Divide and conquer, dynamic programming and greedy: More practice problems

Here's a bunch more problems in case you are looking for more practice, or in case you find these problems fun. These are totally optional and not required in view of the exam (the examples we worked on in class, labs and assignments are sufficient).

1. **Playing a game:** Let's play a game where the input is a sequence $A[1], .., A[n]$ of numbers. At each turn, you can either (1) take the first number in the sequence, add it to your score, and then delete that element from the sequence; or (2) you can delete the first number in the sequence, take the second number in the sequence and add it to your score, and then finally delete the second number from the sequence. You keep playing until all of elements in the sequence have been deleted.

   Find a choice of moves that maximize your score.

   For instance, if the sequence is $5, -2, 4, -4, -1, 5$, your best choice of moves is "take first, take second, take second, take first", for a total score of $5 + 4 + -1 + 5 = 13$.

   Hint: Let $f(i)$ denote the best score attainable starting from the sequence $A[i], .., A[n]$. Write a recursive formulation for $f(i)$.

2. **Finding the singleton:** You are given a sorted array of numbers where every value except one appears exactly twice, and one value appears only once. Design an efficient algorithm for finding which value appears only once. Here are some example inputs to the problem:

   $$1, 1, 2, 2, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8$$
   $$10, 10, 17, 17, 18, 18, 19, 19, 21, 21, 23$$
   $$1, 3, 3, 5, 5, 7, 7, 8, 8, 9, 9, 10, 10$$

   Note: A general solution should not assume anything about the numbers in the array; specifically, they may not be in a small range, and may not be consecutive. Hint: Look at the indices in the array of the pairs that are equal, before the singleton, and after the singleton, and think along the lines of binary search, aiming for $O(\lg n)$.

3. **The skis and skiers problem**[1]**:** You've decided to become a ski bum, and hooked up with Sugarloaf Ski Resort. They'll let you ski for free all winter, in exchange for helping their ski rental shop with an algorithm to assign skis to skiers. Ideally, each skier should obtain a pair of skis whose height matches his or her own height exactly. Unfortunately, this is generally not possible. We define the *disparity* between a skier and his/her skis as the absolute value

[1]Adapted from Harvey Mudd College.

of the difference between the height of the skier and the height of the skis. The objective is to find an assignment of skis to skiers that minimizes the sum of the disparities.

(a) First, let's assume that there are n skiers and n skis (i.e. same number of skis and skiers). Consider the following algorithm: consider all possible assignments of skis to skiers; for each one, compute the sum of the disparities; select the one that minimizes the total disparity. How much time would this algorithm take on a 1GHz computer, if there were 50 skiers and 50 skis?

(b) One day, while waiting for the lift, you make an interesting discovery: if we have a short person and a tall person, it would never be better to give to the shorter person a taller pair of skis than were given to the taller person. Follow the proof below and fill in the missing parts:

Proof: Let P1, P2 be the length of two skiers, and S1, S2 the lengths of the skis. We assume $P1 < P2$ and $S1 < S2$. We'll prove that pairing P1 with S1 and P2 with S2 is better than pairing P1 with S2 and P2 with S1. Basically we'd like to show that $|P1 - S1| + |P2 - S2| \leq |P1 - S2| + |P2 - S1|$. To do so we'll consider all possible cases:

- $P1 < S1 < P2 < S2$:
  Draw this case and show graphically that $(S1 - P1) + (S2 - P2) \leq (S2 - P1) + (P2 - S1)$

  All the other cases can be shown in a similar way (you don't need to do it).
- $P1 < S1 < S2 < P2$:
- $P1 < P2 < S1 < S2$:
- $S1 < P1 < P2 < S2$:
- $S1 < P1 < S2 < P2$:
- $S1 < S2 < P1 < P2$:

(c) Describe a greedy algorithm to assign the skis to skiers (when there are as many pairs of skis as skiers), and argue why this algorithm is correct. What is the time complexity of your algorithm?

(d) Consider the general case where there are m skiers and n pairs of skis (and $n \geq m$). The goal is assign skis to skiers in order to minimize the sum of the disparities.

