Homework 2

ECE345 - Group 16

October 13th, 2023

Total pages: 21

Team Member	Student Number
Ardavan Alaei Fard	1007934620
Rowan Honeywell	1007972945
Isaac Muscat	1007897135

Contents

Question 1

Question 2

Question 3

(a)

(b)

(c)

(d)

Question 4

(a)

(b)

The answer to this problem will be shown through a progression of algorithms which output the number of deadlines handled by Xun given optimal choices, from the most time-complex to the least. For the initial approach to this problem, consider the pseudocode below:

Q4B-WORST($d, i, N, D, Xsum, Worker$)

1   **if** $N - i < 3D$
2   **if** $Worker = Xun$
3           $Xsum = Xsum + \text{SUM}(d, i, N)$
4   **return** $Xsum$
5   **if** $Worker = Xun$
6       $max\text{-}deadlines = -\infty$
7   **for** $X = 1$ **to** $3D$
8           $new\text{-}i = i + X + 1$
9           $new\text{-}D = \max(X, D)$
10          $new\text{-}Xsum = Xsum + \text{SUM}(d, i, X)$
11          $deadlines = \text{Q4B-WORST}(d, new\text{-}i, N, new\text{-}D, new\text{-}Xsum, Yuntao)$
12          $max\text{-}deadlines = \max(max\text{-}deadlines, deadlines)$
13   **return** $max\text{-}deadlines$
14 **if** $Worker = Yuntao$
15      $min\text{-}deadlines = \infty$
16   **for** $X = 1$ **to** $3D$
17          $new\text{-}i = i + X + 1$
18          $new\text{-}D = \max(X, D)$
19          $deadlines = \text{Q4B-WORST}(d, new\text{-}i, N, new\text{-}D, Xsum, Xun)$
20          $min\text{-}deadlines = \min(min\text{-}deadlines, deadlines)$
21   **return** $min\text{-}deadlines$

Explanation of Q4b-Worst**:**

The pseudocode above uses the minimax principle, where one person is trying to maximize some score, and the other is trying to minimize it. In this case, the score in question is the total number of deadlines handled by Xun, where Xun is also the person who is trying to maximize the score. The base case in this recursive function is when the size of the passed range $(i - j)$ is less than or equal to $3D$. This is because both Xun and Yuntao choose optimally, so if either of them have the chance to take the remainder of the deadlines, then they will.

Initially, this function will be called as:

$$\text{Q4b-Worst}(d, 1, N, 1, 0, Xun)$$

This passes in the full array of deadlines, sets $D = 1$, the number of Xun's deadlines $(Xsum)$ to 0, and sets Xun as the person currently working. When Xun is working (lines 5 - 13), each value of X, $1 \leq X \leq 3D$ is tried by calling the function recursively, now with $X + 1$ as the new start of the deadline array, $D = \max(X, D)$, the sum of deadlines from i to X is added to $Xsum$, and finally the worker is now Yuntao. Xun's tests return the maximum number of deadlines possible for a certain starting point in the deadline list, given that Yuntao is also making optimal decisions.

When it is Yuntao's turn (lines 14 - 21), he is trying to minimize the number of deadlines handled by Xun. This part of the algorithm is fundamentally the same as Xun's, however instead of returning a recursive maximum, it returns a recursive minimum. Also, instead of updating $Xsum$, it simply is left as how it was passed. This is because Yuntao is **taking away** X deadlines from Xun, not adding them.

Analysis of Q4B-WORST**:**

There are two main reasons why this algorithm is the "worst". First, each time that a sum

is computed, the array of deadlines must be iterated through:

SUM(d, i, j)
1   $sum = 0$
2   **for** $x = i$ **to** j
3        $sum = sum + d[x]$
4   **return** sum

Asymptotically, the SUM function is $\mathcal{O}(n)$ in time since the distance between i and j is only

bounded by the size of d.

