Joey Troyer

Lab 3

Vishwanathan Roopa

CS 372

27 February 2023

INTRODUCTION:

In this lab, this objective was to gain a deeper understanding of the sorting algorithms insertion sort and quick sort. Our goal was to analyze how these algorithms perform differently, keeping track of their sorting times when presented with pre-sorted, reverse-case sorted and fixed arrays. By closely monitoring these differences, we aimed to further our knowledge and comprehension of sorting algorithms.

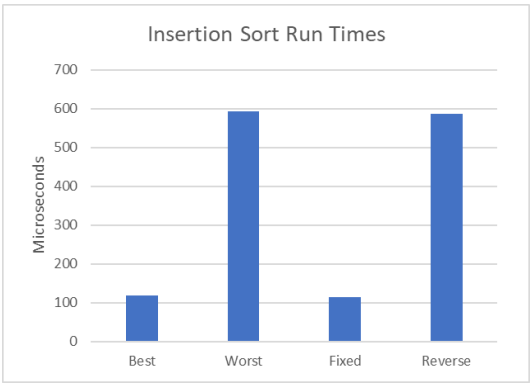
RESULTS:

First, we will start with the results from the insertion sort

```
jtroyer@brooks:~/CS372/Lab3> python3 insertion.py
Verifying sorted lists:
✅ Best Case
✅ Worst Case
✅ Fixed Case
✅ Reverse Case

Size = 1000
Testing each function 100 times

Best Case      AVG TIME: 118.86 micro seconds
Worst Case     AVG TIME: 591.66 micro seconds
Fixed Case     AVG TIME: 114.26 micro seconds
Reverse Case   AVG TIME: 585.81 micro seconds
```



These are the results of the run times for insertion sort. I sorted arrays of size 1000 and sorted each 100 times. I then got the avg time of these 100 runs, which is displayed in the graph and the terminal

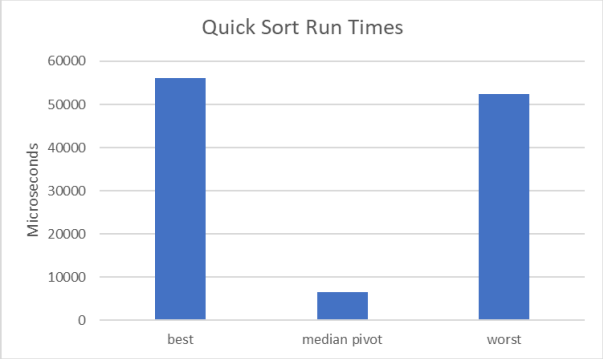
above. The results show that Insertion Sort performs the best in the Fixed case scenario, with an average execution time of 114.26 microseconds. This is followed by the Best case scenario, where Insertion Sort takes an average of 118.86 microseconds to execute. The Reverse case scenario takes an average of 585.81 microseconds to execute, which is longer than the Worst case scenario, where Insertion Sort takes an average of 591.66 microseconds to execute. The reason for the longer execution time in the Worst case scenario is that every element in the input array needs to be moved to its correct position, resulting in more comparisons and swaps. In the Reverse case scenario, the input array is already sorted in reverse order, which makes every element in the array need to be moved to the beginning of the list, resulting in more comparisons and swaps as well.

Now we'll look at the quick sort algorithm

```
jtroyer@brooks:~/CS372/Lab3> python3 quickSort.py
Verifying sorted lists:
✅ Best Case
✅ Median Pivot
✅ Worst Case

Size = 1000
Testing each function 10 times

Best Case       AVG TIME: 56065.99 micro seconds
Median pivot    AVG TIME: 6466.35 micro seconds
Worst Case      AVG TIME: 52433.07 micro seconds
```



For quicksort I tested each case with an array of size 1000 I then ran each case 10 times and averaged their run times which are shown in the chart and terminal above. Comparing the results for "Best case" and "worst case". It seems that the best case actually takes the most time. So a presorted array takes longer than a reverse-sorted array. This is interesting, but it makes sense because with a presorted array, it would divide the array into two subarrays, one of which is empty and the other of which contains all the elements of the array except for the pivot.. The worst case wasn't too much faster than "best case" but could still be a significant time difference

on a large scale. We can see that by far the actual best case is using the median pivot. I believe this is because if we choose the median as the pivot, the algorithm will split the sub-arrays in more or less equal sizes. While if we choose a bad pivot, we could end up with sub-arrays of vastly different sizes, which will cause more comparisons to be made.

CONCLUSION:

The best-case time complexity of Quicksort is O(n log n), where n is the size of the input array. The worst-case time complexity of Quicksort is O(n^2), which occurs when the input array is already sorted or almost sorted. For Insertion Sort, the best-case time complexity is O(n), which occurs when the input array is already sorted, while its worst-case time complexity is O(n^2), which occurs when the input array is in reverse order. When we compare the results of running Insertion Sort and Quicksort with the asymptotic bounds we saw in class, we can see that the algorithm's performance matches our expectations. Quicksort seems to generally have a better average and best-case performance than Insertion Sort, both algorithms have a similar worst-case performance. Based on the result, insertion sort performed much better but I believe that is only true with smaller arrays, and if we used a much larger array, quicksort would sort much faster. In the end, they are both valid sorting algorithms, and deciding to use them will depend heavily on their use case.