CS 577: Introduction to Algorithms Fall 2020

Homework 1

Instructor: Dieter van Melkebeek TA: Andrew Morgan

This homework covers graph primitives and the divide and conquer paradigm. **Problems 1 and 3 must be submitted for grading by 11:59pm on 9/14.** Problem 1 will be discussed in your discussion section on Friday. Please refer to the homework guidelines on Canvas for detailed instructions.

Warm-up problems

1. [Graded, 2 points] In a tree T, the distance between two vertices is the number of edges on the unique simple path between them. The *depth* of a nonempty rooted tree T is the largest distance between any vertex and the root, and the *diameter* of T is the largest distance between any pair of vertices.

   The algorithm below purports to compute the diameter of a given nonempty rooted tree.

   Argue that FindDiameterAndDepth and FindDiameter

   (a) correctly implement their specifications and

   (b) run in linear time.

Algorithm 1

Input: T, where T is a nonempty rooted tree
Output: the diameter of T

```
1: procedure FindDiameter(T)
2:     (D, d) ← FindDiameterAndDepth(T)
3:     return D
```

Input: T, where T is a nonempty rooted tree
Output: (D, d) where D is the diameter of T and d is the depth of T

```
4: procedure FindDiameterAndDepth(T)
5:     if T's root has no children then
6:         return (0, 0)
7:     else
8:         T_1, ..., T_k ← the subtrees of T rooted at the children of the root of T
9:         for i = 1 to k do
10:            (D_i, d_i) ← FindDiameterAndDepth(T_i)
11:        d ← 1+ largest d_i
12:        if k = 1 then d' ← 0 else d' ← 1+ second largest d_i including repetitions
13:        D ← max(D_1, ..., D_k, d + d')
14:        return (D, d)
```

2. Consider the problem of powering an integer:

   Input: (a, b) with $a, b \in \mathbb{Z}$ and $b \geq 1$

   Output: a^b, which we define as $a^b \doteq \underbrace{a \cdot a \cdot \cdots \cdot a}_{b \text{ times}}$, where "$\cdot$" denotes multiplication.

   Design an algorithm for this problem that uses at most $O(\log b)$ multiplications. Hint: First consider the case where b is a power of 2.

Regular problems

3. [Graded, 3 points]

   (a) You are given a perfect binary tree T with $n = 2^d$ leaves, where each leaf contains an integer value. Reading the leaf values from left to right yields a sequence of integers. The question is how small we can make the number of inversions in that sequence by applying any number of operations of the following type: Select an internal vertex and swap the two child subtrees. Data associated to a vertex in a subtree follow the vertex in the swap.

   For example, if the sequence of leaf values is $(4, 2, 1, 3)$, then a swap at the root followed by a swap at the right child of the root turns the sequence into $(1, 3, 2, 4)$, which has only one inversion. See Figure 1. It is impossible to do better, so the answer for this particular example is 1.

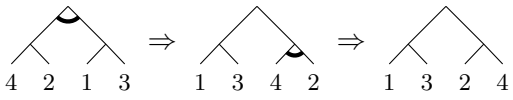


Figure 1: Example for Problem 3(a)

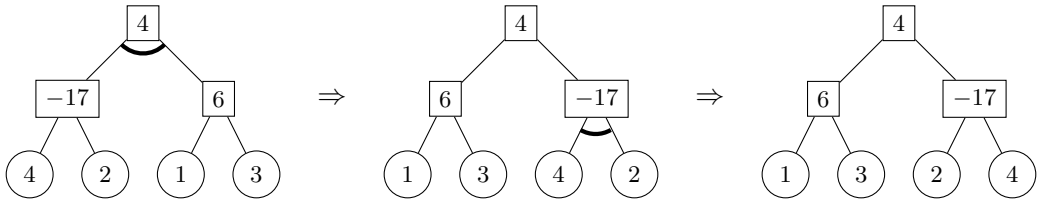
   Design an $O(n \log n)$ algorithm for this problem.

   (b) Consider the following generalization of part (a). You are given a perfect binary tree T with $n = 2^d$ leaves, where each leaf *and each internal vertex* contains an integer value (positive or negative). As before, reading the leaf values (and the leaf values only) from left to right yields a sequence of n integers. The values stored at the internal vertices factor into determining the *undesirability* of inversions in this sequence. Given an inversion (i, j), its undesirability is defined to be the value stored at the lowest-level common ancestor of the i-th and j-th leaves of T. The question is how small can we make the sum of the undesirabilities of all inversions in the sequence by applying any sequence of the same type of operations as in part (a).

   Continuing the example above, suppose the root stores an undesirability value of 4, the root's left child stores -17, and the root's right child stores 6. The same sequence of swaps as before can be depicted as shown in Figure 2. The selected sequence of swaps no longer minimizes the total undesirability. The minimum turns out to be -13, as coincidentally attained by the second tree in the figure (swapping at the root only).

   Explain how to adapt your solution to part (a) to handle part (b). Your algorithm should continue to run in $O(n \log n)$ time. Note that part (a) corresponds to the special case of part (b) in which all undesirabilities are 1.

Figure 2: Example for Problem 3(b).



In the first tree, the inversion between the first two leaves has undesirability -17 since their lowest-level common ancestor is the left-child of the root. The inversions between the first and third leaves, the first and fourth leaves, and the second and third leaves all have undesirability 4, since each of those pairs of leaves has the root as their lowest-level common ancestor. There are no other inversions. The total undesirability of the first tree is therefore $-17 + 4 + 4 + 4 = -5$. The total undesirability of the second tree is $-17 + 4 = -13$, and of the third tree is 4.

4. Consider the following computational problem:

   Input: Array $A[1, \ldots, n]$ of positive integers.

   Output: Array $C[1, \ldots, n]$ where $C[i]$ is the number of $j \in \{1, \ldots, i-1\}$ with $A[j] \geq A[i]$.

   For example, if $A = [8, 12, 10, 9, 10, 12, 7]$ then $C = [0, 0, 1, 2, 2, 1, 6]$.

   Design an $O(n \log n)$ algorithm for this problem.

Challenge problem

5. Show that every comparison-based algorithm for problem 3(a) requires $\Omega(n \log n)$ comparisons. You may first want to prove the same lower bound for problem 3(b).

Programming problem

6. SPOJ problem Insertion Sort (problem code CODESPTB).