Greedy → Maximising Profit / Minimising Loss

Pair of shoes → 10 K, 8 K } multiple factors

Q → N items

WiseJoy Global

A[i] → Expiry of i^{th} item

B[i] → Profit margin of i^{th} item

Sell items s.t sum of profit is maximised.

Time starts from T=0 & it takes 1 unit of time to sell 1 item. Item can only be sold if T < A[i].

```
        0  1  2  3  4
A = [ 3  1  3  2  3 ]
B = [ 6  5  3  1  9 ]
```

	index	profit
T = 0	4	9
1	0	6
2	2	3
		18

	index	profit
T = 0	1	5
1	0	6
2	4	9
		20 ✓

```
        0  1
A = [ 1  2 ]
B = [ 3  1500 ]
```

T → 0 1

Ans = 1503

Max Profit → sell all items ←

⇒ order of selling → ascending order of expiry

```
        0  1  2  3  4  5  6  7
A = [ 1  3  3  3  5  5  5  8 ]
B = [ 5  2  7  1  4  3  8  1 ]
```

Correcting an incorrect step from past

T → 0 1 2 3 4 5

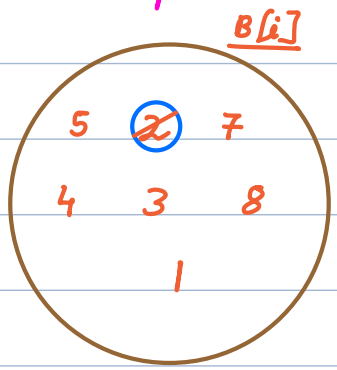
if profit is too less → ignore

total = 28

identify smallest profit ⇒ min heap

	0	1	2	3	4	5	6	7
A =	1	3	3	3	5	5	5	8
B =	5	2	7	1	4	3	8	1

T → 0 1 2 3 4 5



TC = O(N log (N))

SC = O(N)

```
// sort A & B wrt A[i] in ascending order
T = 0                    // min heap → h
sum = 0
for i → 0 to (N-1) {
    if (T < A[i]) {      // not expired
        sum += B[i]
        h.insert(B[i])
        T++
    } else if (B[i] > h.root()) {
        sum -= h.getMin()  // remove root
        sum += B[i]
        h.insert(B[i])
    }
} return sum
```

Q → Given N students with their marks in A[].
Teacher wants to give candies s.t.
a) Every student should have atleast 1 candy
b) Student with more marks have more candies wrt the neighbours (on left & right)
Find min candies to distribute.

```
      0  1  2  3
A = [ 1  5  2  1 ]
C →   1  3  2  1      Ans = 7
```

```
      0  1  2  3  4
A = [ 4  4  4  4  4 ]
C →   1  1  1  1  1      Ans = 5
```

```
      0  1  2  3  4  5  6  7  8
A = [ 1  6  3  1  10 12 20 5  2 ]
C →   1  ✗  ✗  1  ✗  ✗  ✗  ✗  1
L→R      ✗  2     2  3  4  2     L←R
         3
```

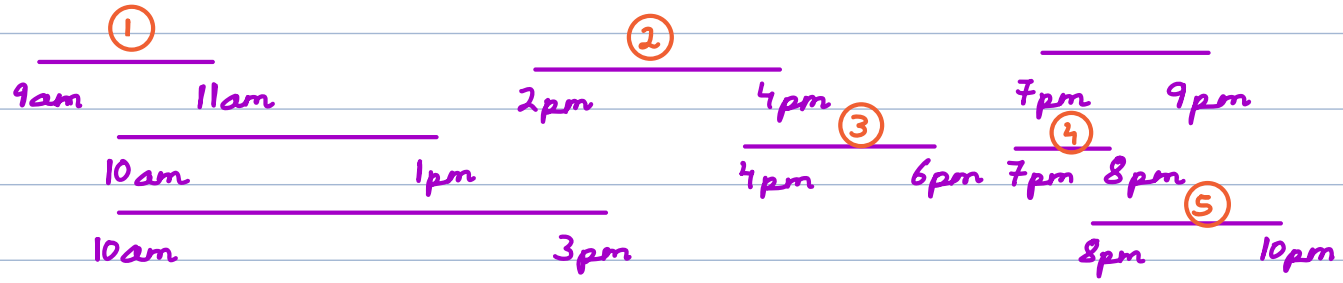
1) $\forall i$, $c[i] = 1$ ✓
2) left → if $(A[i] > A[i-1]) \Rightarrow c[i] > c[i-1]$ ✓ → $c[i] = c[i-1] + 1$
   right → if $(A[i] > A[i+1]) \Rightarrow c[i] > c[i+1]$ → $c[i] = \max(c[i], c[i+1] + 1)$

```
∀i, c[i] = 1
for i → 1 to (N-1) {
    if (A[i] > A[i-1])
        c[i] = c[i-1] + 1
}
Sum = C[N-1]
```

```
for i → (N-2) to 0 {
    if (A[i] > A[i+1])
        c[i] = max(c[i], c[i+1]+1)
    sum += c[i]
}
return sum
```

TC = O(N) SC = O(N)

Q → Given N jobs with start & end time. Find max jobs that can be completed if only 1 job can be done at a time.



Ans = 5

start[K^th] >= End[K-1^th]

	0	1	2	3	4	5
S =	1	5	8	7	12	13
E =	2	10	10	11	20	19

✓ ✓ ✓

Ans = 3

Factors →

1) Start time

X

✗ 1 —— 10

✓ 2 — 5

✓ 5 — 9

Ans = 2

2) Duration

X

✗ 5 — 7 (2)

✓ 1 —— 6 (5)

✓ 6 —— 10 (4)

Ans = 2

3) End time (start + duration) ✓

```
// sort wrt end time
ans = 1          L = E[0]
for i ⟶ 1 to (N-1) {
    if (S[i] >= L) {
        ans++
        L = E[i]
    }
}
return ans
```

TC = O(N log(N))

SC = O(1) / O(N)