Report on the running time of algorithms

Written by: Junwei Yang, Student Number: 667275

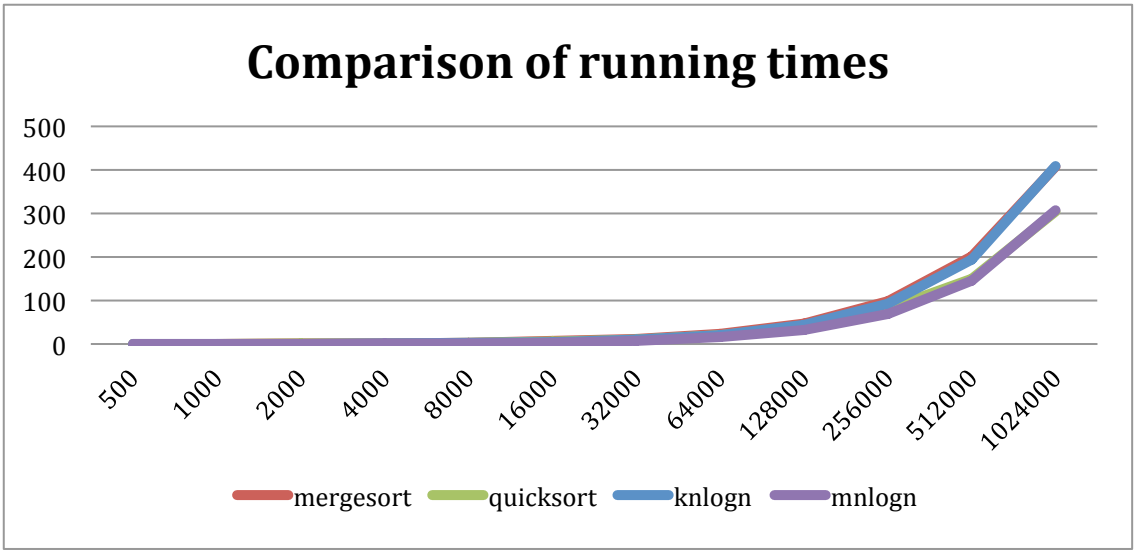
For the project, the basic requirement is to implement quick sort and merge sort algorithm with an explicit stack. Moreover, explanation of the performance based on running times is also needed. Therefore, what will be covered in this report contains the graph of running times on random inputs of increasing size, an analysis of running times for each program combining the result from the graph, and the discussion for the performance of both programs under different cases.

Some clarifications are needed to ensure the data accuracy. Firstly, the test was conducted under the environment of Terminal using MacBook Pro 13'' (Mid 2012). Then, the numbers for the test were generated randomly, and the running time is calculated using the average for ten times for each quantity of numbers.

numbers	500	1000	2000	4000	8000	16000	32000	64000	128000	256000	512000	1024000
mergesort	0	0	1	2	3.1	7.1	12	23.6	47.3	98.1	200.8	407.1
quicksort	0	0	1	2	3	5.6	9.9	20	39.3	80.9	150.6	304.5
knlogn	0.1	0.2	0.4	1.0	2.1	4.5	9.6	20.4	43.4	92.0	194.2	408.9
mnlogn	0.1	0.1	0.3	0.7	1.6	3.4	7.2	15.3	32.6	69.0	145.7	306.7

(Where $k = 2\times10^{-5}$, $m = 1.5\times10^{-5}$, time unit is ms)

In the table above, time taken for each sorting algorithm as well as the nlogn function for comparison is shown. Turning the data into a graph gives us:



As it can be seen from the graph, the actual running times fit quite well with the nlogn function. Therefore the assumption could be made that the running time for each program is O(nlogn), and more analysis based on algorithms will be presented to prove the assumption.

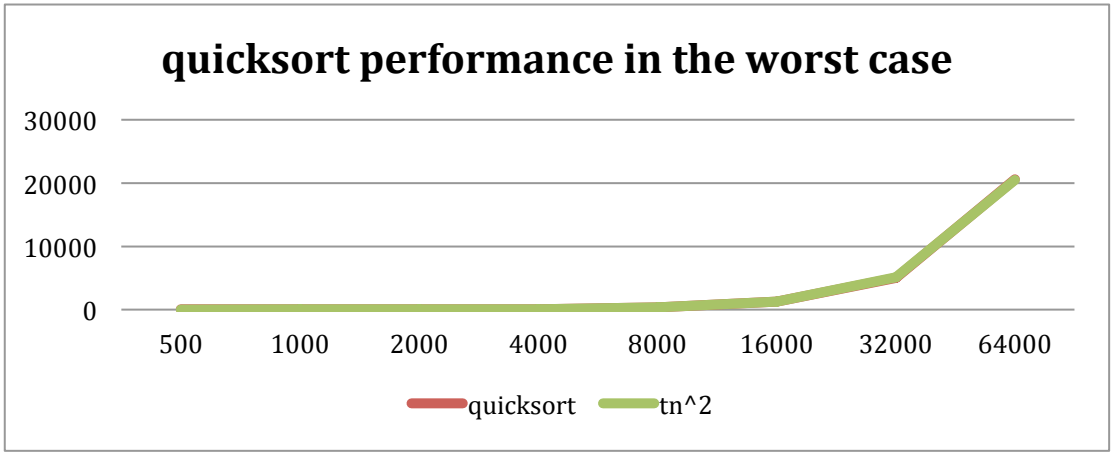
Starting with the partition process, any item less than the pivot will be moved to the left part and those not less than the pivot will be moved to the right part by two cursers. The process ended when the two cursers meet, so the running time would be O(n) as the number of items that cursers pass through is n. After that, for the quicksort process, the basic idea of the program is just as the same as the normal quicksort, the difference is whether using a stack to store recursion information or not. Therefore the recurrence is T(n) = T(n-k) +T(k) + O(n), where k is the size of left part. In the average case, when k= 1/2, according to the master theorem, the running time is O(nlogn).

Then for the merge sort process, the idea of the program is to divide the array into two parts recursively until there is only one item in a part, and merge every part back into the original array. However, the running time for the merge process is O(n) for the reason that n numbers have to be compared to be sorted. Therefore the recurrence is T(n) = T(n/2)+T(n/2) +O(n), and the running time is O(nlogn) by the master theorem.

Under different cases the performance of each algorithm can vary. For example, an array is to be sorted using quicksort. If for every time the pivot chosen is the largest item in the array in reversed order, the recurrence is T(n) = T(n-1) + T(1) + O(n) = (T(n-2)+T(1)+α(n-1)) + T(1) + αn = … = T(n – i) + iT(1) +α(n – i + 1 + … + n – 2 +n – 1 + n) = nT(1) +α(1 + 2 + … +n – 1 + n) = nT(1) +$\frac{\alpha(1+n)n}{2}$, which is O(n^2), and the result is shown below. As for the merge sort, the worst case is also O(nlogn) by the recurrence, the only difference is the times for comparison.

numbers	500	1000	2000	4000	8000	16000	32000	64000
quicksort	2	5.9	21.8	82.1	320.9	1259	5031.2	20573
tn^2	1.25	5	20	80	320	1280	5120	20480

(Where t = 5×10^{-6}, time unit is ms)



In conclusion, the running time on average is proved to be O(nlogn), and as it can be seen from the graph with basic fitting, what has been proved is correct. The running time for the program based on quicksort algorithm is O(nlogn) on average and O(n^2) in worst case, and the running time for the one using merge sort is O(nlogn).