# BLG335E
# ANALYSIS OF ALGORITHMS I
# FALL 2021

## ASSIGNMENT #1 QUICKSORT REPORT

ŞERİFE DAMLA KONUR
040160433

CRN : 11458
Date of Delivery : 25.11.2021

In this assignment, we are expected to implement randomized and deterministic quicksort algorithms, all the codes are written in main.cpp file. Also the compilation and execution commands as follows :

**Compilation : g++ main.cpp**

**Execution : ./a.out numberOfTweet pivotSelectionMode inputFileName outputFileName**

Example :

```
-zsh
(base) damlakonur@Damla-MacBook-Pro 1 % g++ main.cpp
(base) damlakonur@Damla-MacBook-Pro 1 % ./a.out 10000 randomized tweets_unsorted.csv sorted_1000.csv
```

The in-place quicksort algorithm uses divide and conquer paradigm as follows :

**Divide :** Partition the array A[ p . . r ] into two subarrays A[p..q−1]and A[q+1..r]such that each element of A[p .. q−1] is less than or equal to A[q], which is, in turn, less than or equal to each element of A[q + 1 . . r ]. The index q is computed as part of this partitioning procedure.

**Conquer :** Sort the two subarrays A[p..q−1]and A[q+1..r] by recursive calls to quicksort.

- Since the subarrays are sorted in place, no work is needed to combine them: the entire array A[ p . . r ] is now sorted.

The main difference between deterministic and randomized pivot selection algorithms is that instead of always using A[p] as the pivot, a randomly chosen element can be used from the subarray A[ p . . r ]. Then, element A[p] is exchanged with an element chosen at random from A[ p . . r ]. This modification, ensures that the pivot element x = A[p] is equally likely to be any of the r − p + 1 elements in the subarray. Because the pivot element is randomly chosen, the split of the input array is expected to be reasonably well balanced on average.

**Q- 1 :** Write down the asymptotic upper bound for the Quicksort with **deterministic** pivot selection for best case and worst case by solving the recurrence equations.

Quicksort's execution time is determined by whether the partitioning is balanced or unbalanced, and this, in turn, is determined by which elements are utilized for partitioning.

**Worst Case :** The worst case behaviour is observed when partitioning step produced two subproblems with (n-1) and (0) elements. Actually this means that, array is sorted or reversely sorted, partition around min or max element. This kind of partitionings are unbalanced, one side of partititon always has no elements. If this unbalanced partitioning arising for each recursive call, the partitioning costs $\Theta(n)$ time, and the subpart with 0 element costs $\Theta(1)$ time. The recurrence equation of the worst case running time becomes :

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

$$T(n) = T(n-1) + \Theta(1) + \Theta(n)$$

→ This recurrence equation can be solved by substitution method

$$T(n) = T(n-1) + cn$$

$$T(n-1) = T(n-2) + c(n-1)$$

$$T(n-2) = T(n-3) + c(n-2)$$

...

T(0) = 0

$$T(n) = T(1) + 2 + 3 + \cdots c(n-1) + cn \approx \frac{N^2}{2} = O(n^2)$$

**Best Case :** In the best case, partitioning step splits the array evenly. So the recurrence becomes :

$$T(n) = 2T(n/2) + \Theta(n)$$

With the master theorem : a = b = 2 $n^{\log_b a} = n$ f(n) = n

Case 2 : → $O(n.\log_2 n)$

**Q-2 :** Write down the asymptotic upper bound for the Quicksort with randomized pivot se- lection by doing probabilistic analysis.

Partition on a random element, running time is independent from input order. In this way, the scenario which causes the worst case behaviour can be prevented.

Let T(n) = the random variable for the running time of randomized quicksort on an input of size n, assuming random numbers are independent. Let X denote the random variable counting the number of comparisons in all calls to Randomized-Partition. For k = 0, 1, ..., n–1, define the indicator random variable :

$$X_K \begin{cases} 1 \; if \; partition \; generates \; (n-k-1) \; split \\ 0 \qquad\qquad\qquad\qquad o/w \end{cases}$$

$E[X_K] = P_r\{X_K = 1\} = \frac{1}{n}$

First of all, when the pivot is picked, (n − 1) comparisons are performed in order to split the array. Depending on the pivot, the array might be splitted into a LESS of size 0 and a GREATER of size n−1, or into a LESS of size 1 and a GREATER of size n−2, and so on, up to a LESS of size n−1 and a GREATER of size 0. All of these are equally likely with probability 1/n each. Therefore, we can write a recurrence for the expected number of comparisons T(n) as follows:

$T(n) = (n-1) + \frac{1}{n}\sum_{k=0}^{n-1}(T(k) + T(n-k-1))$

The equation can be rewritten by groupping and getting rid of T(0)

$T(n) = (n-1) + \frac{2}{n}\sum_{k=0}^{n-1}(T(k))$ Taking expectations of both sides :

$E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$

(The k = 0, 1 terms can be absorbed in the Θ(n).) Now, we can solve this by the "guess and prove inductively" method.

