CMPT 225 A4 Report

Gurveen Nanua
301546459

In my experiment the four different types of quicksort algorithms I used were:
1. Using the median of A[lo], A[(lo+hi)/2] and A[hi] as the pivot (A[ ])
2. Using an element chosen uniformly at random as the pivot (B[ ])
3. Special handlings on small sequences, otherwise it will run as a quicksort algorithm like the first one (C[ ])
4. Std::sort (D[ ])

Question: which variation of quicksort runs the fastest on a large input of values?

For my inputs, I used a constant variable SIZE to change the size of the array. (the sizes consist of 1000, 10000, 100000 and 500000) Then I made 4 different arrays consisting of the same elements by inputting random numbers. Each array is labeled A[ ], B[ ], C[ ], D[ ] respectively.

Average runtime for each case:

	A	B	C	D
SIZE = 1000	0.0001626	0.0001640	0.00090	4.21943*10^-5
SIZE = 10000	0.0021159	0.0019401	0.08261	0.000995195
SIZE = 100000	0.0237173	0.0170441	8.14841	0.007178293
SIZE = 500000	0.095959	0.0994863	203.361	0.04115

In the results above, when the SIZE was 1000 there was very little difference between A and B. As the SIZE increased, it was clear that array A, (using the median of A[lo], A[(lo+hi)/2] and A[hi] as the pivot) was slower. Array B, (using an element chosen uniformly at random as the pivot, varied in runtimes) sometimes was as slow as A, or as D. Lastly, C was the slowest variation, for SIZE = 500000, it was extremely slow, at about 203 seconds. So I concluded it was unreliable. Array D, using std::sort was by far the fastest for all sizes.

In conclusion, C had the slowest runtime of the algorithms I coded when the size was significantly large (such as SIZE = 100000 and SIZE = 500000). Therefore, using special handlings on small sequences, where constant K = 100, and then implementing a quicksort algorithm by taking the median of C[lo], C[(lo+hi)/2] and C[hi] as the pivot, is the least ideal. To answer the question, using std::sort will give the overall fastest result on a large input of values.