Assignment 4 Writeup

Pehara Vidanagamachchi

October 2022

1 Questions

- **What you learned from the different sorting algorithms? Under what conditions do sorts perform well? Under what conditions do sorts perform poorly? What conclusions can you make from your findings? -** Throughout this assignment, I have been able to familiarize myself with the several different sorting algorithms I encountered. I have been able to learn the different methods of sorting such as bubble sort which iterates through all the elements and swaps them around till they're in order from smallest to biggest. Following that there is heap sort where it searches for the smallest element and orders them from the desired parameters. It iterates through the code from the minimum to the maximum. There is also the quick sort which is also known as the partition-exchange sort is a sorting algorithm that follows a divide-and-conquer principle, it is executed by spitting larger arrays into small arrays and running shell sort if the value is below eight. Finally, the shell sort algorithm works by sorting out elements far apart from each other so it moves elements in front until they are all in the right place, by working between the gaps it is able to organize all the elements. I have found that sorts perform best when given data that puts it in the desired order. Each algorithm, however, has its own set of preferences such as bubble sort which works well with large data sets where they are able to be sorted in a single iteration. So it runs fastest on a small or close to the sorted set. Quick sort on the other hand runs best when the data known is similar and it is able to travel sequentially. While Quick sort is able to run much faster than the other algorithms and works better with arrays. The heap sort algorithm is by far the most inefficient one and is 2-3 times slower than quick sort and is generally seen as an unstable sorting algorithm. And given that all the sorting algorithms run better under different circumstances I was able to conclude that based on the data given some programs will function better than others, and vice versa.

- **Graphs explaining the performance of the sorts on a variety of inputs, such as arrays in reverse order, arrays with a small number of elements, and arrays with a large number of elements.**

Your graphs must be produced using either gnuplot or matplotlib. You will find it helpful to write a script to handle the plotting. As always, awk will be helpful for parsing the output of your program. - I produced different graphs for an array in reverse, an array with a large number of elements and an array with a small number of elements. Doing this I was able to see that regardless of the case they produced similar outputs to their old test cases. To begin with, the backwards array bubble sort took a longer time to generate the results, quicksort on the other hand produced its results much quicker and with a shorter amount of comparisons, Heapsort sat somewhere in the middle. Shell sort ran the slowest of them all. the large array didn't affect bubble sort too differently from its previous runs. Quick sort was able to run extremely quick once again and was effective again. Heapsort was able to run the larger array faster than previously. Shell sort followed the same slower path. the small array also didn't change the speed of bubble sort too much, quick sort like its previous runs was able to run efficiently again with fewer comparisons. Heapsort in this case ran smaller than when it did with the larger list. Overall shell sort was the slowest program of all.

2 Main Program

- **Bubble Sort -** Bubble sort was a sorting algorithm I was tasked with creating that works by swapping elements repeatedly if they are in the correct order. The algorithm is not suitable for large data sets since it has high time complexity. which works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

- **Heap Sort -** Heap sort was a sorting algorithm I was also tasked with creating that proved to be my most difficult algorithm to create. The way that it works is based on first finding the minimum value and placing it at the beginning and repeating the process till they're maxed out. It is one of the slower algorithms.

- **Shell Sort -** Shell sort is a sorting algorithm I created that's a variation of another sorting algorithm known as insertion sort. Shell sort works by sorting the array by reducing the array value till it becomes one. This method is fairly fast and efficient.

- **Quick Sort -** As in the name quick sort, this algorithm was one I had to make that is considered one of the more efficient. The way it works is divide and conquer where it picks the first element as a pivot point and then the last one. It uses partition to iterate through the elements to produce a sorted array.