The University of Melbourne
School of Computing and Information Systems
COMP90038 Algorithms and Complexity

Assignment 1, Semester 2 2019

Released: Friday August 23 2019. Deadline: Sunday September 8 2019 23:59

Sample Solutions

Objectives

To improve your understanding of the time complexity of algorithms and recurrence relations. To develop problem-solving and design skills. To improve written communication skills; in particular the ability to present algorithms clearly, precisely and unambiguously.

Problems

Let's play a game. Or, more precisely, design and analyse some algorithms that might be used in a simple two-dimensional (2D) game.

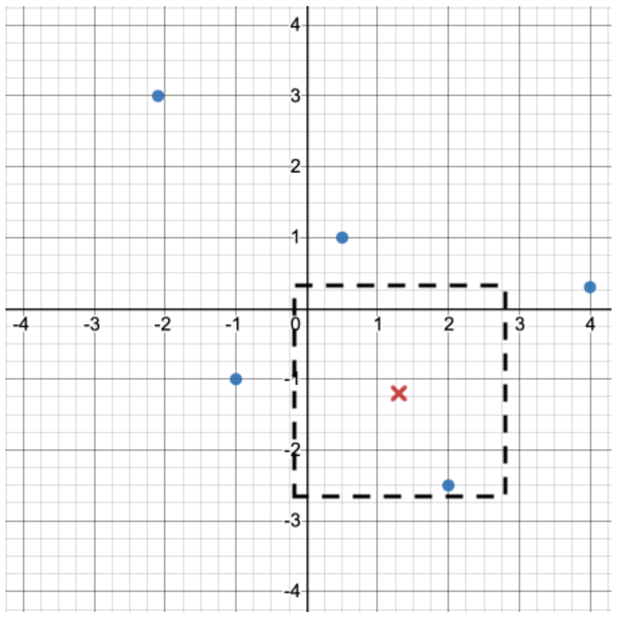

Figure 1: A tiny example game scenario. Enemy (AI-controlled) players are shown in blue. The fixed location of the human player is marked by the red cross.

Consider a game played on an two-dimensional grid, whose Cartesian coordinates range from $(-M, M) \dots (M, M)$. Figure 1 depicts a game board for which $M = 4$.

The game contains a fixed number N of enemy players, who are AI-controlled. Each player (including the human player and each enemy AI) is located at some arbitrary but fixed position (x, y) on the board, i.e. for which $-M \leq x \leq M$ and $-M \leq y \leq M$.

The human player can perform certain attacks, which damage all enemy players within a certain *range* of the human player. To avoid the need to calculate with expensive multiplication and square root operations, we will approximate the range by a square situated about the human player.

Specifically, given some fixed bound b, for which $0 \leq b \leq M$, if the human player is located at position (p_x, p_y) then all enemy players that lie within or on the borders of the box whose bottom-left corner is $(p_x - b, p_y - b)$ and whose top-right corner is $(p_x + b, p_y + b)$ are affected by the attack. Figure 1 depicts an example box for the bound $b = 1.5$.

1. [1 mark] Implement an $\Theta(1)$ function to determine whether an enemy at position (x, y) is affected by the player's attack, given the player's location (p_x, p_y):

```
function IsAffected($(p_x, p_y), (x, y), b$)
    ...
```

Solution:

```
function IsAffected($(p_x, p_y), (x, y), b$)
    return $(|p_x - x| \leq b \wedge |p_y - y| \leq b)$
```

2. [2 marks] Suppose (for now) that the positions of the enemy players are stored in an unsorted array $A[0] \ldots A[N-1]$.

The following function uses the *divide and conquer* approach to *mark* all enemy players affected by a player attack. Marking is implemented by the Mark function whose details are not important. (Note: the division below is integer division, i.e. it rounds down to the nearest integer.)

```
function MarkAffected($(p_x, p_y), A, b$)
    MarkAffectedDC($(p_x, p_y), A, 0, N - 1, b$)
function MarkAffectedDC($(p_x, p_y), A, lo, hi, b$)          ▷ assume $lo \leq hi$
    if $lo = hi$ then
        if IsAffected($(p_x, p_y), A[lo], b$) then
            Mark($A[lo]$)
    else
        $mid \leftarrow lo + (hi - lo)/2$
        MarkAffectedDC($(p_x, p_y), A, lo, mid, b$)
        MarkAffectedDC($(p_x, p_y), A, mid + 1, hi, b$)
```

Explain the purpose of the outermost "**if/else**" test. In particular, suppose we removed the line "**if** $lo = hi$" and the "**else**" line. In no more than a paragraph explain how this would affect the algorithm.

