7. Astro Void se dedica a la compra de asteroides. Sea $p \in \mathbb{N}^n$ tal que p_i es el precio de un asteroide el i-ésimo día en una secuencia de n días. Astro Void quiere comprar y vender asteroides durante esos n días de manera tal de obtener la mayor ganancia neta posible. Debido a las dificultades que existen en el transporte y almacenamiento de asteroides, Astro Void puede comprar a lo sumo un asteroide cada día, puede vender a lo sumo un asteroide cada día y comienza sin asteroides. Además, el Ente Regulador Asteroidal impide que Astro Void venda un asteroide que no haya comprado. Queremos encontrar la máxima ganancia neta que puede obtener Astro Void respetando las restricciones indicadas. Por ejemplo, si $p = (3, 2, 5, 6)$ el resultado es 6 y si $p = (3, 6, 10)$ el resultado es 7. Notar que en una solución óptima, Astro Void debe terminar sin asteroides.

---

a) Convencerse de que la máxima ganancia neta (m.g.n.), si Astro Void tiene c asteroides al fin del día j, es:

- indefinido (i.e., $-\infty$) si $c < 0$ o $c > j$, o
- el máximo entre:
  - la m.g.n. de finalizar el día $j-1$ con $c-1$ asteroides y comprar uno en el día j,
  - la m.g.n. de finalizar el día $j-1$ con $c+1$ asteroides y vender uno en el día j,
  - la m.g.n. de finalizar el día $j-1$ con c asteroides y no operar el día j.

✓

---

b) Escribir matemáticamente la formulación recursiva enunciada en *a*). Dar los valores de los casos base en función de la restricción de que comienza sin asteroides.

Sea $p = \{p_1 ... p_n\}$

$$\text{mgn}_p(j, c) = \begin{cases} -\infty & \text{si } c < 0 \vee c > j \\ 0 & \text{si } j = 0 \wedge c = 0 \\ \max \begin{cases} \text{mgn}_p(j-1, c+1) + p_j \\ \text{mgn}_p(j-1, c-1) - p_j \\ \text{mgn}_p(j-1, c) \end{cases} & \text{sino} \end{cases}$$

---

c) Indicar qué dato es la respuesta al problema con esa formulación recursiva.

Se resuelve con $\text{mgn}_p(n, 0)$

---

d) Diseñar un algoritmo de PD *top-down* que resuelva el problema y explicar su complejidad temporal y espacial auxiliar.

```
f solve(p):
    memo = {}
    n = |p|

    f bt(j,c):
        si c < 0 ó c > j:
            ret -inf
        si j = 0 y c = 0:
            ret 0
        si memo[j][c] existe:
```

```
            ret memo[j][c]
        sino:
            memo[j][c] = max(bt(j-1,c+1)+p[j],bt(j-1,c-1)-p[j],bt(j-1,c))
            return memo[j][c]

    bt(n,0)
```

La complejidad espacial es tal que $0 \leq j \leq n, 0 \leq c \leq n \Rightarrow O(n^2)$

La complejidad temporal está limitada por la espacial, cada llamada recursiva es $O(1)$ y se hace máximo n^2 veces.

---

e) (Opcional) Diseñar un algoritmo de PD *bottom-up*, reduciendo la complejidad espacial.

```
f solve(p):
    n = |p|
    old = actual = [-inf]*n
    old[0] = 0

    para cada j en [1...n]:
        para cada c en [0...j]:
            actual[c] = max(old[c], old[c-1]-p[j-1], old[c+1]+p[j-1])

        para cada c en [j+1...n]:
            actual[c] = -inf

        old, actual = actual, old

    ret old[0]
```

La complejidad temporal es $O(n^2)$, trivial, hago n loops de 2 loops sumados de $0...j$ y $j+1...n$

La complejidad espacial, como guardamos solo el nivel anterior y el actual, es $2 \cdot n \in O(n)$