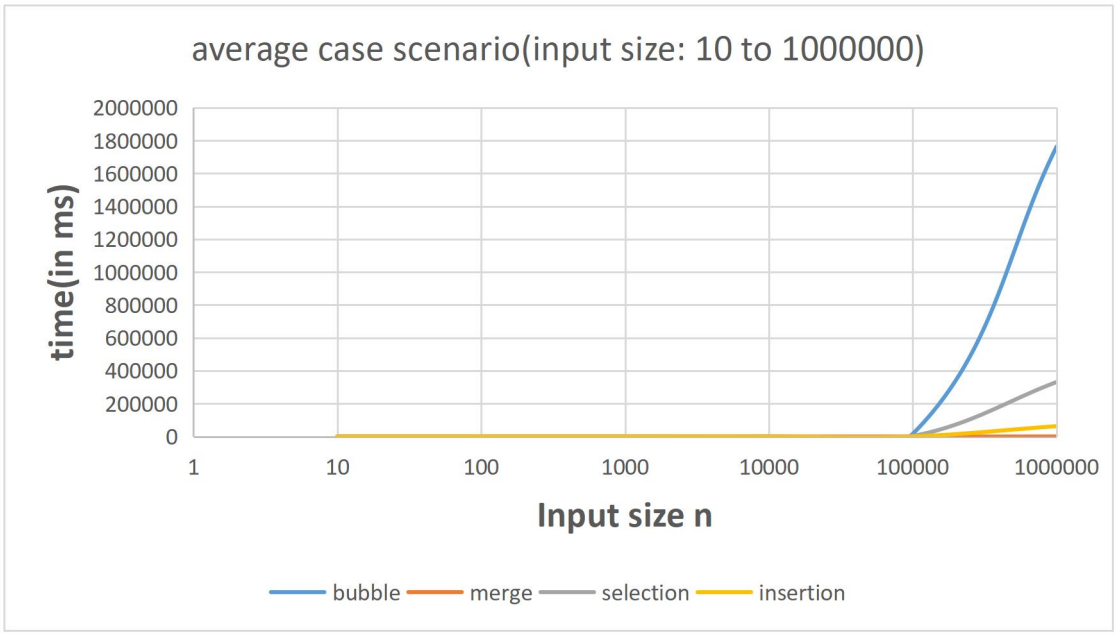


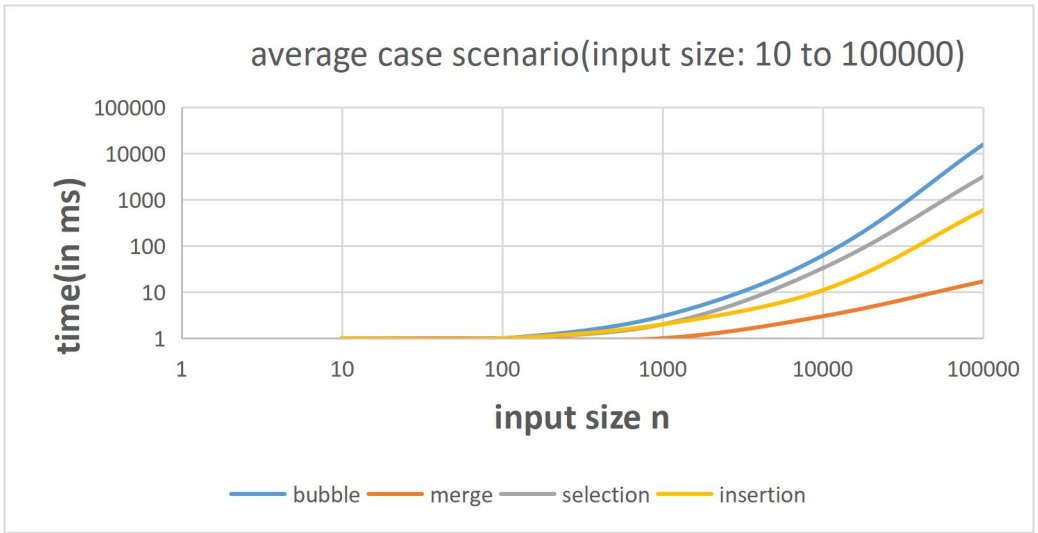
3.Average case scenario:

The average case scenario happens when the user inputs “random” in the beginning.

Data collected for average case scenario:

algo \ size	10	100	1000	10000	100000	1000000
bubble sort	0 ms	0 ms	3 ms	62 ms	15681 ms	1762572 ms
merge sort	0 ms	0 ms	1 ms	3 ms	17 ms	150 ms
selection sort	0 ms	0 ms	2 ms	33 ms	3165 ms	331250 ms
insertion sort	0 ms	0 ms	2 ms	11 ms	591 ms	63023 ms





Question 2: You will write the complexity of the algorithm. You can write what we discussed in the class.

We have implemented four sorting algorithms (merge, selection, bubble, insertion) and each algorithm will encounter three case scenarios (random, ascending, descending). Therefore, there are 12 cases scenarios in total.

