Input: list of scores

scores = [8, 4, 2, 1, 3, 6, 7, 9, 5]
4 3 2 1 2 3 4 5 1

Output: minimum number of rewards that you must give out to students

25 // following rewards : [4, 3, 2, 1, 2, 3, 4, 5, 1]

---

① Clarifying questions

- Are we only given integers or something else? Are they positive integers? ⟶ yes, only positive integers
- Can we sort the array? ⟶ no, but the input array could be given to us sorted (order matters)
- Are all scores unique? ⟶ yes, there are no duplicate values

② Naive Solution : Time : $O(n^2)$ ; space : $O(n)$ where n is the # of elements in the input array

- Iterate through the array and
  - for the first number, give it a # ↖ 1
  - If nextNum < currNum give it # and the iterate back to the previous only and give it # +1 ↗ 2

[8, 4 2, 1, 3, 6, 7, 9, 5]
1
2 1
3 2 1
4 3 2 1

- If nextNum > currNum give it # + 1

[8, 4 2, 1, 3, 6, 7, 9, 5]
4 3 2 1 2 3 4 5

★
- When iterating back we assign the prev num based on max ( rewards [j] , rewards [j+1] +1 )

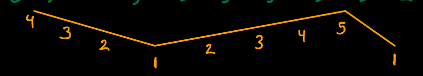
[8, 4 2, 1, 3, 6, 7, $\overset{j}{9}$, 5]
4 3 2 1 2 3 4 5 1 } since max (5, 1)

- Stop iterating when j-1 > j since values before it would already be fixed

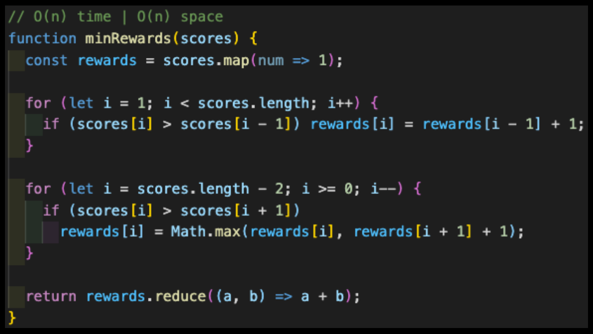
③ How can the solution be better

☆ Peaks and valley's technique:

[8, 4 2, 1, 3, 6, 7, 9, 5]



☆ Iterating forward . Iterating backward



```
// O(n) time | O(n) space
function minRewards(scores) {
  const rewards = scores.map(num => 1);

  for (let i = 1; i < scores.length; i++) {
    if (scores[i] > scores[i - 1]) rewards[i] = rewards[i - 1] + 1;
  }

  for (let i = scores.length - 2; i >= 0; i--) {
    if (scores[i] > scores[i + 1])
      rewards[i] = Math.max(rewards[i], rewards[i + 1] + 1);
  }

  return rewards.reduce((a, b) => a + b);
}
```

Idea:

- Create a rewards array of 1's
- Iterate forward (starting at index 1) and check if the prevNum < currNum. If it is, increment rewards [num] = rewards [prevNum] + 1
- Iterate backward (starting at second last index) and check if currNum > prevNum, if it is the assign currNum the max (currNum, prevNum +1)
- return sum of rewards array

Time: O(n) (where n is the # of elements in the score's array) since we iterate thru forward (n) and backward (n+n = 2n) which is O(n)

Space: O(n) because of the rewards array