**Prove:** E[T(n)] ≤ anlgn for constant a > 0. • Choose a large enough so that anlgn dominates E[T(n)] for sufficiently small n ≥ 2.

**Use Fact :** $\sum_{k=2}^{n-1} klgk \le \frac{1}{2}n^2lgn - \frac{1}{8}n^2$

$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} aklgk + \Theta(n) \le \frac{2a}{n}(\frac{1}{2}n^2lgn - \frac{1}{8}n^2) + \Theta(n)$

$= anlgn - (\frac{an}{4} - \Theta(n))$ Express as desired – residual.

$E[T(n)] \le anlgn$ If a is chosen large enough so that $\frac{an}{4}$ dominates the $\Theta(n)$

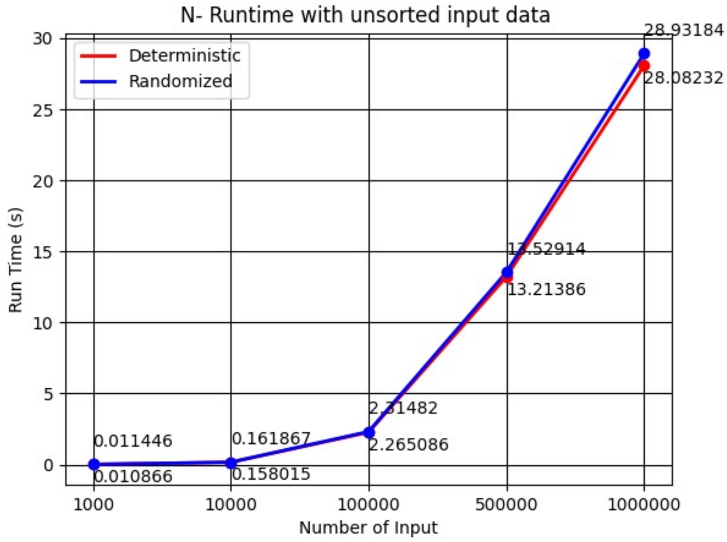
So Expected Number of Comparisons / Expected Running Time O(nlgn)

**Q-3 :**

The average running times of both algorithm on unsorted data is calculated.

| N | Pivot Selection | |
|---|---|---|
| | Deterministic | Randomized |
| 1000 | 0.010866 (s) | 0.011446 (s) |
| 10K | 0.158015 (s) | 0.161867 (s) |
| 100K | 2.265086 (s) | 2.31482 (s) |
| 500K | 13.21386 (s) | 13.52914 (s) |
| 1M | 28.08232 (s) | 28.93184 (s) |

Because the input data is not sorted, we can say that the running times of both algorithm are $O(n.\log_2 n)$. For the deterministic pivot selection, the worst case only happens the input is sorted or reversely sorted, also the randomized pivot selection algorithm's expected running time is $O(\text{nlgn})$ as found in question 1 and 2.




| N | T(n) | Value | Rate of Increase | Running Time (Deterministic) | Rate of Increase |
|---|---|---|---|---|---|
| 1000 | n*lgn | 9965.78 | | 0.010866 (s) | |
| 10K | n*lgn | 132877.12 | 13.333 | 0.158015 (s) | 14.5421 |
| 100K | n*lgn | 1660964.04 | 12.50 | 2.265086 (s) | 14.3346 |
| 500K | n*lgn | 9465784,28 | 5.69 | 13.21386 (s) | 5.8337 |
| 1M | n*lgn | 19931568,56 | 2.105 | 28.08232 (s) | 2.1252 |

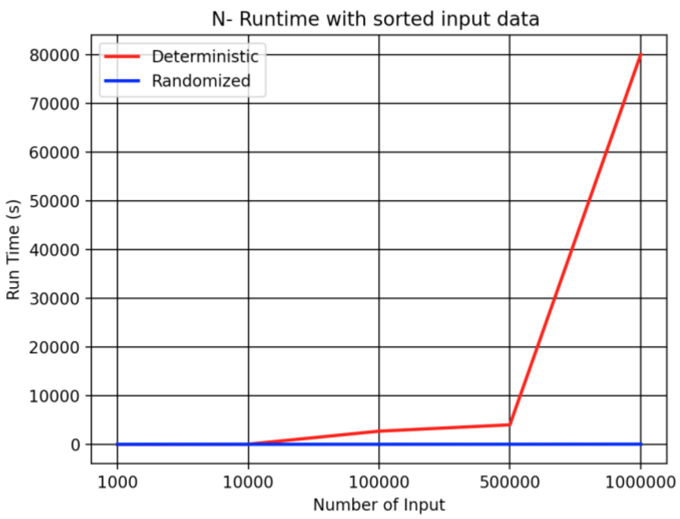
As can be seen in above table, for the same input values on the deterministic algorithm the increasing ratio of T(n)s and increasing ratio of running times is so close.

