CS 630 – Fall 2024 – Lab 9
Nov 13, 2024

Problem 1 *Consider the* ***randomized quicksort*** *algorithm applied to an array $A[1 \dots n]$ containing n distinct elements. In this version of quicksort, the pivot is chosen uniformly at random from the subarray being sorted at each recursive call.*

Define the indicator random variable X_{ij} for each pair of indices $1 \leq i < j \leq n$ as follows:

$$X_{ij} = \begin{cases} 1, & \textit{if elements } A_i \textit{ and } A_j \textit{ are compared during the execution of the algorithm,} \\ 0, & \textit{otherwise.} \end{cases}$$

Prove that for any pair of indices $i < j$, the probability that A_i and A_j are compared during the execution of the algorithm is:

$$\mathbb{P}(X_{ij} = 1) = \frac{2}{j - i + 1}.$$

Using the result from part (a), compute the expected total number of comparisons made by the randomized quicksort algorithm.

Solution:

(a) **Proof:**

Consider the set of elements between positions i and j inclusive:

$$S = \{A_i, A_{i+1}, \dots, A_j\}.$$

In randomized quicksort, elements A_i and A_j are compared if and only if one of them is chosen as a pivot before any other element in S is chosen as a pivot.

Each element in S has an equal chance of being the first pivot selected from S. There are $|S| = j - i + 1$ elements in S.

The probability that either A_i or A_j is the first pivot chosen from S is:

$$\mathbb{P}(X_{ij} = 1) = \frac{2}{|S|} = \frac{2}{j - i + 1}.$$

Therefore, the probability that A_i and A_j are compared is $\dfrac{2}{j - i + 1}$.

(b) **Calculation of Expected Total Number of Comparisons:**

The expected total number of comparisons $\mathbb{E}[C]$ is the sum of the probabilities that each pair of elements is compared:

$$\mathbb{E}[C] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{P}(X_{ij} = 1) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}.$$

Let $k = j - i$, so k ranges from 1 to $n - i$. Rewrite the sum:

$$\mathbb{E}[C] = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}.$$

Swap the order of summation:

$$\mathbb{E}[C] = 2 \sum_{k=1}^{n-1} \frac{1}{k+1} \sum_{i=1}^{n-k} 1 = 2 \sum_{k=1}^{n-1} \frac{n-k}{k+1}.$$

Simplify the expression:

$$\mathbb{E}[C] = 2 \sum_{k=1}^{n-1} \left(\frac{n}{k+1} - 1 \right).$$

Separate the sums:

$$\mathbb{E}[C] = 2n \sum_{k=1}^{n-1} \frac{1}{k+1} - 2(n-1).$$

Recognize that $\sum_{k=1}^{n-1} \frac{1}{k+1} = H_n - 1$, where H_n is the n-th harmonic number. Therefore:

$$\mathbb{E}[C] = 2n(H_n - 1) - 2(n-1) = 2nH_n - 2n - 2n + 2 = 2nH_n - 4n + 2.$$

Simplify:

$$\mathbb{E}[C] = 2nH_n - 4n + 2.$$

Alternatively, since H_n grows approximately like $\ln n + \gamma$ (where γ is the Euler-Mascheroni constant), for large n, the expected number of comparisons is approximately $2n \ln n$.

Problem 2 *k-median*

Given n numbers, the k-median is the kth smallest amongst them. (We can assume that numbers are distinct) In this problem you will design an algorithm to find it.

Design an algorithm that takes an unordered length-n array A of numbers, and an integer k as input and returns the kth lowest value in A.

1. design a deterministic (= non-random) algorithm for this that finds the k-median in time $O(n \log n)$.

> **Solution:** The deterministic algorithm is very simple: use a sorting algorithm with runtime $O(n \log n)$ to sort A, e.g. MergeSort and BinarySearchInsertion work well for this. Then return the kth value in the sorted order.

2. design a QuickSort-style randomized algorithm for this same problem.

   As it turns out the expected runtime of this is in fact $O(n)$ (linear!) and not $O(n \log n)$ as for QuickSort. For those interested, you can find the analysis of this in the lecture slides from 11/12. (We didn't cover this, it's only for your interest.)

> **Solution:** We run a modified version of QuickSort: In each iteration pick a random pivot, compute the two subarrays of items below and above the pivot. Then recurse on the half that contains the kth lowest element.
>
> Here is how to find which subarray to use: since the QuickSort algorithm in class does the sorting in place, by the end we know that in the sorted order the k-median resides in $A[k]$. Hence, when we make our recursive call around the pivot index q, if $q > k$ then recurse on the left subarray. Otherwise recurse on the right.
>
> The description above is enough, but here is the pseudocode for your convenience:

Algorithm 1: Rnd-kMedian(A, p, r, k)

```
if p < r then
    q ← RND-PARTITION(A,p,r) /* indentical to partition function from class */
    if q > k then
        Rnd-kMedian(A,p,q-1);
    else
        Rnd-kMedian(A,q+1,r);
```