Solution:
Without it the algorithm would not terminate. It would also repeatedly mark the array elements, which could be problematic as Mark is not idempotent.

3. [2 marks] Consider the following recurrence relations. Which one best describes the worst-base complexity of the MarkAffectedDC function, whose input size is $n = hi - lo + 1$ and whose basic operations are calling IsAffected and Mark? Justify your answer in no more than a paragraph of text.

$T(1) = 1$	$T(1) = 2$	$T(1) = 0$	$T(1) = 2$
$T(n) = 2T(n/2)$	$T(n) = 2T(n/2) + 2$	$T(n) = 2T(n/2)$	$T(n) = 2T(n/2)$

Solution:
The final one. Why? n is $hi - lo + 1$ and, since $lo \leq hi$ always, $n \geq 1$. When $lo = hi$ (i.e. $n = 1$) there are two basic operations performed and no recursive calls. So $T(1) = 2$ is the correct base case. Otherwise, when $hi > lo$, no basic operations are performed other than those in the two recursive calls. Each of which works on approximately half the array, i.e. on a problem of size approximately $n/2$. Hence $T(n) = 2T(n/2)$.

4. [2 marks] Use telescoping (aka back substitution) to derive a closed form for your chosen recurrence relation and thereby prove that the worst case complexity of the MarkAffectedDC algorithm is in $\Theta(n)$.

Solution:

$$\begin{aligned} T(n) &= 2T(n/2) \\ &= 4T(n/4) \\ &= 8T(n/8) \\ &= 16T(n/16) \\ &= iT(n/i) \\ &= .. \\ &= nT(n/n) \\ &= nT(1) \\ &= n \cdot 2 \\ &= 2n \end{aligned}$$

Since $2n \in \Theta(n)$, the complexity is clearly in $\Theta(n)$.

5. [2 marks] Can we do better than this? Recall that the human player is at some *fixed* location (p_x, p_y). Your task is to work out how you would sort the array A so that those enemy AIs that need to be marked can be identified in $\log(N)$ time.

   Specifically, complete the following comparison function that would be used while sorting the array A. Here, (x_1, y_1) and (x_2, y_2) are two points from the array A. The function should return **true** if it considers the first point to be less than or equal to the second, and should return **false** otherwise. Your function can use the player's coordinate (p_x, p_y) as global variables, i.e. you are allowed to refer to p_x and p_y in your function.

```
function LessOrEqualTo($(x_1, y_1), (x_2, y_2)$)
    ...
```

Solution:

```
function LessOrEqualTo($(x_1, y_1), (x_2, y_2)$)
    return $(\text{Max}(|x_1 - p_x|, |y_1 - p_y|) \leq \text{Max}(|x_2 - p_x|, |y_2 - p_y|))$
```

6. [3 marks] Now, supposing the array has been sorted using your comparison function, implement an algorithm whose worst case complexity is in $\Theta(\log(N))$ that determines which array elements should be marked. Your function should take the bound b as an argument, and may also take the player's coorinates (p_x, p_y).

Solution:
Binary search for the last element (x, y) in the array for which $\textsc{Max}(|x - p_x|,|y - p_y|) \leq b$. Then this element plus all prior ones in the array need to be marked. -1 is returned when no element should be marked.

```
function FindLastMarkIndex($A, N$)
    $lo \leftarrow 0$
    $hi \leftarrow N - 1$
    while $lo \leq hi$ do
        $mid \leftarrow (lo + hi)/2$
        if IsAffected($(p_x, p_y), A[mid]$) then
            $lo \leftarrow mid + 1$
        else
            $hi \leftarrow mid - 1$
    return $hi$
```

7. [2 marks] How many elements will need to be marked in the worst case and what, therefore, would be the worst-case complexity of an algorithm that, given an array A sorted according to your comparison function, implemented the behaviour of MarkAffected above? Express your answer in Big-Theta notation.

Solution:
N. Therefore worse case complexity is in $\Theta(N)$. Hence, it is no better than brute force asymptotically speaking.