| N | T(n) | Value | Rate of Increase | Running Time (Randomized) | Rate of Increase |
|---|---|---|---|---|---|
| 1000 | n*lgn | 9965.78 | | 0.011446 (s) | |
| 10K | n*lgn | 132877.12 | 13.333 | 0.161867 (s) | 14.14179 |
| 100K | n*lgn | 1660964.04 | 12.50 | 2.31482 (s) | 14.3007 |
| 500K | n*lgn | 9465784,28 | 5.69 | 13.52914 (s) | 5.84457 |
| 1M | n*lgn | 19931568,56 | 2.105 | 28.93184 (s) | 2.13848 |

The same table is created for randomized pivot selection algorithm and the results which can be seen above table very similar to the deterministic algorithm. With the observed similarities between two algorithm and increasing rate of nlgn we can say that the both of these algorithms worked with $O(\text{nlgn})$ running times, as found in previously questions.

**Q- 4 :**

The average running time of both algorithm on sorted data is calculated. As can be seen in table with deterministic pivot selection algorithm, the average runnig times for 500K and 1M input values could not calculated. Because the data is already sorted, worst case happened for deterministic pivot selection algorithm, its running time is $O(n^2)$ and with large input values it takes very long time.




As can be stated before, in the randomized version of Quick sort we impose a distribution on input by picking the pivot element randomly. Randomized Quick Sort works well even when the array is sorted/reversely sorted and the complexity is more towards O(n log n).

Obviously, in the case the data is already sorted the randomized pivot selection algorithm should be used, because it is unlikely that this algorithm will choose a terribly unbalanced partition each time (except the case all the array elements are identical), so the performance is very good almost all the time. In the case data is unsorted, it is seen that both algorithms are work with $O(nlgn)$ running time. I would rather randomized quick sort for unsorted data. In fact both analyses are very similar, however in practice the randomized algorithm ensures that not one single input elicits worst case behavior.

| N | Pivot Selection | |
|---|---|---|
| | Deterministic | Randomized |
| 1000 | 0.2682824 (s) | 0.0108104 (s) |
| 10K | 25.58149 (s) | 0.1595882(s) |
| 100K | 2696.68 (s) | 2.145922 (s) |
| 500K | Very Large | 12.7351 (s) |
| 1M | Very Large | 28.15812 (s) |

**Q-5:**

In the dual-pivot quick sort algorithm, the partitioning happens around two pivots as left < right. Then three sub-arrays are sorted recursively. For each array element x, its class is determined

Average case analysis :

Small -> for x < left

Medium –> for left < x < right

Big -> for right < x

By comparing x to left **and/or** right.

Also, in order to arrange the array, small elements need 1 others need 2 comparisons.

**On average :**

$\frac{1}{3} * \mathbf{1} + \frac{2}{3} * \mathbf{2} = \frac{5}{3}$ comparisons are needed per element

So; any partitioning method requires $\frac{5}{3} * (\boldsymbol{n} - \mathbf{2}) \approx \frac{5\boldsymbol{n}}{3}$ comparisons

**Worst Case :** If the input array is already sorted or reversely sorted the worst case behaviour will be observed. The following recurrence is worst case partitioning :

$$\mathrm{T(n) = T(n-2) + T(0) + T(0)} + \Theta(n)$$

$$\mathrm{T(n) = T(n-2) + cn}$$

By solving this recurrence ; $\mathrm{O(n^2)}$ worst case running time is determined

**Best Case :**

If the pivot is selected median of the list then the recurrence :

$\mathrm{T(n) = 2T}\left(\frac{(\mathrm{n}-1)}{2}\right) + \Theta(n)$ by solving this best case : ; $\mathrm{O(nlgn)}$

The best and worst case running times of both algorithms are same, but according to experiments dual pivot quick sort algorithm is a bit faster than classical quick sort. It is more efficient to use partitioning of the unsorted array to 3 parts instead of the usage of the classical approach. The more the size of of the array to be sorted , the more efficiently the new algorithm works in comparison with the classic quick sort. For example, since Java 7 release back in 2011, its default sorting algorithm is Dual Pivot Quick Sort.