CS 580, Fall 2022; Instructor: Simina Brânzei.
TAs: Shoaib Khan, Anuj Singh, Lu Yan, Zheng Zhong.
Due: September 30, 11:59PM EST. Submit to Gradescope

Problem Set 2

Reading material : Slides from class. See Kleinberg-Tardos and CLRS for additional readings on quicksort and quickselect.

Collaboration policy : Acknowledge your collaborators on the homework. You may discuss proof strategies, but the solution should be written individually in your own words.

Submission format : The solutions must be typed in Latex and submitted via Gradescope.

Problem 1. *(20 points) Recall the QuickSort algorithm done in class, where the pivot is chosen uniformly at random from the elements of the vector. The goal of this question is to understand how robust this method of choosing the pivot is.*

Consider a variant of the algorithm, which works the same way, except that for any input vector $\vec{x} = (x_1, \ldots, x_n)$, element x_i is chosen to be the pivot with probability proportional to $1 - 1/(n+i)^2$. More precisely, the probability is

$$\frac{1 - 1/(n+i)^2}{\sum_{k=1}^{n} 1 - 1/(n+k)^2}.$$

Can you still show an upper bound of $O(n \log n)$ comparisons in expectation for this variant of QuickSort? Justify your answers.

Problem 2. *(20 points) Recall the problem of computing the maximum element of a vector with n elements using k rounds of interaction with the oracle answering comparison queries. Find the function $f(n, k)$ such that the number of queries necessary and sufficient to find the maximum element in k rounds by deterministic algorithms, for constant k, is $\Theta(f(n, k))$. Justify your answer. (Recall that in class, we found the answer for $k \in \{1, 2, 3\}$).*

Problem 3. *(10 pts) Implement QuickSort and MergeSort in Python using the Jupyter framework. Plot on the same graph the expected number of comparisons that each algorithm makes as a function of the number of elements n as follows. More precisely, for each $n = 1$ to 1000:*

(a) *Generate 1000 random vectors of length n, such that each element of the vector is a random uniform number from the interval $[0, 1]$. This can be accomplished with the random() command, which generates a number $x \in [0, 1]$. Afterwards, run both MergeSort and QuickSort on this input vector, recording the number of comparisons made by each algorithm. For QuickSort, use the version that selects a pivot uniformly at random.*

(b) *Record the average, taken over the instances generated in (a), for the number of comparisons generated by MergeSort (denote it by $m[n]$) and QuickSort (denote it by $q[n]$), respectively.*

Plot the vectors m and q. What do you observe? Which algorithm made fewer comparisons in your experiment? Submit the source code for your program as well.

Problem 4 (Bonus, 20 points)**.** *Consider another variant of QuickSort, which works the same way as the standard algorithm except that the pivot is chosen as follows:*

(a) *Select a tentative pivot y uniformly at random from the input vector, denoted $\vec{x} = (x_1, \ldots, x_n)$. Sample $\log n$ elements uniformly at random from $\vec{x}$; call S the set obtained. Compare y with every element from S and check if it is in the middle third of the sorted vector of elements in S. If yes, then continue with the Partition operation as usual. Else, discard y and go back to step (a).*

The goal of this procedure was to throw away bad pivots without doing the entire Partition operation, which takes $O(n)$ comparisons, by doing a faster check first. What upper bound on the number of comparisons can you show for this procedure? Can you obtain a better constant compared to the standard version of QuickSort? Justify your answer.