Time complexity of these four algorithms under different case scenarios:

	best case	average case	worst case
bubbles sort	O(n)	O(n^2)	O(n^2)
merge sort	O(nlogn)	O(nlogn)	O(nlogn)
selection sort	O(n^2)	O(n^2)	O(n^2)
insertion sort	O(n)	O(n^2)	O(n^2)

Question 3: Interpretation. What does the empirical data and complexity analysis tell us about the algorithms?How do the algorithms compare with each other? Does the order of input matter?

No matter what case scenario is, all of these sorting algorithms run very fast if the user inputs an integer less than 1000. There's almost no difference in efficiency among them.

However, by looking at the graph we made before, we can see that the cut-off point occurs if the user inputs an integer larger than 10000. Those sorting algorithms have completely different performances when size(n) is larger than 10000. The order of input really matters. Below are my analysis.

Analysis of merge sort based on empirical data and graph:

With size n becomes larger and larger, merge sort still runs very fast under three different case scenarios. When n equals to 100000 and 1000000, the execution time of merge sort is only slightly higher than the execution time when n is equal 1000. We can say that the increase of n and different case scenarios have trivial effect on the program execution efficiency. Compared with other sorting algorithms, Undoubtedly merge sort is the best algorithm among them, especially

when n is extremely large.

Analysis of bubble sort based on empirical data and graph:

From the graph, we can see that the bubble sort is very efficiency when n less than 10000. However, bubble start to run slower when n equals to 100000 under worst and average case scenarios, especially in the average case scenario. It takes me nearly 30 minutes to run the whole program when I input 1000000 under the average scenario. Compared with other sorting algorithms, bubble sort is slowest one.

Analysis of selection sort based on empirical data and graph:

Selection sort is very similar to bubble sort in terms of execution time. Selection sort runs very fast when n is less than 10000 and it started to run much slower when n is larger than 1000000 under every case scenarios. However, even though it runs much slower when n is extremely large, it still performs better than bubble sort.

Analysis of insertion sort based on empirical data and graph:

Insertion sort is a special algorithm among those sorting algorithms. When n is less than 10000, insertion sort runs very fast. With n become larger and larger, insertion sort will run much slower under the average and worst scenario(still faster than bubble sort and selection sort). The

interesting point is that insertion sort runs really fast under the best case scenarios no matter how large n is. It runs the fastest among those algorithms under the best case scenario, even faster than the merge sort.

Since merge sort runs much faster than the insertion under the average and worst case scenario, we can say that insertion sort is worse than the merge sort but better than bubble sort and selection sort.

How do algorithms within the same big-O classification compare to each other?

Analysis of these sorting algorithm based on Big-O notation

Worst case scenario： The time complexity of merge sort is O(nlogn), while the other three are all O(n^2). Since O(log n) is much more efficient than O(n^2), so merge sort is the best sorting algorithm. This is also confirmed by the empirical data and our previous analysis .

Although the time complexity of the rest of three algorithms are the same, we can still rank them in terms of efficiency based on the data in the previous graph. In the worst scenario, bubble sort is the worst and insertion sort is the best.

Average case scenario:

This scenario is similar to the worst scenario. Merger sort has the best time complexity, which is O(n logn). This can also be confirmed by the graph. The rest of sorting algorithm all have the time complexity of O(n^2). From the previous graph we can know that insertion sort is the best and bubble sort is the worst.

Best case scenario:

In the best case scenario, merge sort is still the best because its time complexity is O(nlogn). The time complexity of insertion sort and bubble are O(n) and the time complexity of selection sort is O(n^2). If we merely look at the Big-O notation, bubble sort is more efficient than selection. However, to my surprise, although bubble sort has a better time complexity than selection sort, its actual running time is longer than selection time. It can be confirmed by the empirical data.

Conclusion: By combining the data(graph) analysis and Big-O notation analysis, we can rank them from best to worst:

Merge sort > insertion sort > selection sort > bubble sort

Question 4:Conclusions. What algorithms are appropriate for practical use when sorting either small or large amounts of data? Are there any limitations with any of the algorithms? How does your analysis support these conclusions?

If we want to sort an array with small inputs, we can choose either one because all the sorting algorithm run very fast for small inputs. If we want to sort an array with huge inputs, just don't hesitate to choose merge sort because merge sort has a better time complexity and it runs much faster than others. The limitation of merger sort is that it is an external sorting algorithm which requires extra space.

if we know that we have to sort an ascending order with huge inputs, we can choose to use insertion sort. However, this situation rarely happens because sorting an ascending array is just a waste of time and completely meaningless.

We should try to avoid using bubble sort because it has confirmed to be the worst sorting algorithm in our previous analysis. Also, selection sort is also a bad choice when it comes to sort an array with huge inputs.

Citations:

https://medium.com/javarevisited/sorting-algorithms-slowest-to-fastest-a9f0e30937b9