Homework 1: Sorting Algorithms

Jackie Gan

October 2022

1 Introduction

A sorting algorithm is an algorithm[1] that uses the given array to rearrange by comparing other elements in the array and output in a certain order such as increment, decrement or others. Different types of algorithms has different types of techniques to optimize the run time such as time and space complexity. In this homework, we will discuss about Insertion Sort, Merge Sort, Quick Sort(Lomuto), Quick Sort(Hoare), and Counting Sort then analyze their time and space complexity with our theoretical and empirical result.

2 Analysis

Algorithm	Best	Average	Worst	Space
Insertion Sort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Merge Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$
Quick Sort(Hoare)	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(\log n)$
Quick Sort(Lomuto)	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$
Counting Sort	$\mathcal{O}(n+k)$	$\mathcal{O}(n+k)$	$\mathcal{O}(n+k)$	$\mathcal{O}(n+k)$

Figure 1. n = size of array. k = size of 0 to max(n) array. Theoretical Results of Best, Average, Worst case time complexity and space complexity

In this section, we will explain the theoretical big $\mathcal{O}$ of Insertion Sort[3], Merge Sort[4], Quick Sort(Lomuto)[5], Quick Sort(Hoare)[5], and Counting Sort[2] In insertion sort, we start out with a for loop that iterates through length of A which is O(n) then we get a key at the ith element and use a while loop to compare all the elements of the key. This would result in a $\mathcal{O}(n * n)$ implies $\mathcal{O}(n^2)$. In the worst case or the average case we have to check all of the numbers 1 by 1 with a for loop followed by a while loop. However, in best case when the array is sorted, the while loop would not execute because A[j] is already greater than key. Also the space complexity is $\mathcal{O}(1)$ since no extra memory is allocated. Merge sort is a recursive divide and conquer algorithms which is different from the iterative algorithm. In a divide and conquer algorithm we will split the array into smaller parts then sort them individually and combine them together.

In merge sort we will split the array in half into many parts until there is one or less than one elements in the array, so the space complexity is $\mathcal{O}(n)$. Then we sort them and combine them together, this is a stable algorithms because it will always split into n/2 and the depth of the recursive tree is $\log n + 1$. Thus $\log n + 1 * n/2$ is $\mathcal{O}(n \log n)$. Quick Sort Hoare and Lomuto are both a recursive divide conquer algorithm. The problem with quicksort is that, if we choose a pivot that is not good like the highest or lowest element in the array, the partition will use the pivot compare all the elements. This is will very expensive overtime and it's $\mathcal{O}(n^2)$ as the worst case. In the best or average case, when we choose the pivot that is the median element or close to the median element, quick sort will only be $n/2$ or $n/4$ for close to median element in the sub array. Then the best case will be $\mathcal{O}(n \log n)$ like merge sort. The average case will also be $\mathcal{O}(n \log n)$ because $c * \log n$ depth with n partition. The space complexity is $\mathcal{O}(n)$ for Lomuto since it will require to allocate memory for every array when we pick a bad pivot. In Hoare's case, because the partition does less comparisons since it balances the partition swaps by checking both left and right index. Thus the space complexity is $\mathcal{O}(\log n)$. Quick sort is not a stable algorithm and I expect the simulation results can vary between best and average case. In counting sort, the for loops in counting sort doesn't interfere with other loops, but we do initialize a new counting array that is from 0 to max element of A. So the $\mathcal{O}(n + k)$ and the space complexity is $\mathcal{O}(n + k)$ since it need to allocate an counting array that is size $n + k$. As the array size increases the better big $\mathcal{O}$ results.

3 Results

List Size(n)	Insertion	Merge	Quick Hoare	Quick Lomuto	Counting
10	8.1062×10^{-6}	2.0981×10^{-5}	1.1682×10^{-5}	7.1526×10^{-6}	7.1525×10^{-6}
1000	2.7447×10^{-2}	2.2237×10^{-3}	1.1761×10^{-3}	1.1032×10^{-3}	2.6298×10^{-4}
5000	4.4062×10^{-1}	9.0971×10^{-3}	5.6059×10^{-3}	5.4331×10^{-3}	9.8991×10^{-4}
10000	1.8077	1.9256×10^{-2}	1.1403×10^{-2}	1.1071×10^{-2}	2.0120×10^{-3}
20000	7.1604	4.1204×10^{-2}	2.5288×10^{-2}	2.3975×10^{-2}	3.9248×10^{-3}

Figure 2. Simulation results. The units are in seconds. The significant figures is 5, for more details please check text file.