The second reason why Q4B-WORST is poor, is because of the number of recursive calls

that it makes. For each sub-array, there is some chosen X that divides it into two more

sub-arrays, $d[i$ **to** $X]$ and $d[X + 1$ **to** $N]$. We can therefore express the number of sub-array

divisions, S, that occurs as:

$$S(n) = \begin{cases} 1 \Rightarrow n = 1 \\ \sum_{X=1}^{n-1}(S(X) + S(n - X)) \Rightarrow n \geq 2 \end{cases}$$

By expanding and simplifying the sum in the second case, we get:

$$S(n) = \begin{cases} 1 \Rightarrow n = 1 \\ 2\sum_{X=1}^{n-1} S(X) \Rightarrow n \geq 2 \end{cases}$$

This sum can be simplified to (CLRS, 3rd Edition, Pg. 364, Eqn. 15.4):

$$2\sum_{X=1}^{n-1} S(X) = 2(2^{n-1}) = 2^n$$

Therefore, it is noted that the number of recursive calls that are required grows exponentially with the size of the input array d. Since, for each recursive division, a sum must be calculated (in $\mathcal{O}(n)$ time), the total time complexity of this algorithm is asymptotically $\mathcal{O}(n2^n)$.

Introducing dynamic programming: The aforementioned issues can be mitigated through the use of dynamic programming. To begin, we can use a bottom-up approach to store the sum of each possible sub-array for $\mathcal{O}(1)$ use in the Q4B algorithm.

BOTTOMUPSUMS(d, N)

1   Let s by an $N \times N$ array of zeros
2   **for** $i = 1$ **to** N
3        $s[i, i] = d[i]$
4   **for** $l = 2$ **to** N
5   **for** $i = 1$ **to** $N - l + 1$
6            $j = i + l - 1$
7            $s[i, j] = s[i, j - 1] + s[j, j]$
8   **return** s

In the algorithm above, lines 2 and 3 first find the sums of the smallest possible sub-arrays, which are single indices. From there, the **for** loop at line 4 specifies the size of the next sub-arrays whose sums will be calculated. For the corresponding size, l, the nested **for** loop (lines 5 - 7) finds two values in s which will directly add up to the correct sum. For example, it will calculate the sum of a sub-array from indices i to j, by adding the sums of the sub-arrays from i to $j - 1$ and j to j. Since the sub-arrays are calculated in increasing order of length (line 4) there will have already been sums stored for the sub-arrays from i to $j - 1$ and j to j. Since there are $\binom{N}{2}$ total sub-arrays, the space complexity of this algorithm is $\mathcal{O}(n^2)$. In terms of time, the bottom-up approach allows this algorithm to be $\mathcal{O}(n^2)$ as well, since each $m[i, j]$ takes $\mathcal{O}(1)$ to calculate.

Next, we can address the issue of how many recursive calls are made. The dynamic programming approach that must be used is top-down, or "memoization". It is important to note that the bottom-up approach can not be used for this task since the possible outcomes given a specific sub-array depend on the value of D (a value that is updated from the top-down) that is passed. Therefore, a structure that we can use to store such data is a hash table, where the key is a tuple of three values ($Worker$, D, i), and the value is the outcome given the respective configuration. This tuple contains i, the starting point of the remainder of the array after some set of deadlines are chosen, the corresponding D value, and the worker (Xun or Yuntao) whose turn is next.

This addition greatly reduces the number of recursive calls that are made. In short, there are a total of n sub-arrays that span some index i to the end of the full array. Letting k represent the number of possible values D may take, the total number of recursive calls that needs to be made is $\mathcal{O}(kn)$. However, since the value of D is only bounded by n, we can express the new total number of recursive calls as $\mathcal{O}(n^2)$.

Consider the pseudocode below:

INITMEMO(N)
1   Let m be a hash table that takes keys of three-integer tuples
2   **for** $D = 1$ **to** N
3   **for** $i = 1$ **to** N
4           $m[(Xun, D, i)] = -1$
5           $m[(Yuntao, D, i)] = -1$
6   **return** m

The above pseudocode shows how the hash table used to memoize the tuples would be initialized, with each unseen configuration set to -1.

We must also consider the space requirements of this incorporation. Since we are storing a

single integer for $2n^2$ possible pairs, the space complexity of this structure is $\mathcal{O}(n^2)$.

