Name: Muhammad Hassan

Student number: 19408114

Written report

1.Introduction

The algorithms I chose were bubble sort, selection sort and insertion sort.

The selection sort algorithm sorts the array by finding the minimum element from the part that is unsorted and add it to the beginning of the array it continues to do this until the full array is sorted. I chose selection sort because it of the time complexity 0(n^2).

The bubble sort algorithm is an algorithm that goes through the list and compares adjacent integers and swaps them if they are in the wrong order. I chose bubble sort because in its best case when the array is already sorted it is o(n) time complexity.

The insertion sort algorithm is an algorithm that spits the array in to a sorted an unsorted part. Integers from the unsorted part and placed in the sorted. I chose this algorithm because it has time complexity of 0(n^2).

2.Datasets

The dataset I chose to test the algorithms on was a random number generator.in my opinion this was the best way to test the true performance of the algorithms as it would not generate two of the same numbers beside ach other also it was better than a reverse list as it would show the true potential of the algorithm. I used malloc because I was unable to test over 100,000 when I didn't have malloc in my program. I was creative because my number generator generates truly random numbers and does not have a cap on how big the number can be.

3.Algoritm performance

The selection sort algorithm was very quick when I tested it for 10 randomly generated numbers. It took less than a second. For 1000 randomly generated numbers it took 1 second.

For 100000 randomly generated it took 10 seconds. I could tell from this point as the numbers increased the algorithm got slower. It took 15 minutes to sort 1 million numbers. The time increased to 2 hours for 2 million numbers and it increased again for 5 million.

When tested on a sorted list insertion sort did not have that big of difference as I would have thought. It took less than a second to test 10 numbers and took up to 12 to test a

million numbers. I concluded that that the insertion sort algorithm best and worst-case time complexity is 0(n^2).

The insertion sort algorithm was very quick when tested on 10 numbers that were randomly generated. It took less than a second. for 1000 words it took 1 second. For 100000 numbers that were randomly generated it took 10 seconds. For one million it took 13 minutes and for 5 million it took 4 hours. I tested insertion sort on an unsorted list and a sorted list. when it was tested on an unsorted list it took longer than when it was tested on a sorted list this was because its time complexity when tested on an unsorted list is o(n^2) and when it is tested on a sorted list its time complexity was its best case which was o(n) time complexity.

When the sorted list was tested on 100000 numbers it half the time of the unsorted (5 seconds). Insertion sort algorithm works best when there is small number or there is a sorted list.

The bubble sort algorithm was very quick when tested on 10 it did in less than a second. It sorted 1000 in a second. It slowed down when it had to sort 100000 numbers it took 30 seconds. When it had to sort 1 million it slowed down a lot it did in 25 minutes. The bubble sort algorithm when tested on a sorted list became a lot quicker because that was its best-case scenario time complexity which when tested on a sorted list it is 0(n). whereas when it is tested on an unsorted list it is o(n^2). The bubble sort algorithm works best with small number or sorted lists.

4. negatives

I feel I should have tested the algorithm performance on a lot of different data sets. This would have given me abetter idea of how well the data set adapts to different types of testing. Also, at first my random number generator could not test numbers bigger than a 100000 then I had to implement malloc into my program in order to be able to sort large numbers. While sorting the larger numbers the algorithms took a lot longer. But as I have learned researching that the slower the algorithm the less ram it uses it is a positive and a disadvantage.

5. conclusion and future work

If I had more time , I would have chosen to do algorithms with different time complexities and I would have tested my algorithms on a lot of different data types more than 4 so that I would to gasp an understanding of how well the algorithm actually works. I would have also tried to make a better random number generator that I do have currently so that I would be easier to test the algorithms.in conclusion I learned lot about time complexity and how different algorism are used in different scenarios.