Question 1. Divide-and-Conquer [20 marks]

Write a divide-and-conquer algorithm that finds the maximum difference between any two elements of a given array of n numbers (not necessarily distinct) *in $O(n)$ time*. For example, on input $A = [4.5, 10, -2, \pi, -7.115]$, your algorithm should return 17.115.

Justify briefly that your algorithm is correct and runs within the required time bound. (For your reference, the Master Theorem states that a recurrence of the form $T(n) = aT(n/b) + \Theta(n^d)$ has solution $\Theta(n^d)$ if $a < b^d$, $\Theta(n^d \log n)$ if $a = b^d$, and $\Theta(n^{\log_b a})$ if $a > b^d$.)

Note: For full marks, your answer *must* make use of the divide-and-conquer method. Partial marks will be given for an $O(n \log n)$ divide-and-conquer method.

Sample Solution:

The maximum difference is simply the difference between the maximum and the minimum elements. We find the pair (minimum element, maximum element) recursively (using divide-and-conquer), then return their difference.

```
MaxDiff($A$):
    $min, max \leftarrow MinMax(A)$
    **return** $max - min$

MinMax($A$):
    **if** $n = 1$:        // $n$ = size of $A$
        **return** $A[1], A[1]$
    **else**:
        $m \leftarrow \lfloor n/2 \rfloor$
        $n_1, m_1 \leftarrow MinMax(A[1 \ldots m])$
        $n_2, m_2 \leftarrow MinMax(A[(m+1) \ldots n])$
        **return** $\min(n_1, n_2), \max(m_1, m_2)$
```

Procedure MinMax runs in time $\Theta(n)$ because its worst-case running time satisfies the recurrence $T(n) = 2T(n/2) + \Theta(1)$, so MaxDiff also runs in time $\Theta(n)$.

Question 2. Greedy [20 marks]

Consider the following *Art Gallery Guarding* problem. We are given a set $\{p_1, p_2, \ldots, p_n\}$ of positive numbers that specify the positions of paintings in a long hallway in an art gallery. We want to position guards in the hallway to protect every painting, using as few guards as possible. Suppose that a guard at location x can protect all paintings within 1 distance (i.e. in interval $[x-1, x+1]$), and any painting protected by at least one guard is considered protected.

We propose to solve this problem using the following greedy algorithm.

```
Sort the positions so $p_1 \leqslant p_2 \leqslant \cdots \leqslant p_n$.
$G \leftarrow \varnothing$    # current set of guards' locations
$g \leftarrow -\infty$    # position of rightmost guard in $G$ = maximum number in $G$
for $i \leftarrow n, \ldots, 1$:    # start at the rightmost painting and move to the left
    if $p_i - g > 1$:    # $p_i$ is unprotected by the guards currently in $G$
        # Place a guard 1 unit to the right of $p_i$.
        $g \leftarrow p_i + 1$
        $G = G \cup \{g\}$
return $G$
```

(a) (2.5 marks) The above algorithm does not quite work. Provide a counterexample of painting positions where the above algorithm would use more guards than required. No justification is needed.

(b) (2.5 marks) The above algorithm can be fixed by changing a single line. What is the change needed to make the algorithm always find a solution with the minimum number of guards? For partial marks, you can propose any other greedy algorithm which always finds a solution with the minimum number of guards.

(c) (2.5 mark) What is the time complexity of your algorithm from part (b)?

(d) (10 marks) Prove that your fix to the algorithm (or any other greedy algorithm you proposed) in (b) is correct.

(e) (2.5 marks) The exhibition is then moved to another gallery where the art is put up on easels in a wide open space, i.e., each painting is placed at a 2D point $p_i = (x_i, y_i)$. Suppose we use a variant of your fixed greedy algorithm from part (b), where you sort the paintings by x-coordinate, check if a painting p_i is greater than a unit Euclidean distance from the current guard (i.e., $\|p_i - g\| > 1$), and if so, place a new guard at $g \leftarrow p_i + (1, 0)$. Either prove that this algorithm will always return a solution with the minimum number of guards or provide a counterexample.

Note: The Euclidean distance is given by $\|(x,y) - (x', y')\| = \sqrt{(x-x')^2 + (y-y')^2}$.

SAMPLE SOLUTION:

Let $G_0, G_1, \ldots, G_n$ be the partial solutions generated by the algorithm. Say G_i is "promising" iff there is some solution G_i^* such that:

- $G_i \subseteq G_i^*$ (G_i^* contains every guard position in G_i);
- $\forall g \in G_i^* - G_i, g > g_i = \max\{g' \in G_i\}$ (every additional guard in G_i^* is positioned further right than the last guard in G_i)—we say G_i^* "extenda" G_i if it satisfies the first two properties;
- $|G_i^*|$ is minimum (G_i^* is optimum).

