Lab 3D5A
Hugh Lavery – 14313812
Assignment 2 – Sorting
Monday 14th November

Abstract/Introduction

The Aim of this lab was to implement two sorting algorithms. One of my own choice, I chose bubble sort[1], and quicksort[2]. The sorting algorithms had to then be tested against a set of data and all code had to be commented and explained.

Bubble sort works by comparing each element with its adjacent element and then swaps them (if necessary) putting the lower one on the left. It iterates through the list repeatedly until the whole list is ordered.

Quicksort works by dividing a list into a two smaller lists. One with lower elements and one with higher and sorts each sub list a bit. It then recursively does this until the whole list is fully sorted and then it merges the sub lists back into one list. This is known as a divide and conquer method.

Bubble sort is quite an inefficient method as is has to repeatedly look through the list and compare all elements over and over. Quicksort is quite efficient as it finds a pivot point to split the list and then sorts each sub list a bit.

Method

I implemented Bubble sort first and then quicksort. For the data set I chose to use an array filled with randomly generated numbers using the rand() function and I modulated all the data to 1000 so all the numbers would be between 1-1000. For finding the big O complexity for bubble sort I placed a probe inside the for loop which is nested inside a while loop. For quicksort I placed the probe in the if statement inside the for loop in the partition function.

I did a set of tests with an array of size 1000 and of size 100. I ran each function 10 times and got an average of the amount of probes to find their complexity.

Results/Discussion

As expected the quicksort algorithm performed much better than the bubble sort. Noticeably when the array size was 1000 bubblesorts worst case scenario required over 1,000,000 probes where as quicksorts works care was only 7028.

Test results

	Bubblesort	Quicksort
Array size	1000	1000
Ave Number probes	999700	5655
Worst case	1001000	7028
Best case	942000	5033
Ave O( )	O (n^2)	nlog(n)

	Bubblesort	Quicksort
Array size	100	100
Ave number probes	8,877	323
Worst case	10100	481
Best case	7700	259
Ave O( )	O(n^2)	nlog(n)

On further research [3] I found worst time complexity for quicksort was O(n^2). I believe this would occur if you had a list that was in reverse order, in this situation quicksort would have to iterate many times to fully sort the list. The best/average in O(nlog(n)) as I found. For bubble sort best case scenario was O(n) where as average/worst case is O(n^2) as I have found. Bubble sort would only have time complexity O(n) for a list that was almost entirely sorted where the function would only have to pass over the list once and perform 1 or 2 switches.

Type	Best O( )	Average O( )	Worst O( )
Bubble sort	O(n)	O(n^2)	O(n^2)
Quicksort	O(nlog(n))	O(nlog(n))	O(n^2)

References

[1]- https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm

[2] -
https://interactivepython.org/runestone/static/pythonds/SortSearch/TheQuic
kSort.html
[3] - http://bigocheatsheet.com/