8. [2 marks] The worst case is quite pessimistic. On average, we might expect that enemy AIs are uniformly distributed throughout the game board.

   Let d be the expected number of enemy AIs contained in or on the borders of a box of bound b (i.e. whose width and height are each $2b$) on the board. For simplicity, we limit our attention to boxes that are contained entirely within the game baord.

   Consider an algorithm that, given an array A sorted according to your comparison function, implemented the behaviour of MarkAffected above by first using your $\Theta(log(N))$ function to determine the elements that need to be marked (of which we expect there to be d), and then marking those d elements. We might characterise its expected complexity as:

$$\log(N) + d$$

   Derive a formula for d in terms of b, M and N.

Solution:
There are N AIs in the $2M \times 2M$ game board, i.e. the density is $\frac{N}{4M^2}$. A $2b \times 2b$ bounding box has area $4b^2$. Hence, we should expect $\frac{N}{4M^2} \cdot 4b^2$ AIs in each bounding box, i.e. $d = \frac{b^2}{M^2}N$.

9. [2 marks] As a point of comparison, what is the expected complexity of the divide-and-conquer algorithm above in this average case, expressed in Big-Theta notation? Justify your answer in no more than a paragraph of text.

Solution:
$\Theta(N)$, since the algorithm has to perform N basic operations (calls to IsAffected) no matter how many AIs need to be marked.

10. [2 marks] Now compare the best case complexity of the divide-and-conquer algorithm and the one that uses a pre-sorted array A described above. Express the best-case complexity of each in Big-Theta notation as a function of N. Justify both in no more than a paragraph of text.

Solution:
The best case for the decrease-and-conquer algorithm still performs N calls to IsAffected. Hence its best case complexity is in $\Theta(N)$. On the other hand the best case for the other algorithm is when it has to mark no AIs, in which case $d = 0$ and so its complexity is in $\Theta(\log(N))$.

11. [4 marks] Suppose now that the enemy AIs can send each other messages. However, two AIs, whose positions are (x_1, y_1) and (x_2, y_2) respectively, can directly communicate only if the line (x_1, y_1) — (x_2, y_2) that connects them does not pass through the rectangle (including its borders) surrounding the human player whose bottom left corner is $(p_x - b, p_y - b)$ and whose top-right corner is $(p_x + b, p_y + b)$, where (p_x, p_y) is the human player's position.

Implement a function that determines whether two enemy AIs can directly communicate. Include a brief description (no more than a single paragraph) of how your function works, i.e. the rationale behind its design.

```
function CanDirectlyCommunicate($(x_1, y_1), (x_2, y_2), (p_x, p_y), b$)
    ...
```

Solution:
If either node is in the box then we must return false. If both nodes are on the right (or left / upper / bottom) side of the right (or left / upper / bottom) boundary of the box, we should return true. Otherwise, we test whether all four corners of the box is on the same side of the line connecting two nodes.

```
function ORIENTATION($(x_1, y_1), (x_2, y_2), (x_3, y_3)$)
    if $(x_2 - x_1) \cdot (y_3 - y_1) - (y_2 - y_1) \cdot (x_3 - x_1) > 0$ then
        return “r”
    else if $(x_2 - x_1) \cdot (y_3 - y_1) - (y_2 - y_1) \cdot (x_3 - x_1) < 0$ then
        return “l”
    else
        return “c”
function CANDIRECTLYCOMMUNICATE($(x_1, y_1), (x_2, y_2), (p_x, p_y)$)
    if ISAFFECTED($(p_x, p_y), (x_1, y_1)$) $\vee$ ISAFFECTED($(p_x, p_y), (x_2, y_2)$) then
        return false
    if $((x_1 - p_x) > b \wedge (x_2 - p_x) > b) \vee$
$((x_1 - p_x) < -b \wedge (x_2 - p_x) < -b) \vee$
$((y_1 - p_y) > b \wedge (y_2 - p_y) > b) \vee$
$((y_1 - p_y) < -b \wedge (y_2 - p_y) < -b)$ then
        return true
    $(x_{c1}, y_{c1}) \leftarrow (p_x - b, p_y - b)$
    $(x_{c2}, y_{c2}) \leftarrow (p_x - b, p_y + b)$
    $(x_{c3}, y_{c3}) \leftarrow (p_x + b, p_y - b)$
    $(x_{c4}, y_{c4}) \leftarrow (p_x + b, p_y + b)$
    if ORIENTATION($(x_1, y_1), (x_2, y_2), (x_{c1}, y_{c1})$) =
ORIENTATION($(x_1, y_1), (x_2, y_2), (x_{c2}, y_{c2})$) =
ORIENTATION($(x_1, y_1), (x_2, y_2), (x_{c3}, y_{c3})$) =
ORIENTATION($(x_1, y_1), (x_2, y_2), (x_{c4}, y_{c4})$) then
        return true
    else
        return false
```

