1. (15 points) In this question you are going to prove the so called Master Theorem for divide and conquer algorithms. The proof is stated in several books and webpages, but you are required (and encouraged) to prove this by yourself without looking it up. Let $a, b > 0$ be integers and $c, d > 0$ be real numbers. Let $\alpha = \log_b a$. Suppose that $T(1) = c$, and for every $n$ divisible by $b$, we have the recursion $T(n) = aT(n/b) + cn^d$. By using induction prove that for $n = b^m$ where $m \in \mathbb{N}$, we have

- If $d < \alpha$, then $T(n) = \Theta(n^\alpha)$.
- If $d = \alpha$, then $T(n) = \Theta(n^\alpha \log(n))$.
- If $d > \alpha$, then $T(n) = \Theta(n^d)$.

**Proof:** We prove by induction on $m$ that for $m > 0$, $T(b^m) = c \sum_{i=0}^{m} a^i b^{(m-i)d}$.

- **Base case:** $m = 1, n = b$. $T(b) = aT(b/b) + cb^d = cab^d + ca^0b^d$.
- **Induction Hypothesis:** For $m = k - 1, n = b^{k-1}$, we have $T(b^{k-1}) = c \sum_{i=0}^{m} a^i b^{(m-i)d}$.
- **Induction Step:** Let $m = k, n = b^k$. We have $T(n) = T(b^k) = aT(b^{k-1}) + ca^0b^d$. Now applying induction hypothesis, we get

$$T(b^k) = a \times c \left( \sum_{i=0}^{k-1} a^i b^{(k-1-i)d} \right) + ca^0b^d = ac \sum_{i=0}^{k-1} a^i b^{(k-1-i)d} + ca^0b^d = c \left( a^0b^d + \sum_{i=0}^{k-1} a^{i+1} b^{(k-1-i)d} \right),$$

which is equal to $c \sum_{i=0}^{k} a^i b^{(k-i)d}$ as desired.

**Case 1:** $d < \alpha$. In this case we have

$$c \sum_{i=0}^{m} a^i b^{(m-i)d} = ca^m \left( \sum_{i=0}^{m} (b^d/a)^{m-i} \right) = ca^m \left( \sum_{i=0}^{m} (b^d/a)^i \right) = \Theta(a^m)$$

as $b^d/a < 1$.

**Case 2:** $d = \alpha$. In this case, $b^d = a$, and thus $c \sum_{i=0}^{m} a^i b^{(m-i)d} = c(m+1)a^m = \Theta(n^\alpha \log n)$.

**Case 3:** $d > \alpha$. In this case we have

$$c \sum_{i=0}^{m} a^i b^{(m-i)d} = cb^{dm} \left( \sum_{i=0}^{m} (a/b^d)^i \right) = \Theta(b^{dm})$$

as $a/b^d < 1$. 

1
2. We are given a sequence of positive integers $a_1, \ldots, a_n$ corresponding to the prices of a stock in times $1, \ldots, n$. We want to decide when to buy the stock and when to sell it. More precisely we want to find $i < j$ such that $a_j - a_i$ is maximized. For example if the sequence is $(30, 10, 5, 9, 6, 11, 10)$, then the optimal strategy is to buy at time 3 (at the price of 5) and sell at time 6 (at the price of 11).

(a) (3 points) Describe a simple $O(n^2)$ solution to this problem.

(b) (12 points) Improve the running time by using a divide-and-conquer recursive algorithm and analyze its correctness and running time.

(a) Solution:
For every stock price, check every price that follows it. Return the pair of indices that gives the maximum difference between prices.

(b) Solution:
Divide the sequence into sub-lists. For each sub-list we track the optimal strategy $\max_{i,j,i<j} a_j - a_i$, as well as $\max_k a_k$ and $\min_l a_l$, along with the corresponding indices $i, j, k, l$. When the list is only one or two elements, the values can be initialized trivially, along with some indicator of no optimal strategy in the case of a single element.