Finally, let us consider a much faster algorithm:

Q4B-BETTER$(d, i, N, D, Xsum, Worker, s, m)$

```
 1   if N − i < 3D
 2       if Worker = Xun
 3           Xsum = Xsum + s[i, N]  // Getting a sum in O(1) time
 4       return Xsum
 5   if Worker = Xun
 6       max-deadlines = −∞
 7       for X = 1 to 3D
 8           if m[(Yuntao, D, X + 1)] ≠ −1  // Checking for memoization
 9               deadlines = m[(Yuntao, D, X + 1)]  // Memoized - no recursive call needed
10           else
11               new-Xsum = Xsum + s[i, X]  // Getting a sum in O(1) time
12               deadlines = Q4B-BETTER(d, i + X + 1, N, max(X, D), new-Xsum, Yuntao, s, m)
13               m[(Yuntao, D, X + 1)] = deadlines − Xsum  // Memoization of the final result
14               m[(Xun, D, X + 1)] = m[(Yuntao, D, X + 1)] − s[X + 1, N]
15           max-deadlines = max(max-deadlines, deadlines)
16       return max-deadlines
17   if Worker = Yuntao
18       min-deadlines = ∞
19       for X = 1 to 3D
20           if m[(Xun, D, X + 1)] ≠ −1  // Checking for memoization
21               deadlines = m[(Xun, D, X + 1)]  // Memoized - no recursive call needed
22           else
23               deadlines = Q4B-BETTER(d, i + X + 1, N, max(X, D), Xsum, Xun, s, m)
24               m[(Xun, D, X + 1)] = deadlines − Xsum  // Memoization of the final result
25               m[(Yuntao, D, X + 1)] = m[(Xun, D, X + 1)] − s[X + 1, N]
26           min-deadlines = min(min-deadlines, deadlines)
27       return min-deadlines
```

Explanation of Q4B-BETTER**:**

The algorithm above takes two more parameters than the "worst" one. These parameters, as previously described, are the table of sums, s, and the memo, m. The only differences between this algorithm and Q4B-BETTER is within line 3, lines 8 - 14 and lines 20 - 25. However, lines 8 - 14 and lines 20 - 25 are the also the same despite the former being attributed to Xun's decision, and the latter to Yuntao's.

For each choice that can be made for $1 \leq X \leq 3D$, we must first check if what would be left of the array of deadlines has already been evaluated in a previous call (line 8). Since line 8 falls under Xun's choice, the worker who would be assigned to a given choice would be Yuntao. Therefore, the memo, m, is checked for the tuple $(Yuntao, D, X+1)$. If it exists within m, then we can simply set the deadline count for that iteration (line 9). Otherwise, we call the algorithm recursively as we orginally did. Now that a recursive call has been made, we can update m. For the next worker, Yuntao, we must store the value of the recursion output **minus** the sum, $Xsum$, that was passed (line 13). The reason that this is done is to treat each sub-array as if it is its own set of deadlines, separate from the full array. Line 14 shows an interesting shortcut we can take. Instead of waiting for a different recursive call to memoize the sub-array for the other worker, we can directly memoize it since it would be the "sub-complement" of value that was previously memoized. Basically, what this is saying is that if there are N total deadlines and Xun handles k of them, then Yuntao must handle $N - k$.

Getting a final answer:

Finally, since the Q4B algorithm only gives us the number of deadlines that Xun handles, we can find who handles the most deadlines by the same principle used in the last sentence

of the previous section. Therefore, consider the pseudocode below:

Q4B-FINALANSWER(d, N)

1 $s = $ BOTTOMUPSUMS(d, N)
2 $m = $ INITMEMO(N)
3 $deadlines\text{-}Xun = $ Q4B-BETTER$(d, i, N, D, 0, Xun, s, m)$
4 **if** $deadlines\text{-}Xun > \frac{1}{2}s[1, N]$
5   **return** Xun
6 **elseif** $deadlines\text{-}Xun < \frac{1}{2}s[1, N]$
7   **return** $Yuntao$
8 **else**
9   **return** Tie

If the number of deadlines that Xun handles is greater than half of the total deadlines, then she handles the most. Otherwise if it is less, then Yuntao does. Finally, if the number of deadlines handled by each worker is equal, then it is a tie.

Summary:

In the solution to this question, we first examined the baseline minimax implementation of the Xun and Yuntao's strategies, Q4B-WORST. Then, we introduced dynamic programming to greatly decrease the original algorithm's time complexity. This was done in two steps:

- A bottom-up approach for summing sub-arrays

- A top-down (memoization) approach for storing the output of deadlines completed by Xun within a certain sub-array

The first optimization in Q4B-BETTER allowed for an $\mathcal{O}(1)$ retrieval of sums instead of $\mathcal{O}(n)$, and the second optimization limited the number of recursive calls that needed to be made to $\mathcal{O}(n^3)$.