알고리즘설계와분석 Homework#1

ICT융합학부 2018045841 정대윤

1. Quicksort

Number 1-1 is to pivot the first element to turn the quicksort alignment. Number 1-2 is to pivot a randomly selected value between low and high to rotate the quicksort alignment. Number 1-3 is to set an intermediate value between low and high, set the value to pivot, and turn the quicksort alignment. When I saw the question before starting the task, I thought it would be the fastest way to turn the quicksort alignment by setting the middle value between low and high and setting the value to pivot, and I saw the result by turning the code to the slowest way to turn the quicksort alignment by pivot. In all three cases, the number of elements was executed at the value given by the professor. I recorded the duration of each sort up to 6 decimal places

1-1) Pivot is the first one

Sorting Type	Quicksort	
n	Comparison Exchange	Time(ms)
100	604	0.000000 ms
200	1547	0.000000 ms
500	4802	0.000000 ms
1000	10132	0.000000 ms
2000	23286	0.001074 ms
3000	39539	0.001995 ms
4000	54425	0.004634 ms
5000	73872	0.006056 ms

1-2) Pivot is a randomly selected value between low and high

Sorting Type	Quicksort	
n	Comparison Exchange	Time(ms)
100	938	0.000000 ms
200	2530	0.000000 ms
500	9815	0.001054 ms
1000	27158	0.001997 ms
2000	78883	0.004992 ms
3000	123525	0.009509 ms

4000	220072	0.013037 ms
5000	291248	0.017866 ms

1-3) Pivot is set an intermediate value between low and high

Sorting Type	Quicksort	
n	Comparison Exchange	Time(ms)
100	699	0.000000 ms
200	1533	0.000000 ms
500	4566	0.000000 ms
1000	12052	0.001008 ms
2000	25546	0.001986 ms
3000	40662	0.002986 ms
4000	54617	0.004011 ms
5000	72672	0.004981 ms

2. Heapsort

The Heapsort compares the number of comparisons and the execution time according to the number of n. The execution time was expected to be O(nlgn) before performing. I recorded the duration of each sort up to 6 decimal places

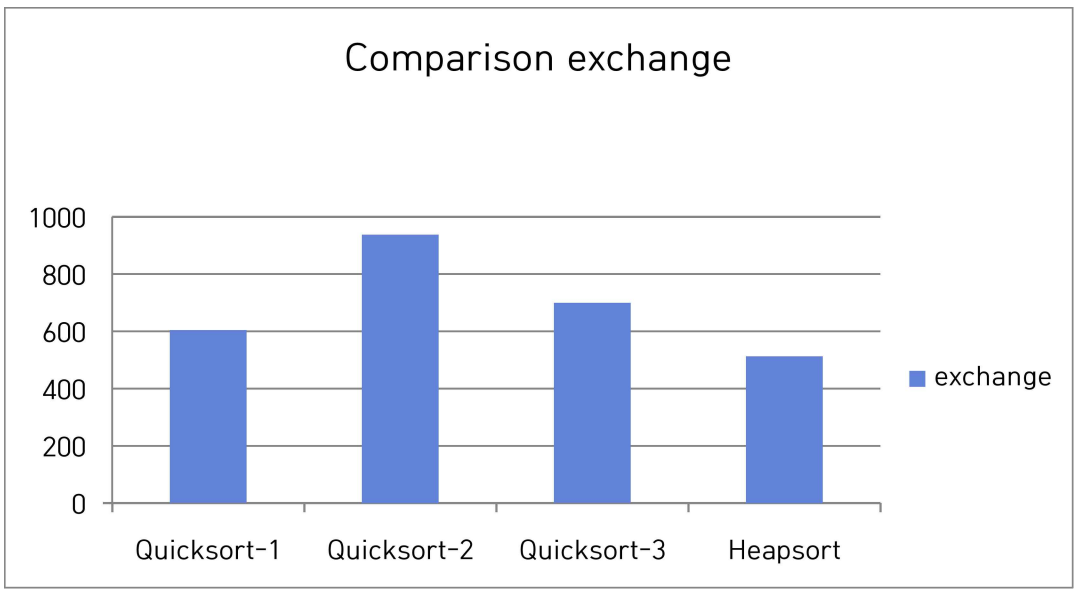
Sorting Type	Heapsort	
n	Comparison Exchange	Time(ms)
100	513	0.000000 ms
200	1428	0.000000 ms
500	4082	0.000997 ms
1000	8436	0.002006 ms
2000	21721	0.004983 ms
3000	32652	0.006656 ms
4000	47863	0.010886 ms
5000	62438	0.012051 ms