Hint: Sort the skiers and skis by increasing height. Let h_i denote the height of the ith skier in sorted order, and s_j denote the height of the jth pair of skis in sorted order. Let OptSkis(i, j) be the optimal cost (disparity) for matching the first i skiers with skis from the set $\{1, 2, ..., j\}$. The solution we seek is OptSkis(m, n). Define OptSkis(i, j) recursively. What is the running time of OptSkis(m, n)?

(e) Memo-ize the recursive formulation above. What does the running time of OptSkis(m, n) become?

(f) Briefly describe how your algorithm can be modified to allow you to find an actual optimal assignment (rather than just the cost) of skis to skiers. How long will this take?

(g) Illustrate your algorithm by explicitly filling out the table OptSkis(i, j) for the following sample data: ski heights 1, 2, 5, 7, 13, 21. Skier heights 3,4, 7, 11, 18.

Note: This requires a LOT of patience, but it really helps you understand how recursion works. I think we shoud all write the code for this.

4. **Longest increasing subsequence:**[2] A *subsequence* of a sequence (for example, an array, linked list or string), is obtained by removing zero or more elements and keeping the rest in the same order. A subsequence is called a *substring* if its elements are contiguous in the original sequence. For example:

   (a) GORIT, ALGO, RITHMS are all substrings of ALGORITHMS

   (b) GRM, LORI, LOT, AGIS,GORIMS are all subsequences of ALGORITHMS

   (c) ALGOMI, OG are *not* subsequences of ALGORITHMS

   The problem: Given an array $A[1..n]$ of integers, compute the length of a *longest increasing subsequence* (A sequence $B[1..k]$ is *increasing* if $B[i] < B[i+1]$ for all $i = 1..k-1$). For example, given the array

   $$\{5, 3, 6, 2, 1, 5, 3, 1, 2, 5, 1, 7, 2, 8\}$$

   your algorithm should return 5 (for e.g. correponding to the subsequence $\{1, 3, 5, 7, 8\}$; there are other subsequences of length 5). Describe an algorithm which, given an array $A[]$ of n integers, computes the LIS of A.

```
EXAMPLES:

Input: arr[] = {3, 10, 2, 1, 20}
Output: Length of LIS = 3
The longest increasing subsequence is 3, 10, 20

Input: arr[] = {3, 2}
Output: Length of LIS = 1
The longest increasing subsequences are {3} and {2}

Input: arr[] = {50, 3, 10, 7, 40, 80}
Output: Length of LIS = 4
The longest increasing subsequence is {3, 7, 40, 80}

Input: A[]={0,8,4,12,2,10,6,14,1,9,5, 13,3,11,7,15}
Output: Length of LIS = 6
Explanation:Longest increasing subsequence 0 2 6 9 13 15, which has length 6

Input: A[] = {5,8,3,7,9,1}
Output: Length of LIS = 3
Explanation:Longest increasing subsequence 5 7 9, with length 3
```

[2]Leetcode #300

Hint: Use the following subproblem: Let $L(i)$ be the length of a LIS starting with $A[i]$. Express $L(i)$ rercursively. Assume you computed $L(i)$ for all $i = 1, 2, ..., n$. How do you find the LIS of A?

5. **Party problem:** Suppose you are in charge of planning a party for Bowdoin College. The college has a hierarchical structure, which forms a tree rooted at President Zaki. On the very last level are the faculty, grouped by department. Each faculty has "underneath" all students taking a class with her (that semester). Assume that every person is listed at the highest possible position in the tree and there are no double affiliations (everybody has one and only one supervisor in this hierarchy and no student is in more than one class).

   You have access to a secret database which ranks each faculty/staff/student with a conviviality rating (a real number, which can be negative if the person is unkind and unpleasant). In order to make the party fun for everybody, President Zaki does not want both a faculty/staff/student and his or her immediate "supervisor" to attend.

   You are given a tree that describes the strucure of Bowdoin College. Each node has a (down) pointer to its left-most child, and a (right) pointer to its next sibling (if unclear read Section 10.4 in CLR). Each node also holds a name and a conviviality ranking. Describe an algorithm to make up a guest list that maximizes the sum of the conviviality rankings of the guests. Analyze the running time of your algorithm.

   Note: A different way to word this is that you are given a tree, each node has a value, and you want to find a subset of nodes in the tree so that their total value is maximized, with the following constraint: if a node is included, none of its immediate children are included.