When merging the left list $b$ and the right list $c$ into list $a$ we compute the optimal strategy of $a$

$$\max_{i,j,i<j} a_j - a_i = \max(\max_{i,j,i<j} b_j - b_i, \max_{i,j,i<j} c_j - c_i, \max_k c_k - \min_k b_k)$$

along with $\max_k a_k = \max(\max_k b_k, \max_k c_k)$, $\min_l a_l = \min(\min_l b_l, \min_l c_l)$ and all corresponding indices. Return the optimal strategy of the full list.

Running time:
As dividing the list into sub-lists takes $O(1)$ and merging the sub-lists takes $O(1)$ then by Master’s theorem we have a running time of $T(n) = 2 \cdot T(\frac{n}{2}) + O(1) \Rightarrow O(n)$.

Correctness:
Proof by induction.
Base case: When the list contains only a single element $a_0$, the both the max and min are $a_0$ and there is no optimal solution. When the list contains two elements the optimal solution must be to buy for $a_0$ and to sell for $a_1$.

When merging lists, the optimal strategy will involve buying the stock in either the first or second half, and selling the stock in the first or second half. This corresponds to three possibilities:

(a) The stock is bought and sold in the first half: $\max_{i,j,i<j} b_j - b_i$
(b) The stock is bought and sold in the second half: $\max_{i,j,i<j} c_j - c_i$
(c) The stock is bought in the first half and sold in the second half: $\max_k c_k - \min_k b_k$

By the inductive hypothesis we can assume we have the solutions to case (a) and (b), and we can easily compute the solution to case (c). Taking the max over all three returns the optimal strategy.

3. (10 points) Use the Huffman code to convert the following sentence$^1$ into a sequence of bits: “that rug really tied the room together.”

Solution: We have a total of 39 characters. The frequency of individual characters are given below:

\[ \text{do not forget to consider the space and the period as characters too} \]

$^1$do not forget to consider the space and the period as characters too
Starting with the least frequency characters we build the Huffman tree as shown below:

![Huffman Tree](image)

**Figure 1: Huffman Tree**

Based on the tree, the encoding of different characters is as follows:

<table>
<thead>
<tr>
<th>Character</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>000</td>
</tr>
<tr>
<td>space</td>
<td>001</td>
</tr>
<tr>
<td>e</td>
<td>010</td>
</tr>
<tr>
<td>r</td>
<td>100</td>
</tr>
<tr>
<td>h</td>
<td>0110</td>
</tr>
<tr>
<td>o</td>
<td>0111</td>
</tr>
<tr>
<td>a</td>
<td>1010</td>
</tr>
<tr>
<td>g</td>
<td>1011</td>
</tr>
<tr>
<td>l</td>
<td>1100</td>
</tr>
<tr>
<td>u</td>
<td>11010</td>
</tr>
<tr>
<td>y</td>
<td>11011</td>
</tr>
<tr>
<td>i</td>
<td>11100</td>
</tr>
<tr>
<td>d</td>
<td>11101</td>
</tr>
<tr>
<td>m</td>
<td>11110</td>
</tr>
<tr>
<td>period</td>
<td>11111</td>
</tr>
</tbody>
</table>

Finally, the sentence can be encoded into a sequence of bits by replacing each character with it’s encoding from the above table.

4. (20 points) Show that no compression scheme can be expected to compress all of the \( n \)-character files. Here a character is any of the 256, 8-bit characters. Here a compression scheme is an
injective map \( C : \Omega^n \to \{0,1\}^* \), where \( \Omega \) is the set of 256 characters, and \( \{0,1\}^* \) denotes the set of all finite strings of 0’s and 1’s. In other words, you need to show that for every such \( C \), there exists some \( x \in \Omega^n \) such that \( |C(x)| \geq 8n \), as \( 8n \) is the initial size of the file.

Solution:

An injective map means the number of possible inputs to the mapping will be at least as great as the number of possible outputs.

