CS/ECE 374 P18

Jiawei Tang, Pengxu Zheng, Junquan Chen

TOTAL POINTS

64.5 / 100

QUESTION 1

1 Problem 18.A. **44.5 / 70**

+ **7 pts** Clear English description of the function you are trying to evaluate.
- ✓ + **7 pts** **Stated how to call your function to get the final answer.**
+ **7 pts** Correct base case(s).
- ✓ + **3.5 pts** **MINOR BUG: like a typo or an off-by-one error in base case(s).**
- ✓ + **21 pts** **Correct recursive case(s). No credit for the rest of the problem if the recursive case(s) are incorrect.**
+ **14 pts** MINOR BUG: like a typo or an off-by-one error in recursive cases.
- ✓ + **7 pts** **Described the memoization data structure.**
- ✓ + **14 pts** **Described a correct evaluation order; a clear picture is usually sufficient.**
- ✓ + **7 pts** **Correct time complexity.**
+ **17.5 pts** IDK
- ✓ - **15 pts** **Extra penalty for not having English description**
- **10 pts** Using code (that is hard to read) rather than pseudocode
+ **0 pts** Incorrect (see comments below)

Base cases are not included in iterative solution.
No English description of solution.

QUESTION 2

2 Problem 18.B. **20 / 30**

- **10** **Point adjustment**

Suboptimal runtime

HW Solution

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.0**

Submitted by:

- **≪Jiawei Tang≫**: **≪jiaweit2≫**
- **≪Junquan Chen≫**: **≪junquan2≫**
- **≪Pengxu Zheng≫**: **≪pzheng5≫**

18

Solution:

A)

Let $FC(i, j)$ denote whether a solution exists, where $0 \le i \le n$, $0 \le j \le L$. There are two cases: s_i will either be $+1$ or -1. This function obeys the following recurrence:

$$\text{FC(i,j)} = \begin{cases} \text{TRUE}, & \text{if } i = n+1 \\ \text{FALSE}, & \text{if } j \pm a_i \notin [0, L] \\ (FC(i+1, j+a_i) \text{ or } FC(i+1, j-a_i)) & \text{otherwise} \end{cases}$$

We can memorize the function FC into an array $FC[0..n+1, 0..L]$. Each entry $FC[i, j]$ depends on entries in the next row so we fill the array in reverse row-major order.

> **for** $i \leftarrow n$ down to 0 **do**
>   **for** $j \leftarrow L$ down to 0 **do**
>   **if** $(j + a_i \in [0, L]$ and $M[i+1, j+a_i]$ ==TRUE) or $(j - a_i \in [0, L]$ and $M[i+1, j-a_i]$ ==TRUE) **then**
> $M[i, j]$ =TRUE
>   **else**
> $M[i, j]$ =FALSE
>   **end if**
>   **end for**
> **end for**
> **for** $j \leftarrow L$ down to 0 **do**
>   **if** $M[0, j]$ ==TRUE **then**
>   return TRUE
>   **end if**
> **end for**
> return FALSE

Basically we are trying to fill up the $n \times L$ table. There are $O(nL)$ subproblems and each requires $O(1)$ time. Therefore, the total time is $O(nL)$.

1 Problem 18.A. **44.5 / 70**

- \+ **7 pts** Clear English description of the function you are trying to evaluate.
- ✓ + **7 pts** **Stated how to call your function to get the final answer.**
- \+ **7 pts** Correct base case(s).
- ✓ + **3.5 pts** **MINOR BUG: like a typo or an off-by-one error in base case(s).**
- ✓ + **21 pts** **Correct recursive case(s). No credit for the rest of the problem if the recursive case(s) are incorrect.**
- \+ **14 pts** MINOR BUG: like a typo or an off-by-one error in recursive cases.
- ✓ + **7 pts** **Described the memoization data structure.**
- ✓ + **14 pts** **Described a correct evaluation order; a clear picture is usually sufficient.**
- ✓ + **7 pts** **Correct time complexity.**
- \+ **17.5 pts** IDK
- ✓ - **15 pts** **Extra penalty for not having English description**
- \- **10 pts** Using code (that is hard to read) rather than pseudocode
- \+ **0 pts** Incorrect (see comments below)

Base cases are not included in iterative solution. No English description of solution.

B)
(Inspired by Bo Wang's group)
L* must be greater or equal to max$\{a1...an\}$.
Proof by contradiction: Assume L* is less than max $\{a1...an\}$ However, $j + max\{a1...an\} >$ L* and $j - max\{a1...an\} < L^*$. There is a contradiction. Therefore, L^* must be greater than or equals to $max\{A_1, ..., A_n\}$.

L^* must be less than or equal to $2 * max\{A_1, ..., A_n\}$ By the recursive case in 18a, we knew that when $L \geq 2 * max\{A_1, ..., A_n\}$ for every i and j, at least either of $(j + a_i)$ or $(j - a_i)$ should be $\in [0, L]$. This indicates that a valid folding must exist when $L \geq 2 * max\{A_1, ..., A_n\}$. Then we can conclude that L^* must be less than or equal to $2 * max\{A_1, ..., A_n\}$
Therefore, we can conclude that, $max\{A_1, ..., A_n\} \leq L^* \leq 2 * max\{A_1, ..., A_n\}$ and for every $L \geq L^*$, there exists a valid folding.

Apply binary search technique to to L^*

Let max be the $max\{A_1, ..., A_n\}$. We need to reuse the function FC in 18a to check whether there exists a valid folding when $L = max - 1$, we know by the above proof that, this call will return False. We call the function again to check if there exists a valid folding when $L = 2 * max$. By the proof above, we know that this call will return True.

Let L denotes the midpoint of $max - 1$ and $2 * max$. Call FC to check if there exist a valid fold when $L = ((max - 1) + 2 * max)/2$ If True, meaning there is a valid fold, let $L = ((max - 1) + ((max - 1) + 2 * max)/2)/2$, meaning L equals to the midpoint of the $m - 1$ and $((max - 1) + 2 * max)/2$. If the FC returns false,let $L =$ midpoint of $((max - 1) + 2 * max)/2$ and $2 * max$. We keep testing on the midpoint of two Ls that first L doesn't have a valid fold while second L has a valid fold. Then keep iterating until that the midpoint is equal to first point.
At last, when we hit the final midpoint, L = midpoint, if FC returns True, then $L^* = midpoint$, if FC returns False, then $L^* = midpoint + 1$

Runtime Analysis:

As described above, max denotes $max\{A_1, ..., A_n\}$ and our binary search is searching an arrary with length of max. We've called $O(log(m))$ times of FC in this process. it costs $O(n * 2 * L^*)$ for each call. In total, the running time should be O(log($max\{A_1, ..., A_n\}$)$n * L^*$)

2 Problem 18.B. **20 / 30**

- **- 10 Point adjustment**

  Suboptimal runtime