4 Some Probes Are More Equal than Others

A set of n points (for integer $n > 0$) numbered $1, 2, 3, \ldots, n$ are arranged in an array.

You are given n and an operation `probe` to access the points. Given a point number `p`, `probe(p)` returns `p`'s associated value.

However, probing a point `p` has a cost `cost(p)`. (Checking the cost of a point costs nothing, and probing a point more than once also costs nothing.)

The points are sorted in increasing order by their associated values; so, if i and j are point numbers with $i < j$, then `probe(i)` < `probe(j)`.

You would like to **find a target value** t in such a way that you minimize the worst-case total cost of your probes.

1. Here is a modest-sized instance of the problem:

index:	0	1	2	3	4	5	6
value:	2	7	8	13	15	25	90
cost:	20	20	24	10	9	40	10

The minimal worst-case cost of 60 is achievable using a standard binary search approach. **Briefly** explain what target generates the worst-case cost and why.

2. Give at least one trivial and one small instance and their minimal worst-case total probe costs.

3. Give an instance—and its minimal worst-case total probe cost—with 3 points that illustrates that binary search is not an optimal approach in general.

4. Complete this recurrence for the minimum worst-case total probe cost for the subarray `A[i..j]`, i.e., the subarray that includes indexes $i, i+1, \ldots, j-1, j$:

$$C(i,j) = \begin{cases} \min_{i \le k \le j}(& \text{when } j \ge i \\ 0 & \text{otherwise} \end{cases}$$

next probe point
subprobs
$\max(C(i,k-1), C(k+1,j)) + cost(k)$

$j < i \Rightarrow$ Delta
$j = i \Rightarrow$ Celt

5. Write pseudocode for a memoized version of this algorithm. Be sure to indicate clearly the size of table you will need and any initialization the table requires.

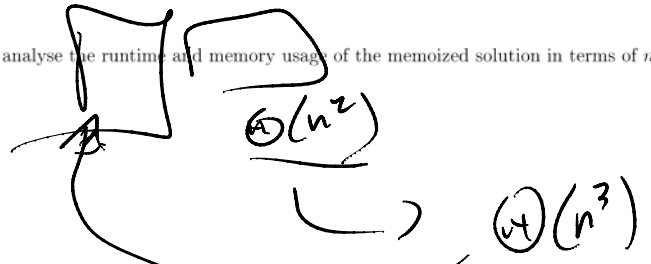
```
C(costs, n):
    initialize table        (how?)
    C_helper(costs, 1, n, table)

C_helper(costs, i, j, table):
    if j < i:
        return 0
    else:
        if table does not contain (i,j):
            COMPUTE (i,j) & PUT IT IN TABLE
        return table[(i,j)]
```

6. Write pseudocode for a dynamic programming version of this algorithm. Be sure to indicate clearly the size of table you will need and the order in which you will solve subproblems.

```
for length = 1 to n:
    for i = 1 to [n - length + mumble]:
        j = i + length - 1
        table[(i,j)] = COMPUTE C(i,j)
```

7. Asymptotically analyse the runtime and memory usage of the memoized solution in terms of n.



8. Assuming you **only** want to know the minimum worst-case cost of solving the problem (not the actual sequence of probes to make), asymptotically analyse the memory usage of an efficient dynamic programming solution.