ASSIGNMENT 2

DUE: Thursday June 9, 11:59 PM. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.

Please read http://www.student.cs.uwaterloo.ca/~cs341 for general instructions and policies.

It is possible to write a three page solution worth full marks. Longer solutions are fine. :)

1. [5 marks] Greedy

Suppose there are $n$ valuable gemstones embedded in the ground, arranged in a line. The gemstones have positive integer values $v_1, v_2, ..., v_n$. You can extract gemstones from the ground using your pickaxe (and subsequently sell them), but in the process of extracting gemstone $v_i$, you will destroy gemstones $v_{i-1}$ and $v_{i+1}$. The exceptions are $v_1$ and $v_n$. Extracting $v_1$ will only destroy one gemstone, $v_2$, and extracting $v_n$ will only destroy $v_{n-1}$. The goal is to extract gemstones in such a way as to maximize the total value you obtain.

![Diagram showing gemstones with some destroyed and some extracted]

Prove that the following greedy algorithms for this problem are incorrect.

(a) [1 mark] Consider gems in the input order, and take every gemstone that you can. (In other words, at each step, if the gemstone being considered is not already destroyed, take it, destroying its neighbours.)

(b) [1 mark] Sort from most valuable to least valuable (i.e., such that $v_1 \geq v_2 \geq ... \geq v_n$) then take every gemstone you can in that order. (Assume we also keep track of where the gems were in the input order, i.e., before the sort, so we know which gems are destroyed when we extract a gem.)

(c) [1 mark] Do preprocessing to determine the sums $S_{odd}$ of all odd $v_i$, i.e., $v_1, v_3, ...$, and $S_{even}$ of all even $v_i$, i.e., $v_2, v_4, ...$. If $S_{odd}$ is larger, extract all of the odd gems. Otherwise extract the even ones.

(d) [1 mark] For each gem $v_i$, $1 < i < n$, let $f_i = \frac{v_i}{v_{i-1}+v_{i+1}}$, and let $f_1 = \frac{v_1}{v_2}$ and $f_n = \frac{v_n}{v_{n-1}}$. Sort from largest to smallest $f_i$ (breaking ties arbitrarily), remembering where each gem was in the input order. Then, take every gem you can (i.e., that is not destroyed) in that order.

(e) [1 mark] For each gemstone $v_i$, $1 < i < n$, let $f_i = v_i - v_{i-1} - v_{i+1}$ (i.e., the value you would get from the gem, minus the values of the gems that would be destroyed). For $v_1$ and $v_n$, let $f_1 = v_1 - v_2$ and $f_n = v_n - v_{n-1}$, respectively. Sort from largest to smallest $f_i$ (breaking ties arbitrarily), then take every gemstone you can (i.e., that is not destroyed by another gem you extract) in that order.

1
2. [7 marks] **Divide & Conquer**

Consider the problem in Q1. Solve the problem using divide & conquer. In other words, design a divide & conquer algorithm that returns the maximum value that can be achieved by extracting gems. Briefly justify correctness. Analyze your algorithm’s runtime using $\Theta$ notation. You can ignore errors of +/- 1 or 2 in your problems sizes.
3. [5 marks] **Greedy**

Recall the non-dominated points problem described in the lecture 5 slides.

**Prove or disprove**: the following greedy algorithm correctly solves the non-dominated points problem.

Hint: one reasonable way to prove this is with a loop invariant.

```python
GreedyNDPoints(S[1..n])
    sort points in S so that S[1].x > S[2].x > ... > S[n].x
    output = new list

    ymax = S[1].y
    output.append(S[1])

    for i = 2..n
        if S[i].y >= ymax
            ymax = S[i].y
            output.append(S[i])

    return output
```
4. [9 marks] **Divide & Conquer**

Suppose we are given \( n \) input points with positive integer coordinates, and we want to find the **optimal triangle** that can be formed by choosing 3 of these input points.

The optimal triangle is the one with the **smallest perimeter**.

Give an efficient divide & conquer algorithm to solve this problem, briefly justify its correctness (can be done as part of the algorithm explanation), and analyze its complexity using either \( \Theta \) or big-\( O \) notation (doesn’t matter which).

In your recurrence relation you may ignore floors/ceilings/off-by-1-or-2.

You may assume you are given a constant time function \( \text{perimeter}(a, b, c) \) that computes the perimeter of a triangle, given three points \( a, b, c \).

Hint: try to modify the algorithm for closest pair.

Hint for the analysis: Suppose a triangle cannot have perimeter smaller than some \( \delta \). Then, how many **triangles** can be in a square of side length \( \delta/4 \)?