LeetCode 561. Array Partition

Question

Given an integer array `nums` of `2n` integers, group these integers into `n` pairs `(a1, b1), (a2, b2), ..., (an, bn)` such that the sum of `min(ai, bi)` for all `i` is **maximized**. Return *the maximized sum*.

Example

Input: nums = [1,4,3,2]
Output: 4
Explanation: All possible pairings (ignoring the ordering of elements) are:

1. (1, 4), (2, 3) → min(1, 4) + min(2, 3) = 1 + 2 = 3
2. (1, 3), (2, 4) → min(1, 3) + min(2, 4) = 1 + 2 = 3
3. (1, 2), (3, 4) → min(1, 2) + min(3, 4) = 1 + 3 = 4
   So the maximum possible sum is 4.

constraints

- `1 <= n <= 104`
- `nums.length == 2 * n`
- `-104 <= nums[i] <= 104`

Definition

- There are 2n length array
- make a pair of array
- get min value from pair
- add two min value and find out maximum
- ! use optimal pairing.
- 최소값들의 합을 최대화하기 위한 최적의 방법 -> Maximize the min

First Code -> 15 min

- ! Failed. Time limit exceeded.

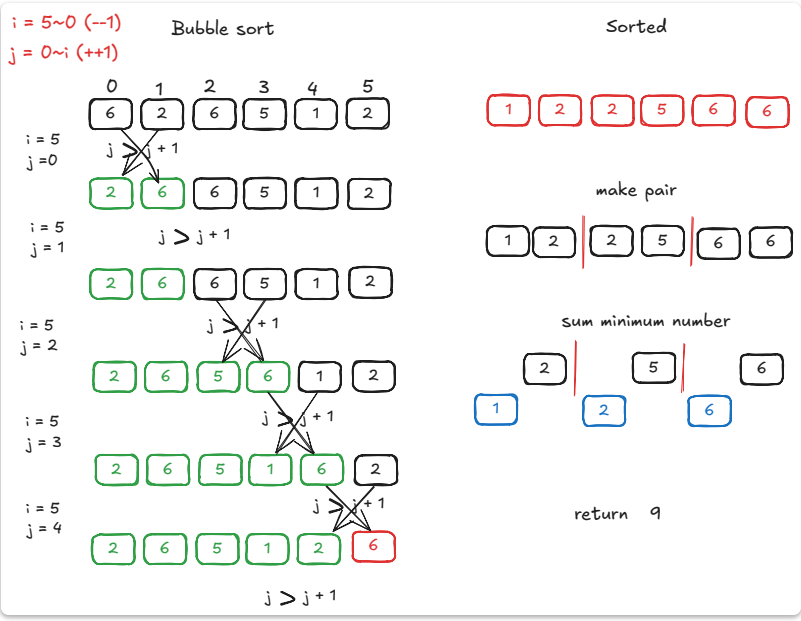
```python
class Solution:

    def arrayPairSum(self, nums: list[int]) -> int:
    # Bubble sort.
        for i in range(len(nums) -1, 0 , -1 ):
            for j in range(i):
                if nums[j] > nums[j+1]:
                    temp = nums[j]
                    nums[i] = nums[j+1]
                    nums[j+1] = temp

    # Make pair
    pairs = []
    for i in range(0,len(nums),2):
        pairs.append(nums[i:i+2])

    # Sum get min
    min_sum = 0
    for i in range(len(pairs)):
        min_sum += pairs[i][0]
    return min_sum
# Time : Time limit exceeded.
```

First Code Explain.



Solution

```python
class Solution:
    def arrayPairSum(self, nums: List[int]) -> int:
        nums.sort()
        sum_ = 0
        for i in range(0,len(nums),2):
            sum_ += nums[i]
        return sum_

class Solution:
    def arrayPairSum(self, nums: List[int]) -> int:
        return sum(sorted(nums)[::2]

# Time : 332 ms
```

Solution Explain

🖊 No need to set another list 'pairs'

Problem is that create pairs aren't good idea.
What We can do is just get even index nums

![example].
[1,2,2,5,6,6]
→ get Even index num = 1,2,6
→ Doesn't need to make other pairs.

🖊 Implemented Sort is More efficient.

Implemented Sort is using 'Merge Sort', "Insertion Sort" Hybrid.

- ! 10,000개의 숫자 처리 시간

버블 정렬	O(n^2)	매우 느림 (100M 연산 이상)	비효율적, 간단한 경우에만 적합
Python 내장 정렬	O(n log n)	매우 빠름 (~100k 연산)	효율적, 대규모 데이터에 적합

- ! Big O

알고리즘	시간 복잡도 (최악)	메모리 사용량	특징
Timsort	O(n log n)	O(n) (병합 단계)	실질적으로 대부분의 데이터에서 빠름.
QuickSort	$O(n^2)$	O(log n)	평균적으로 빠르지만 최악의 경우 느림.
MergeSort	O(n log n)	O(n)	안정적이지만 추가 메모리 사용.
HeapSort	O(n log n)	O(1)	메모리 효율적이지만 상대적으로 느림.