As there are 256 characters, there are \( 256^n \) possible files of length \( n \). Since the compression is an injective map, these \( 256^n \) possible files must map to \( 256^n \) unique files.

Noting that there are \( \sum_{k=1}^{n-1} 256^k = 256^n - 1 \) files of size less than \( n \), then there cannot exist a compression mapping which compresses all possible files of length \( n \).

5. (20 points) Consider a \( 2^n \times 2^n \) board missing one cell. We want to cover this board with \( \frac{4^n-1}{3} \) tiles. The tiles are L-shaped and consist of three adjacent cells. We are given the location of the missing cell as input. Design a divide-and-conquer algorithm that achieves this task.

(a) Algorithm

**Algorithm 1** Tiling(n, location of the missing point)
1: \( \text{if } n == 2 \text{ then} \)
2: \( \text{Place a L-shape tile.} \)
3: \( \text{There is only one way to do the placement as it has one cell missing/covered.} \)
4: \( \text{return} \)
5: \( \text{else} \)
6: \( \text{Divide the board into four equal quadrants}(n/2 \times n/2). \)
7: \( \text{Place a tile in the center such that it does not cover the quadrant with the missing cell.} \)
8: \( \text{Deem the covered cells as new “missing” cells. Now every quadrant has one missing cell.} \)
9: \( \text{Recursively call Tiling for the four quadrants.} \)
10: \( \text{Tiling(n/2, 11);} \)
11: \( \text{Tiling(n/2, 12);} \)
12: \( \text{Tiling(n/2, 13);} \)
13: \( \text{Tiling(n/2, 14);} \)

Note that the location parameter is actually an enumeration variable chosen from four options: top right/left, bottom right/left.

(b) Correctness

Prove its correctness by induction.

i. Base case: There is only one way to place a L-shape tile on a \( 2 \times 2 \) board with one missing cell.
ii. I.H. A board of \( 2^{k-1} \times 2^{k-1} \) with a missing cell can be perfectly tiled by L-shape tiles.
iii. Induction Step: Place a tile in the centre of the \( 2^k \times 2^k \) board. Now we equally divide the board into four quadrants of \( 2^{k-1} \times 2^{k-1} \) with one missing cell each. By induction hypothesis, we can tile these boards using L-shape tiles. □

(c) Time Complexity

Assume it takes constant time to place a tile.
\[
T(1) = \Theta(1), \\
T(n) = 4T(n/2) + \Theta(1) = \Theta(n^2) \text{ by Master’s Theorem}
\]

(d) Number of tiles needed
\[
\sum_{i=1}^{n-1} 4^i = \frac{4^n - 1}{4 - 1} = \frac{4^n - 1}{3}
\]

Figure 2: Example of Tiling Algorithm (n = 8)

6. (20 points) We are given a directed graph on \( n \) vertices. Design a \( o(n^3) \) algorithm (note that this is little-o) that counts the number of triples \((v_1, v_2, v_3)\) such that \( v_1v_2, v_2v_3, v_3v_1 \) are all edges.

Solution: Construct a adjacency matrix \( M \) of the given directed graph where \( M_{ij} = 1 \) if there is an edge from vertex \( i \) to vertex \( j \) and 0 otherwise. We know, from matrix multiplication that:

\[
M^2 = \sum_{k=1}^{n} M_{ik}M_{kj}
\]

From this it can be seen that \( M^2_{ij} \) counts the number of paths of length 2 from \( i \) to \( j \). In the same way, \( M^3_{ij} \) would count the number of paths of length 3 from \( i \) to \( j \).

We want to count the number of triples \((v_1, v_2, v_3)\) such that \( v_1v_2, v_2v_3, v_3v_1 \) are all edges. So, we are looking for paths of length 3 that start and end at the same node. This can be found by calculating the \( \text{trace}(M^3) \). The time complexity would be dominated by calculating cube of \( M \). Using Strassen’s algorithm for matrix multiplication, the overall time complexity would be \( O(n^{\log_2 7}) \).