In homework 1, we have implement 5 different types of sorting algorithms. This includes Insertion Sort, Merge Sort, Quick Sort(Lomuto), Quick Sort(Hoare), and Counting Sort. After implementation, I ran a simulation of different sizes of the array from smallest of 10 to 20000. The outputs results showed the amount of seconds it took to sort the randomized list. This implies the lower the seconds the better faster the algorithms sorted the list.

4 Conclusion

In this section, we will use the expected results from simulations and calculate actual time. We will choose n = 20000 as our baseline to calculate all of them because the bigger the array the better the results due to larger sample size.

$$(\frac{size1}{size2})big\mathcal{O} = \frac{size1time}{size2time} \tag{1}$$

The equation will use average case big$\mathcal{O}$.

4.1 Insertion sort

$$(\frac{size1}{size2})^2 * (size2time) = size1time \tag{2}$$

n	Actual	Expected
20000	7.1604	7.1604
10000	1.8077	1.7901
5000	4.4062×10^{-1}	4.4753×10^{-1}
1000	2.7447×10^{-2}	1.7901×10^{-2}
10	8.1062×10^{-6}	1.7900×10^{-6}

Table 1: The average case is $\mathcal{O}(n^2)$. The units are in seconds.

4.2 Merge Sort

$$(\frac{size1}{size2}) * \log(\frac{size1}{size2}) * (size2time) = size1time \tag{3}$$

n	Actual	Expected
20000	4.1204×10^{-2}	4.1204×10^{-2}
10000	1.9256×10^{-2}	1.0301×10^{-2}
5000	9.0971×10^{-3}	2.5753×10^{-3}
1000	2.2237×10^{-3}	1.0301×10^{-4}
10	2.0981×10^{-5}	1.0300×10^{-8}

Table 2: The average case is $\mathcal{O}(n \log n)$. The units are in seconds.

4.3 Quick Sort Hoare

$$(\frac{size1}{size2}) * \log(\frac{size1}{size2}) * (size2time) = size1time \tag{4}$$

n	Actual	Expected
20000	2.5288×10^{-2}	2.5288×10^{-2}
10000	1.1403×10^{-2}	6.3220×10^{-3}
5000	5.6059×10^{-3}	1.5805×10^{-3}
1000	1.1761×10^{-3}	6.3220×10^{-5}
10	1.1682×10^{-5}	6.3220×10^{-9}

Table 3: The average case is $\mathcal{O}(n \log n)$. The units are in seconds.

4.4 Quick Sort Lomuto

$$(\frac{size1}{size2}) * \log(\frac{size1}{size2}) * (size2time) = size1time \tag{5}$$

n	Actual	Expected
20000	2.3975×10^{-2}	2.3975×10^{-2}
10000	1.1071×10^{-2}	5.9938×10^{-3}
5000	5.4331×10^{-3}	1.4984×10^{-3}
1000	1.1032×10^{-3}	5.9938×10^{-5}
10	7.1526×10^{-6}	5.9938×10^{-9}

Table 4: The average case is $\mathcal{O}(n \log n)$. The units are in seconds.

4.5 Counting Sort

$$(\frac{size1}{size2}) * (size2time) = size1time \tag{6}$$

n	Actual	Expected
20000	3.9248×10^{-3}	3.9248×10^{-3}
10000	2.0120×10^{-3}	1.9624×10^{-3}
5000	9.8991×10^{-4}	9.8120×10^{-4}
1000	2.6298×10^{-4}	1.9624×10^{-4}
10	7.1525×10^{-6}	1.9624×10^{-6}

Table 5: The average case is $\mathcal{O}(n + k)$ but we will treat it like $\mathcal{O}(n)$. The units are in seconds.

4.6 Final Conclusion

From our calculation we can tell that as we decrease the random list size n and the more it differs form the actual results. This is because in a large n we

have a larger sample size which will yield a better results. The smaller the size more likely the random generated list will be sorted or need less sorting. The calculating expected time works best with iterative algorithms such as counting sort and insertion sort because they are extremely stable. Then as in recursive algorithms, merge sort is also very stable compared to quick sort. Both Hoare's and Lomuto's quick sort are not as stable as the because it requires us to pick a certain pivot for the performance. The algorithm from Hoare's and Lomuto's became unstable when the n is 1000. We can conclude the Big O is theoretical did match the expected values.

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[2] Wikipedia. Counting sort — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Counting%20sortoldid=1076101343, 2022. [Online; accessed 11-October-2022].

[3] Wikipedia. Insertion sort — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Insertion%20sortoldid=1114957990, 2022. [Online; accessed 11-October-2022].

[4] Wikipedia. Merge sort — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Merge%20sortoldid=1114193283, 2022. [Online; accessed 11-October-2022].

[5] Wikipedia. Quicksort — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Quicksortoldid=1111427952, 2022. [Online; accessed 11-October-2022].