1. **(15 points)** Prove that

\[ 1 + \log_2 n = O(\log_2 n) \]

**Proof:**

By the definition of Big-O, we need to find constant \( c \) and \( n_0 \) such that when \( n \geq n_0 \),

\[ 1 + \log_2 n \leq c \log_2 n \]

That is, to satisfy

\[ 1 \leq c \log_2 n - \log_2 n = (c - 1) \log_2 n \]

which can be satisfied if we choose \( c = 2 \) and \( n \geq n_0 = 2 \).

2. **(10 points)** An upper bound of an algorithm is the time function within which all instances of size \( n \) can be solved. What is a lower bound of an algorithm?

**Answer:**

A lower bound of an algorithm is a time function below which some instances of size \( n \) cannot be solved.
3. (10 points) Consider the following recursive algorithm to find both the maximum and the second maximum from a list of \( n \) elements. The algorithm first splits the list into two sublists, recursively finds the maximum and second maximum from each sublist, and combine the results from the two sublists to identify the maximum and second maximum for the original list.

Write a recurrence for the time complexity upper bound \( T(n) \) of this algorithm. What is the Big-O function for \( T(n) \)? (No proof is required.)

**Answer:**

Assume the algorithm splits the list of size \( n \) into two sublists \( A \) and \( B \) of sizes \( r \) and \( n - r \), respectively. Note that \( r \) is not necessarily \( \frac{n}{2} \). After recursive calls to compute the maximum and second maximum for sublist \( A \) and the maximum and the second maximum for sublist \( B \), respectively, the algorithm compares these 4 values and returns the top two values as the maximum and second maximum for the original list. So

\[
T(n) = T(r) + T(n - r) + c
\]

and \( T(2) = d \). Using substitution method, we can prove \( T(n) = O(n) \).

4 (10 points) The notion of divide-and-conquer in general refers the approach that breaks the input instance into two or more pieces and solves each piece recursively. **Briefly explain** why it is beneficial to have additional steps to manipulate data such that the “total size” of the broken pieces is a fraction smaller than the size of the original instance.

**Answer:**

Discarding a fraction (size) of data makes it possible to reduce time complexity to:

\[
T(n) = T(\alpha_1 n) + \cdots + T(\alpha_k n) + g(n)
\]

where \( \alpha_1 + \cdots + \alpha_k < 1 \).

For example, when \( g(n) = cn \), a linear function, we can obtain \( T(n) = O(n) \), instead of \( T(n) = O(n \log_2 n) \) that is the result when \( \alpha_1 + \cdots + \alpha_k = 1 \).

5. (10 points) **RANDOMIZED QUICKSORT** has the average time complexity \( O(n \log_2 n) \). Which of the following statements is (are) true?

Two different runs of the algorithm (on the same input)

(1) may generate two different results; [False]
(2) may use two different set of pivots for the partitions; [True]
(3) may conduct two different sets of comparisons; [True]
(4) may conduct two different numbers of comparisons. [True]
6. **(15 point)** It is known that function `MERGE` used by `MERGE_SORT` merges two sorted sublists into a sorted list. The merge process takes time $O(n)$ on two sublists of size $\frac{n}{2}$. Use the decision tree method to outline a proof that the problem to merge two sorted sublists of size $\frac{n}{2}$ has the lower bound $\Omega(n)$.

**Proof:**

This question asks you to use decision tree to prove lower bound $\Omega(n)$ for the problem merging two sorted sublists of length $\frac{n}{2}$ into a sorted single list of length $n$.

The elements on the sorted list come from the two sorted sublists. Any element on the sorted list may come from either the first or second sublist, one of 2 choices. And there are $n$ elements. So the total number of combinations is $2^n$.

Any decision tree would have at least $2^n$ leaves and thus of height at least $\log_2(2^n) = n$. So the lower bound for this problem is $\Omega(n)$.

7. **(15 points)** Consider the following recurrence that formulates the objective function for the 0-1 Knapsack problem whose input consists of $n$ items with size $s_i$ and value $v_i$, $i = 1, 2, \ldots, n$, and bag size $B$.

$$V(k, X) = \max \left \{ \begin{array}{ll} V(k - 1, X - s_k) + v_k & s_k \leq X \\ V(k - 1, X) & \end{array} \right.$$  

where $V(k, X)$ is the maximum value of packing some of the first $k$ items into a space of size $X$, with base cases $V(0, X) = 0$ and $V(k, 0) = 0$.

Fill out the DP table for the following small input (and indicate how each cell is calculated, e.g., with arrows)

$$(s_1 = 2, v_1 = 4), (s_2 = 1, v_2 = 3), \text{ and } B = 3$$
8. **(15 points)** Consider the coin change problem with only two kinds of coins to use: 1-cent and 3-cents. The problem is to **find the minimum number** of such coins for any given input value of cents (e.g., 16 cents). Assume we are to use dynamic programming on this problem. Please design an objective function for the problem and formulate a recurrence for the designed function. Make sure you also include base case(s) for the function.

**Answer:**

Define $Coins(m)$ to be the minimum number of coins (of 3-cents and 1-cents) to change $m$ cents.

Then, recurrence for $Coins(m)$ is

$$Coins(m) = \min \left\{ \begin{array}{ll}
    Coins(m - 3) + 1 & m \geq 3 \\
    Coins(m - 1) + 1 & m \geq 1
\end{array} \right.$$

and $Coins(0) = 0$.

9. **(15 points)** The coin change problem (in Question 8) has a greedy-choice property.

(1) State the greedy-choice property (that you identify).

(2) Prove the greedy-choice property.

**Answer:**

(1) The greedy-choice property is: For any money of three cents or higher, a **3-cents** coin is contained in any minimum set coins.

(2) Proof. Let $A$ be the minimum set of coins for money of three cents or higher.

If $A$ contains a **3-cents** coin, the property is proved.

If $A$ does not contain a **3-cents** coin. Then there must be at least three **1-cent** coins. Let

$$B = A - \{1\text{-cent, 1-cent, 1-cent}\} \cup \{3\text{-cents}\}$$

Then $B$ is also a change for the same amount of money. But $|B| \leq |A|$ so $B$ is also an optimal solution. $B$ contains a **3-cents** coin.