12. [6 marks] Imagine that your CANDIRECTLYCOMMUNICATE function has been used to construct a graph whose nodes are the enemy AIs, such that there exists an edge between node n_1 and n_2 if and only if those two AIs can directly communicate. This graph is stored as an adjacency matrix M. Here the nodes n_i are simply the indices of enemy AIs in the array A, so each of them is simply an integer in the range $0 \leq n_i \leq N - 1$. Therefore the adjacency matrix M is simply a two-dimensional array, $M[0][0] \ldots M[N-1][N-1]$.

    Implement an $O(N^2)$ algorithm that, given two enemy AIs n_1 and n_2 determines whether there exists a path by which they might communicate via other enemy AIs. If a path exists, your algorithm should return the shortest path by which communication is possible. Otherwise, your algorithm should return the empty path. Your algorithm also takes as its input the adjacency matrix representation of the graph described above.

    You should represent a path as a linked list of enemy AI indices n_i. The empty path is therefore the empty linked list without any nodes.

Solution:
Using BFS, start from the source node s, mark each node with its source node. If the destination node t is reached, use backtracking to construct the path.

```
function COMMUNICATE($s, t, M, N$)
    init $source[0..N-1] = \{-1..-1\}$
    $init(queue)$
    $source[s] \leftarrow s$
    $inject(queue, s)$
    while $queue$ is non-empty do
        $u \leftarrow eject(queue)$
        for $v \leftarrow 0$ to $N-1$ do
            if $M[u][v] = 1 \wedge source[v] = -1$ then
                $source[v] \leftarrow u$
                $inject(queue, v)$
    if $source[t] = -1$ then
        return null
    else
        init linked list $path \leftarrow$ null
        $p \leftarrow t$
        while $p \neq s$ do
            init node $x$
            $x.val \leftarrow p$
            $x.next \leftarrow path$
            $path \leftarrow x$
            $p \leftarrow source[p]$
        init node $x$
        $x.val \leftarrow s$
        $x.next \leftarrow path$
        $path \leftarrow x$
        return $path$
```

Submission and Evaluation

- You must submit a PDF document via the LMS. Note: handwritten, scanned images, and/or Microsoft Word submissions are not acceptable — if you use Word, create a PDF version for submission.
- Marks are primarily allocated for correctness, but elegance of algorithms and how clearly you communicate your thinking will also be taken into account. Where indicated, the complexity of algorithms also matters.
- We expect your work to be neat – parts of your submission that are difficult to read or decipher will be deemed incorrect. Make sure that you have enough time towards the end of the assignment to present your solutions carefully. Time you put in early will usually turn out to be more productive than a last-minute effort.
- You are reminded that your submission for this assignment is to be your own individual work. For many students, discussions with friends will form a natural part of the undertaking of the assignment work. However, it is still an individual task. You should not share your answers (even draft solutions) with other students. Do not post solutions (or even partial solutions) on social media or the discussion board. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the student concerned.

  Please see `https://academicintegrity.unimelb.edu.au`

If you have any questions, you are welcome to post them on the LMS discussion board *so long as you do not reveal details about your own solutions.* You can also email the Head Tutor, Lianglu Pan (lianglu.pan@unimelb.edu.au) or the Lecturer, Toby Murray (toby.murray@unimelb.edu.au). In your message, make sure you include COMP90038 in the subject line. In the body of your message, include a precise description of the problem.

Late Submission and Extension

Late submission will be possible, but a late submission penalty will apply: a flagfall of 2 marks, and then 1 mark per 12 hours late.

Extensions will only be awarded in extreme/emergency cases, assuming appropriate documentation is provided – simply submitting a medical certificate on the due date will not result in an extension.