3. Quicksort and Heapsort Comparison

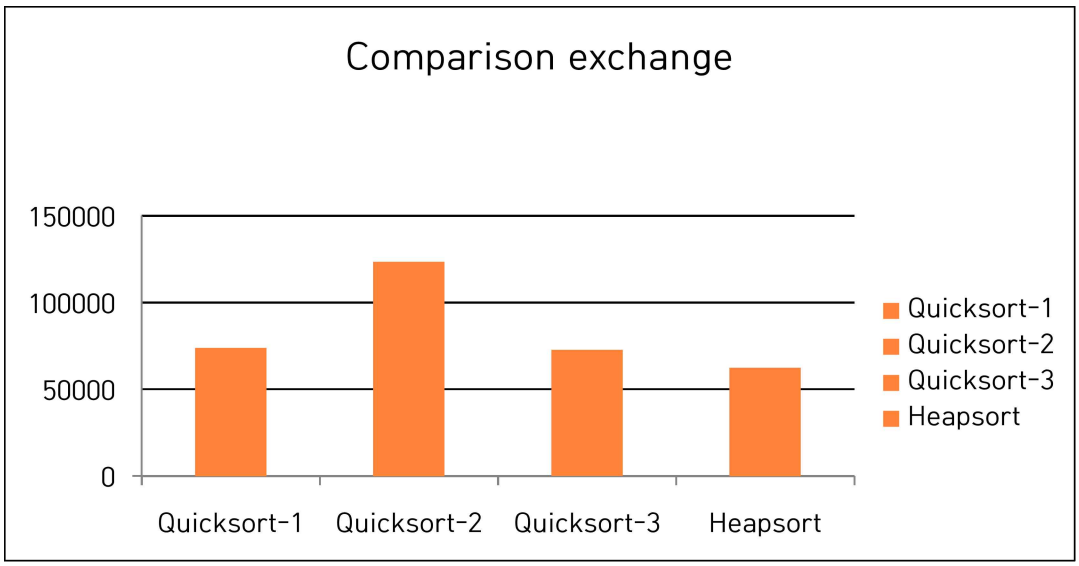
The results of the Quicksort and Heapsort above are shown in a graph.

Indicates when n is 100, 5000. I chose the last value and first value. Because I was wondering what would happen depending on the difference in n.

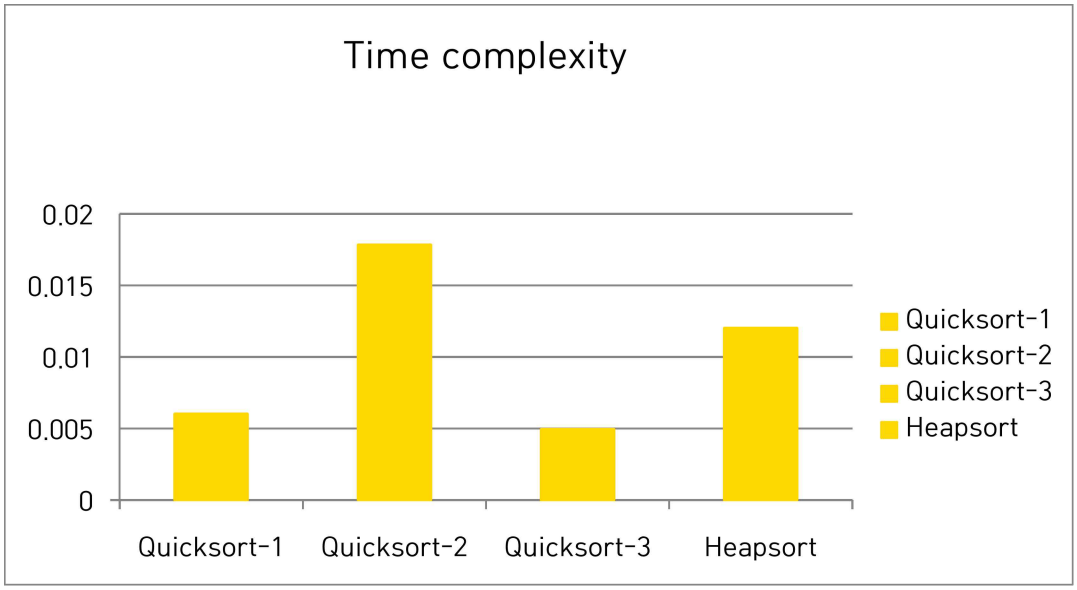
(1) n is 100.



(2) n is 5000.



(3) n is 5000(When n is low, they are all close to 0 ms, so they are not marked)



When I turned the codes for Quicksort and Heapsort, I didn't feel much difference when the value of n was small. Overall, we compared the code about 100 times and found that quicksort comparison frequency and time complexity were much better. And I found that quicksort also produces faster results than heapsort. We found that the three results completely change depending on how you position the pivot value. I thought it was the fastest when the pivot was located in the middle. However, the pivot value was similar to the one in the middle. In some cases, number 1 showed a faster value, and in some cases number 3 showed a faster value. In the case of number 2, I got the result that it is the slowest when it is set randomly. I learned about heapsort, which is a perfect alignment in class, but when I studied, I didn't know why it was good, but when I checked it by making the code myself, I found that the time and number of comparisons of heapsort were ridiculously small.

I also thought about how Quicksort and Heapsort could be efficient. QuickSort had a really big difference in performance depending on how to set the pivot value. So what I think is that you get the index and you get the average of the index. After adding the average value to the array, I thought it would be better to set it as a pivot and turn the quicksort. The addition of one more value to the array did not seem to increase the time complexity. And I thought about hip sort. Hipsault is a good algorithm, so I thought the direction I thought might be worse. In my opinion, in order to become a heap sort, it must be an ECBT with two child nodes. If the number of child nodes from the test question, not the binary tree, increases to three or four, I think we can store and represent more data.
That's it. Thank you for reading this long comment.