CLAIM: $\forall i \geqslant 0, G_i$ is promising.

PROOF: By induction on i.

Base Case: $G_0 = \varnothing$. Let G^* be any optimum solution. Then $G_0 \subseteq G^*$ and $\forall g \in G^*, g > g_0 = -\infty$.

Ind. Hyp.: Suppose $i \geqslant 0$ and G_i^* extends G_i.

Ind. Step: Either $G_{i+1} = G_i$ or $G_{i+1} = G_i \cup \{p_{i+1} + 1\}$.

Case 1: If $G_{i+1} = G_i$, then $G_{i+1} = G_i \subseteq G_i^*$ and $\forall g \in G_i^* - G_{i+1}, g \in G_i^* - G_i \Rightarrow g > g_i = g_{i+1}$. So G_i^* already extends G_{i+1}.

Case 2: If $G_{i+1} = G_i \cup \{p_{i+1} + 1\}$, then either $p_{i+1} + 1 \in G_i^*$ or $p_{i+1} + 1 \notin G_i^*$.

Subcase A: If $p_{i+1} + 1 \in G_i^*$ then G_i^* already extends G_{i+1}: $G_{i+1} = G_i \cup \{p_{i+1} + 1\} \subseteq G_i^*$ and $\forall g \in G_i^* - G_{i+1}, g > g_i \wedge g \neq p_{i+1} + 1 \Rightarrow g > g_{i+1} = p_{i+1} + 1$.

Subcase B: If $p_{i+1} + 1 \notin G_i^*$ then $p_{i+1} > g_i + 1$ (from the algorithm) so p_{i+1} is unprotected by any guard in G_i. This means there is some $g \in G_i^* - G_i$ with $g > g_i$ (by the I.H.) and $g \leqslant p_{i+1} + 1$ (else p_{i+1} would be unprotected by G_i^*). Then, every painting protected only by g is also protected by $p_{i+1} + 1$. So $G_{i+1}^* = G_i^* - \{g\} \cup \{pi + 1 + 1\}$ is an optimum solution that extends G_{i+1}.

Hence, G_i is promising for $i = 0, 1, \ldots, n$. In particular, G_n is promising: $G_n \subseteq G_n^*$ and $\forall g \in G_n^* - G_n, g > g_n$, for some optimum G_n^*. But this means $G_n = G_n^*$, as desired.

Question 3. Dynamic Programming [20 marks]

A new country Oddistan has a currency *oddollar* with coins of denomination 3, 5 and 7.

(a) (2.5 marks) Can you always purchase an item worth n oddollars, where $n >= 5$, with exact change in coins. If so, explain why; otherwise, provide the smallest counterexample $n \geqslant 5$ (no justification is needed).

(b) (2.5 marks) For all n that you can pay for exactly, is the combination of coins unique? If so, explain why; otherwise, provide a example n and two ways to make up n using the coin denominations.

(c) (15 marks) A new government wants to the change the coin denominations to 3, 11 and 13. Write a dynamic programming algorithm that returns (true/false) whether an exact change can be made for n oddollars, given any integer $n > 0$. Clearly define the quantity that your dynamic programming algorithm computes, write a Bellman equation, briefly argue that your Bellman equation is correct, identify the initial conditions, and analyze the time and space complexity of your algorithm.

Sample Solution:

Recursive Structure: If $N = 3x + 11y + 13z$ for some $x, y, z \in \mathbb{N}$, then either $x > 0$ and $N = 3 + 3(x-1) + 11y + 13z$, or $y > 0$ and $N = 11 + 3x + 11(y-1) + 13z$, or $z > 0$ and $N = 13 + 3x + 11y + 13(z-1)$.

Array: For $i = -24, -23, \ldots, -1, 0, 1, \ldots, N$, $A[i] =$ True if i nuggets can be purchased exactly (False otherwise).

Recurrence: $A[-24] = \cdots = A[-1] = \text{False}$
$A[0] = \text{True}$ $(0 = 3 \cdot 0 + 11 \cdot 0 + 13 \cdot 0)$
$A[i] = A[i-3] \vee A[i-11] \vee A[i-13]$, for $i = 1, \ldots, N$ (from argument above)

Algorithm:

```
for $i = -24, \ldots, -1$:
    $A[i] \leftarrow \text{False}$
$A[0] \leftarrow \text{True}$
for $i = 1, \ldots, N$:
    $A[i] = A[i-3] \vee A[i-11] \vee A[i-13]$
return $A[N]$
```

Runtime is $\Theta(N)$.