SID: 510 444 907 14/03/2025

COMP3027 – Assignment 1

Problem 1 – Star Wars

Part A – Algorithm

- Sort clone data by ascending perishing time.
- Initialize an empty reference clone.
- Iterate through the sorted clone data and compare the reference clone with each clone. (Runs in $O(n)$ time)
  In each iteration:
  - If the reference clone is empty, fill the reference clone with the current clone. (Runs in $O(1)$ time)
  - If the compared clone's emergence time is less than or equal to the reference clone's perishing time, we add them to the bribe list. (Runs in $O(1)$ time)
  - If the compared clone's emergence time is greater than the reference clone's perishing time, we replace the reference clone with the currently compared clone. (Runs in $O(1)$ time)

Part B – Correctness

1. **Greedy Choice:**
   We claim that choosing clones with the earliest perishing times (smallest f_i) first will result in an optimal solution.
   Justification:
   - Suppose clones i and j overlap. Without loss of generality, let $f_i \leq f_j$.
   - Choosing the clone with earliest finishing time f_i maximizes the chance of covering multiple overlaps with fewer clones.
   - If we instead pick a clone with later finishing time f_j, we may need more clones to handle other overlapping intervals which earlier clones could have covered.

   Thus, sorting by earliest perishing time and picking clones accordingly is optimal.

2. **Exchange Argument:**
   Suppose for contradiction that there's a better (smaller) solution than our greedy choice.
   - Consider an optimal solution OPT different from the greedy solution $GREEDY$. Let $GREEDY$ select clone set $G = \{g_1. g_2, \ldots, g_k\}$, and OPT select clone set $O = \{o_1. o_2, \ldots, o_m\}$ with $m < k$.

Since the sets differ, there must exist at least one earliest-selected clone by greedy, say g_r, that is not in the optimal set.

- Since greedy selects clones by earliest perishing time, there must be clones overlapping with g_r that greedy covers immediately by including g_r.
- The optimal solution O covers this overlap as well, but by definition does not include g_r. Instead, it selects some other clone o_p later or earlier to cover overlaps.
- However, since the greedy algorithm always chooses the earliest finishing interval, the optimal choice must also cover the same overlaps.
- Replacing clone o_p with the greedy choice g_r would maintain the validity of coverage because g_r ends earlier (covers at least as much as o_p). Thus, no additional intervals are needed.
- Such an exchange would not increase the number of clones in OPT, meaning greedy is at least as optimal as any other optimal solution. Thus, by contradiction, our greedy choice yields an optimal solution.

Thus, by contradiction, our greedy choice yields an optimal solution.

Part C – Time Complexity

Using a merge sort to sort the clone data by ascending perishing times takes $O(n \log n)$ time.

Initializing an empty reference clone takes $O(1)$ time.

Iterating through the sorted clone data and compare the reference clone with each clone runs in $O(n)$ time.

In each iteration:

- Filling the reference clone with the current clone runs in $O(1)$ time.
- Adding a clone to the bribe list runs in $O(1)$ time.
- Replacing the reference clone with the currently compared clone runs in $O(1)$ time.

Summing these running times up, we get:

$$O(n \log n) + O(1) + O(n) + O(n) + O(1) + O(1) + O(1) = O(n \log n)$$

Hence, $O(n \log n)$ running time.

Problem 2 – Age of Empires

Part A – Algorithm

- All checks within Manhattan distance r from any cell runs in $O(4r^2)$ time

Iterate through the middle column and middle row and check each cell.

- Within each check, record the highest recorded value
- Within each check, record the position of the highest recorded value

Now that we know which quadrant the highest position is, we move to that quadrant and repeat the above process within that quadrant until a suitable cell is found.

Part B – Correctness

Base Case:

For sufficiently small grids, say size $n \leq c$ for some constant c (based on r), we have already explicitly verified all cells and their neighbours directly. The correctness for these grids is trivial since we are explicitly checking all neighbours for all cells, guaranteeing that we find a valid cell if it exists.

Inductive Step:

Inductive Hypothesis: Assume that for all grids smaller than $n \times n$ (specifically, smaller by at least half in each dimension), our divide-and-conquer algorithm correctly finds a local maximum (highest-value cell encountered).

Now, consider an $n \times n$ grid:

1. Check middle row and middle column:
   - Iterate through all cells along the middle row and middle column (at most $2n$ cells).
   - For each cell, examine neighbours within Manhattan distance r.
   - Each cell checked takes $O(4r^2)$ time.
   - During this check, record the highest-value cell encountered and its position.

2. Case Analysis:
   - Case 1: A valid cell is directly found on the middle row or column.
     If we find a cell in the middle row/column that is already valid, we are finished, and correctness is trivially achieved.
   - Case 2 (key insight clarified): No valid cell is found.
     If no valid cell is found, then by definition:
     - Every cell we checked (including the highest-value cell X) has at least one strictly greater neighbour within Manhattan distance r.

        - Hence, at least one of the cells we checked on the middle row or column has a neighbour X that's strictly greater.
        - Since X is strictly greater and located within distance r, it is necessarily located in exactly one quadrant. We now explicitly identify and move into this quadrant.

3. Further Reasoning:
    - Each recursive call moves us from an $n \times n$ grid to a smaller quadrant of dimensions approximately $\frac{n}{2} \times \frac{n}{2}$.
    - We have clearly identified this quadrant because we have explicitly found a cell X strictly greater than one of the cells we have previously checked.
    - Thus, the next recursion call is on a strictly smaller quadrant, approximately $\frac{n}{2} \times \frac{n}{2}$.
4. Recursive Call:
    - By the inductive hypothesis, we assume correctness for any smaller quadrant.
    - Thus, applying our algorithm recursively to this smaller quadrant guarantees correctness. It will continue recursively shrinking the grid, quartering the area each step, until the base case is reached, where the local maximum is trivially found.

Part C – Time Complexity

Iterating through the middle column and middle row runs in $O(2n)$. Adding a check to each cell causes the time complexity to run in $O(2n \times 4r^2)$.

The next iteration in one of the quadrants runs in $O(n)$. Adding a check to each cell causes the time complexity to run in $O(n \times 4r^2)$.

Similarly, the next iteration in one of the smaller quadrants runs in $O\left(\frac{n}{2}\right)$. Adding a check to each cell causes the time complexity to run in $O\left(\frac{n}{2} \times 4r^2\right)$.

And so on. To calculate all the tiles checked, we do a sum of cells checked per iteration, which leads to a geometric sum:

$$n + \left(\frac{n}{2} \times 4r^2\right) + \left(\frac{n}{4} \times 4r^2\right) + \left(\frac{n}{8} \times 4r^2\right) + \cdots = n \times 4r^2 \times \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\right)$$

As such, while treating $4r^2$ as a constant, we can get the following time complexity:

$$O\left(n \cancel{\times 4r^2 \times \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\right)}\right) = O(n)$$

Hence, $O(n)$ running time.