CSCI-GA.1170-001/002 Fundamental Algorithms October 14, 2015

Solutions to Problem 2 of Homework 5 (15 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, October 13*

Assume that we are given n bolts and n nuts of different sizes, where each bolt exactly matches one nut. Our goal is to find the matching nut for each bolt. The nuts and bolts are too similar to compare directly; however, we can test whether any nut is too big, too small, or the same size as any bolt.

(a) (4 points) Prove that in the worst case, $\Omega(n \log n)$ nut-bolt tests are required to correctly match up the nuts and bolts.

Solution:

We can model the algorithm for the matching nuts and bolts problem using a decision tree. The tree will be a ternary tree as every comparison can lead to three possible outcomes, less than, greater than or equal to. Therefore, if the height of the tree is h, there will be atmost 3^h number of leaf nodes. We know that the height of the tree corresponds to the worst case number of comparisons made by the algorithm, which is the lower bound of the running time.

Now consider the input of every permutation of the bolts. Now each of this permutation has to map to a distinct leaf node. Let's assume that any two different inputs from this permutation map to the same leaf node, it means that the algorithm applied to both of these inputs the same permutation with respect to the nuts and the algorithm didn't match them correctly to the corresponding permutation of nuts. Therefore, out assumption is wrong. Therefore, every permutation of bolts map to a unique leaf node. This means there are at least $n!$ leaf nodes. Let the number of leaves be l

$$l \geq n!,\ 3^h \geq l \implies 3^h \geq n! \implies h \geq \log_3 n! \implies h = \Omega(n \log n)$$

□

(b) (6 points) Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs. (Hint: prove two separate lower bounds: $\Omega(n)$ and $\Omega(k \log n)$.)

Solution:

We can model the algorithm for matching k arbitrary matching pairs using a decision tree.The tree will be a ternary tree as every comparison can lead to three possible outcomes, less than, greater than or equal to. Therefore, if the height of the tree is h, there will be atmost 3^h number of leaf nodes. We know that the height of the tree corresponds to the worst case number of comparisons made by the algorithm, which is the lower bound of the running time.

For any input of bolts we need to match k arbitrary pairs of nuts with bolts. Therefore, first select any k nuts $= \binom{n}{k}$ and these n can be permuted. Thus, total number of leaf nodes will be at least $n! \times \binom{n}{k} = \frac{n!}{(n-k)!}$. Let l be the number of leaf nodes

$$l \geq n!,\ 3^h \geq l \implies 3^h \geq \frac{n!}{(n-k)!} \implies h \geq \log_3 \frac{n!}{(n-k)!} \implies h = \Omega(n + k \log n)$$

□

(c) (5 points) Give a randomized algorithm that runs in expected time $O(n)$ and finds the k-th largest nut given any integer k. You may assume that it is possible to efficiently sample a random nut/bolt.

Solution:

Algorithm

Let N be nuts and B be bolts. The idea is similar to Quickselect. So the expected running time will be $O(n)$

- Take a random bolt from the set of bolts
- Using this bolt, partition the nuts into two parts, which is smaller than this bolt and larger than this bolt i.e $N[1 \dots p-1]$ and $N[p+1 \dots n]$
- If the matched nut with the above bolt is the k^{th} largest nut then return it
- Else, Using this nut to partition the bolts into two parts, which is smaller than this nut and larger than this nut i.e $B[1 \dots p-1]$ and $B[p+1 \dots n]$
- If $N[p]$ is smaller than the k^{th} largest nut, then recursively try to find k^{th} largest nut in the $N[p+1 \dots n]$ partition
- If $N[p]$ is larger than the k^{th} largest nut, then recursively try to find k^{th} largest nut in the $N[1 \dots p-1]$ partition

Psuedocode

In the below psuedocode, PARTITION procedure partitions the nuts and bolts into two parts, the first part contains all the nuts/bolts smaller than the pivot and the second part contains all the nuts/bolts larger than the pivot. The pivot is provided as an argument. When partitioning the nuts, a the last bolt is taken as pivot and when partitioning bolts, the last nut is taken as pivot

```
1  Algorithm: FIND-NUT(N, B, low, high, k)
2  pivot = PARTITION(N, B, low, high, B[high])
3  if pivot is k then
4  |  Return N[pivot]
5  end
6  PARTITION(B, N, low, high, N[high])
7  if pivot < k then
8  |  FIND-NUT(N, B, pivot + 1, high, k)
9  end
10 else if pivot > k then
11 |  FIND-NUT(N, B, low, pivot − 1, k)
12 end
```

Algorithm 1: Algorithm to find k^{th} largest nut in $O(n)$ time

The algorithm is similar to Quickselect, taught in class. Therefore the expected running time is $O(n)$ □