DDA6050 Assignment 1

Zihan, Zhou 119010484@link.cuhk.edu.cn

If you have any questions about grading, please feel free to reach out to me via email at 119010484@link.cuhk.edu.cn.

1 Array Selection Algorithm (25 pts)

In the following, for simplicity, you may assume that elements in the array are **unique**. Given an unsorted array of n distinct numbers, we want to find the k-th smallest element in this array.

(a) Optimized Selection Algorithm (15 pts)

Describe an algorithm that finds the k-th smallest element using at most $\mathcal{O}(n)$ comparisons in the worst case. (You do not need to write the code. You do not need to analyze the algorithm complexity here.)
Hints:

1. Divide the array into $\lceil n/5 \rceil$ groups.
2. Find the ??? (??? is one of minimum / maximum / medium) of each group.
3. Recursively find the ??? of ??? as the pivot.
4. Partition based on the pivot and recursively search.

(b) Recurrence Relation Analysis (5 pts)

Analyze the recurrence relation for the number of comparisons of your proposed algorithm.
Hints: your recurrence relation should be tighter or equal to

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + \mathcal{O}(n)$$

(c) Complexity Analysis (5 pts)

Analyze the complexity of your proposed algorithm using the big O notation in the worst case.

2 Search in Bitonic Sequence (10 pts)

Assume that the numbers in the array first decrease and then increase (bitonic), i.e., $a_1 > a_2 > \cdots > a_k$ and $a_k < a_{k+1} < \cdots < a_{n-1} < a_n$ for some $l < k < n$, but the value of k is unknown in advance. Some students proposed the following algorithm to check whether the array contains a query number x:

1. Divide the array into two halves at the middle point
2. Analyze the monotonicity of each half:
    - If one half is bitonic and the other is monotonic: recursively search the bitonic half, use binary search on the monotonic half.
    - If both halves are monotonic: use binary search on both halves.
    - If both halves are bitonic: this is impossible.

Return true if the target is found in either half.

```
function searchInBitonic(arr, left, right, target):
    if left > right:
        return false

    if left == right:
        return arr[left] == target

    mid = (left + right) / 2

    // Check if middle element is the target
    if arr[mid] == target:
        return true

    // Analyze monotonicity of left and right segments
    leftSegment = analyze_monotonicity(arr, left, mid)
    rightSegment = analyze_monotonicity(arr, mid+1, right)

    if leftSegment == "bitonic" and rightSegment == "monotonic":
        // Left half is bitonic, right half is monotonic
        return searchInBitonic(arr, left, mid, target) or
               binarySearch(arr, mid+1, right, target)

    else if leftSegment == "monotonic" and rightSegment == "bitonic":
        // Left half is monotonic, right half is bitonic
        return binarySearch(arr, left, mid, target) or
               searchInBitonic(arr, mid+1, right, target)

    else if leftSegment == "monotonic" and rightSegment == "monotonic":
        // Both halves are monotonic
        return binarySearch(arr, left, mid, target) or
               binarySearch(arr, mid+1, right, target)

function binarySearch(arr, left, right, target):
    // Standard binary search (need to determine if ascending or descending first)
    // Implementation omitted

function analyze_monotonicity(arr, left, right):
    // Analyze the monotonicity of the sequence segment
    // Returns "monotonic" or "bitonic"
    // Implementation omitted
```

Analyze the worst-case time complexity of the above algorithm using the big O notation.

3 Master Theorem (20 pts)

If $T(1) = \mathcal{O}(1)$, give asymptotic **upper** and **lower** bounds for $T(n)$ in each of the following recurrences. Make your bounds as tight as possible, and justify your answers. Note: You are only allowed to use the following version of the Master theorem, which is a slightly generalized one compared with that in the course slides and it is stated below. More generalized versions of the Master theorem are not allowed to be directly used.

Master Theorem: Let $T(n)$ be a monotonically increasing function that satisfies

$$T(n) = aT(n/b) + f(n)$$
$$T(1) = c$$

where $a \geq 1, b \geq 2$, $c > 0$. If $f(n)$ is $\Theta\left(n^d\right)$ where $d \geq 0$ then

$$T(n) = \begin{cases} \Theta\left(n^d\right) & \text{if } a < b^d \\ \Theta\left(n^d \log n\right) & \text{if } a = b^d \\ \Theta\left(n^{\log_b a}\right) & \text{if } a > b^d \end{cases}$$

(a). $T(n) = 3T(n/2) + n^2$

(b). $T(n) = 5T(n/4) + n(\log n)^2$

(c). $T(n) = 3T(\frac{n}{3} - 3) + \frac{n}{3}$

(d). $T(n) = 2T(\sqrt{n}) + \log n$

4 Recursion Tree (10 pts)

Plot a recursion tree and prove that $T(n) = \mathcal{O}(n \log^2 n)$ where $T(n)$ is given by the recurrence relation

$$T(n) = 3T(n/5) + T(2n/5) + n \log n.$$

Hint: Let $g(n) = n \log n$. Note that $ag(n) \leq g(an)$ for $a \geq 2$ and $g(n) + g(m) \leq g(n + m)$ for $n, m \geq 2$.

