Problem 1.

Dynamic Programming

McDungle’s is considering opening a series of restaurants along Route 7. The $n$ possible locations are along a straight line, and the distances of these locations from the start of Route 7 are, in miles and in increasing order, $m_1, m_2, \ldots, m_n$. The constraints are as follows:

- At each location, McDungle’s may open at most one restaurant. The expected profit from opening a restaurant at location $i$ is $p_i$, where $p_i > 0$ and $i = 1, 2, \ldots, n$.
- Any two restaurants should be at least $k$ miles apart, where $k$ is a positive integer.

McDungle’s would like to find the set of locations that will yield the maximum total expected profit. For example, if the $n$ locations are at miles: 0, 2, 5, 7, 10 and the $n$ corresponding expected profits are: 2, 5, 4, 3, 1, and $k = 4$, then the optimal solution is the 2nd and 4th locations (i.e. at miles 2 and 7) with expected profit $= 5+3 = 8$.

(a) [21 points] Give an efficient algorithm to compute the maximum expected total profit subject to the given constraints. As we’ve done in class, your solution should describe the following:

- [2] The size of the array or matrix you will use to compute your solution.
- [3] What each entry of the array or matrix holds (in words).
- [7] The dynamic programming formulation (how to fill each entry, i.e. mathematically), including the base cases.
- [3] How to fill the entire array.
- [2] Where the optimal value is located once the array is filled.

You do not need to show how to find the locations.

(b) [15 points] Write a program (either Java or python) to verify that your algorithm is correct. The program should take as command line $n$, $k$, the $n$ mile markers and the $n$ corresponding expected profits, and output the maximum total expected profit.

For the example above, the command line arguments should be:

```
5 4 0 2 5 7 10 2 5 4 3 1
```

and the output should be:

```
8
```

You do not need to worry about finding the locations.

Submit your program via the link on canvas.

Problem 2.

Subset Sum

(a) [10 pts] The following is input to the 01-Knapsack problem:

<table>
<thead>
<tr>
<th>item</th>
<th>weight</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

$W$ (knapsack capacity) = 5
Apply the dynamic programming algorithm we discussed to fill in the following table and find the optimal value we can achieve (do not worry about the actual items). In the table, the columns represent the weight limits and the rows represent the item numbers.

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
0 & & & & & \\
1 & & & & & \\
2 & & & & & \\
3 & & & & & \\
4 & & & & & \\
\end{array}
\]

(b) [21 points] Now consider a new problem: Given a set of positive integers \( L = \{x_1, x_2, \ldots, x_n\} \) and a target \( b \), the Subset Sum problem asks you to find \( S \subseteq L \) such that the sum of the elements in \( S \) is no greater than \( b \) but is also as close to \( b \) as possible.

Give an efficient dynamic programming solution to find the best value (i.e., the sum closest to \( b \), not the actual set of integers). As usual, your solution should include:

- [2] The size of the array or matrix you will use to compute your solution.
- [3] What each entry of the array or matrix holds (in words).
- [7] The dynamic programming formulation (how to fill each entry, i.e., mathematically), including the base cases.
- [3] How to fill the entire array.
- [2] Where the optimal value is located once the array is filled.

(b) [6 points] Describe how to modify your solution if the \( x_i \)'s were not integers. Also describe how to modify your solution if \( b \) was not an integer (assuming the \( x_i \)'s are still integers).

(c) [7 points] Now describe how to find the actual set of items.

22 Problem 3. Activity Selection

Consider a variation of the Activity Selection problem. As in the original problem, we have \( n \) activities to schedule and each activity has a start time \( s_i \) and a finish time \( f_i \), and we cannot schedule two activities if they overlap. However, now each activity has a weight \( w_i \) which is the profit you get for scheduling activity \( a_i \). Our goal is to find a set \( S \) of non-overlapping activities which have maximum total weight. You may assume all the weights are distinct.

(a) [10 points] Consider the following greedy strategy:

1. sort the activities so \( w_1 > w_2 > \ldots > w_n \).
2. for \( i = 1 \) to \( n \)
   
   Put activity \( a_i \) into \( S \) unless it overlaps an activity already in \( S \).

Give an example (by providing values for \( s_i, f_i, \) and \( w_i \)) where this strategy does not yield an optimal solution.

(b) [12 points] Now suppose each activity has length 1 (so \( f_i = s_i + 1 \)) and the start (and therefore end times) are integers. Formally prove that the algorithm of part (a) finds an optimal solution.
You are given an array of \( n \) numbers where every value except one appears exactly twice; the duplicate values appear next to each other and the remaining value appears only once.

For example:

8 8 2 2 1 1 4 4 5 5 3 6 6 7 7

3 appears only once.

18 18 23 10 10 19 19 17 17 21 21

23 appears only once.

(a) [10 points] Design an efficient divide and conquer algorithm to find which value appears only once. Provide pseudocode.

(b) [5 points] Express the running time of your algorithm as a recurrence \( T(n) \).

(c) [5 points] Solve your recurrence to big-Theta accuracy (\( \Theta(n) \)).