5 Password Generation Counting (10 pts)

A security system requires passwords to be generated according to specific rules:

- Passwords may only use digits 1 through 9 (digit 0 is prohibited)
- The sum of all digits in the password must equal a preset value S
- Password length is unlimited, but must contain at least one digit
- Leading zeros are not applicable since digit 0 is prohibited

Task: Given a target sum value S, design a recursive algorithm to calculate the total number of different passwords that satisfy the above conditions. First, state the recurrence relation and base cases (5 pts), and then write the pseudocode (5 pts). Your algorithm must achieve $\mathcal{O}(n)$ time complexity.

6 Ancient Pyramid Treasure Hunt (25 pts)

Explorer Alice has discovered an ancient pyramid ruin. The internal structure of this pyramid can be represented by an $n \times n$ grid map, where each cell has one of the following states:

- **0**: Safe passage - Alice can move through safely
- **1**: Treasure chest containing ancient gold coins - Alice can collect the coins and continue
- **-1**: Dangerous trap zone - impassable

Exploration Rules: Alice must complete a full exploration mission with the following constraints:

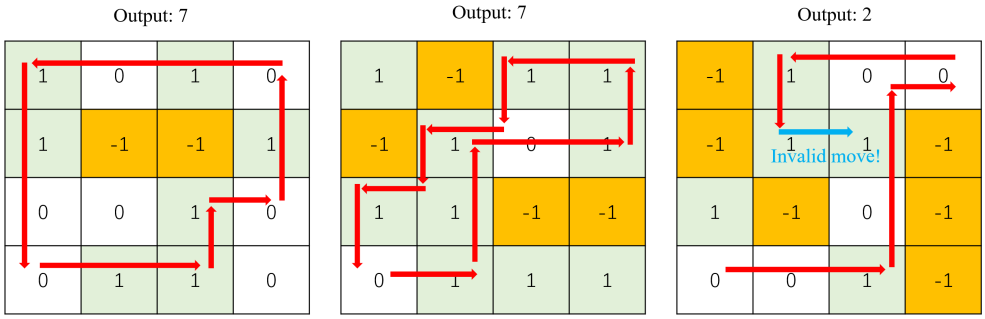
1. **Forward Journey**: Starting from the pyramid entrance at $(0, 0)$, Alice can only move **right** or **up**, and must reach the mysterious chamber at $(n-1, n-1)$.
2. **Return Journey**: From the mysterious chamber at $(n-1, n-1)$, Alice can only move **left** or **down**, and must return to the entrance at $(0, 0)$.
3. **Collection Rules**:
   - When Alice passes through a cell with a treasure chest, she automatically collects the gold coins
   - Once opened, the treasure chest becomes empty (value changes to 0) and cannot be collected again
   - If the forward and return journeys pass through the same treasure location, coins can only be collected once
4. **Movement Constraints**:
   - Alice can only walk in safe areas (cells with value 0 or 1)
   - If no viable path exists from the entrance to the mysterious chamber, the exploration fails

Objective: Calculate the maximum number of gold coins Alice can collect during this complete exploration. If failed, return 0.

Input Format:

- An $n \times n$ integer matrix `pyramid`, where `pyramid[i][j]` represents the state of position (i, j)

Examples:



Constraints:

- $n \geq 2$
- `pyramid[i][j]` $\in \{-1, 0, 1\}$
- `pyramid[0][0]` $\neq -1$ and `pyramid[n-1][n-1]` $\neq -1$ (entrance and chamber are guaranteed to be accessible)

Hints: Rotate the `pyramid[i][j]` grid 45 degrees counterclockwise to transform the problem into a more manageable coordinate system.
Task: Describe an algorithm to solve the problem with time complexity at most $\mathcal{O}(n^3)$. You do not need to write pseudocode. Focus on how to transform the original problem into a dynamic programming formulation by clearly defining the state space, state transitions, and recurrence relation. Your solution should address:

- State definition: What does each DP state represent?
- State transitions: How do you move from one state to another?
- Recurrence relation: What is the mathematical formula that relates current states to previous states?
- Base cases: What are the initial conditions?