2178. Maximum Split of Positive Even Integers

You are given an integer `finalSum`. Split it into a sum of a **maximum** number of **unique** positive even integers.

- For example, given `finalSum = 12`, the following splits are **valid** (unique positive even integers summing up to `finalSum`): `(12)`, `(2 + 10)`, `(2 + 4 + 6)`, and `(4 + 8)`. Among them, `(2 + 4 + 6)` contains the maximum number of integers. Note that finalSum cannot be split into `(2 + 2 + 4 + 4)` as all the numbers should be unique.

Return *a list of integers that represent a valid split containing a **maximum number** of integers*. If no valid split exists for `finalSum`, return *an **empty** list*. You may return the integers in **any** order.

Example 1:

Input: `finalSum = 12`
Output: `[2,4,6]`
Explanation: The following are valid splits: `(12)`, `(2 + 10)`, `(2 + 4 + 6)`, and `(4 + 8)`.
`(2 + 4 + 6)` has the maximum number of integers, which is 3. Thus, we return `[2,4,6]`.
Note that `[2,6,4]`, `[6,2,4]`, etc. are also accepted.

Example 2:

Input: `finalSum = 7`
Output: `[]`
Explanation: There are no valid splits for the given finalSum.
Thus, we return an empty array.

Example 3:

Input: `finalSum = 28`
Output: `[6,8,2,12]`
Explanation: The following are valid splits: `(2 + 26)`, `(6 + 8 + 2 + 12)`, and `(4 + 24)`.
`(6 + 8 + 2 + 12)` has the maximum number of integers, which is 4. Thus, we return `[6,8,2,12]`.
Note that `[10,2,4,12]`, `[6,2,4,16]`, etc. are also accepted.

Constraints:

- $1 \leq$ `finalSum` $\leq 10^{10}$

Solution

Full Solution

First, let's think of how to determine if a sum S can have a valid split that contains exactly K integers.

The first case we should consider is whether or not a sum S can have a valid split of any size. Since our split includes only even integers, S will only have a split if it's even and we will return an empty list if S is odd.

One observation we can make is that if some sum S does have a valid split of K integers, then a sum T will also have a valid split of K integers if T is even and $T \geq S$. Why is this true? Let's denote D as the difference between T and S. From the split of K integers from the sum S, incrementing the largest integer in the split by D results in a valid split of K integers with a total sum of T. It can be observed that increasing the greatest integer will always keep the entire list distinct.

Example

For this example, let's use the split $12 = 2 + 4 + 6$ with $S = 12$ and $K = 3$. How will we construct a split of size K with sum $T = 16$?

First, we'll find the difference $D = T - S = 4$. Then, we'll add D to the greatest integer in the split with sum S, which is 6.

Thus, we obtain the split $16 = 2 + 4 + 10$ with sum T and size K.

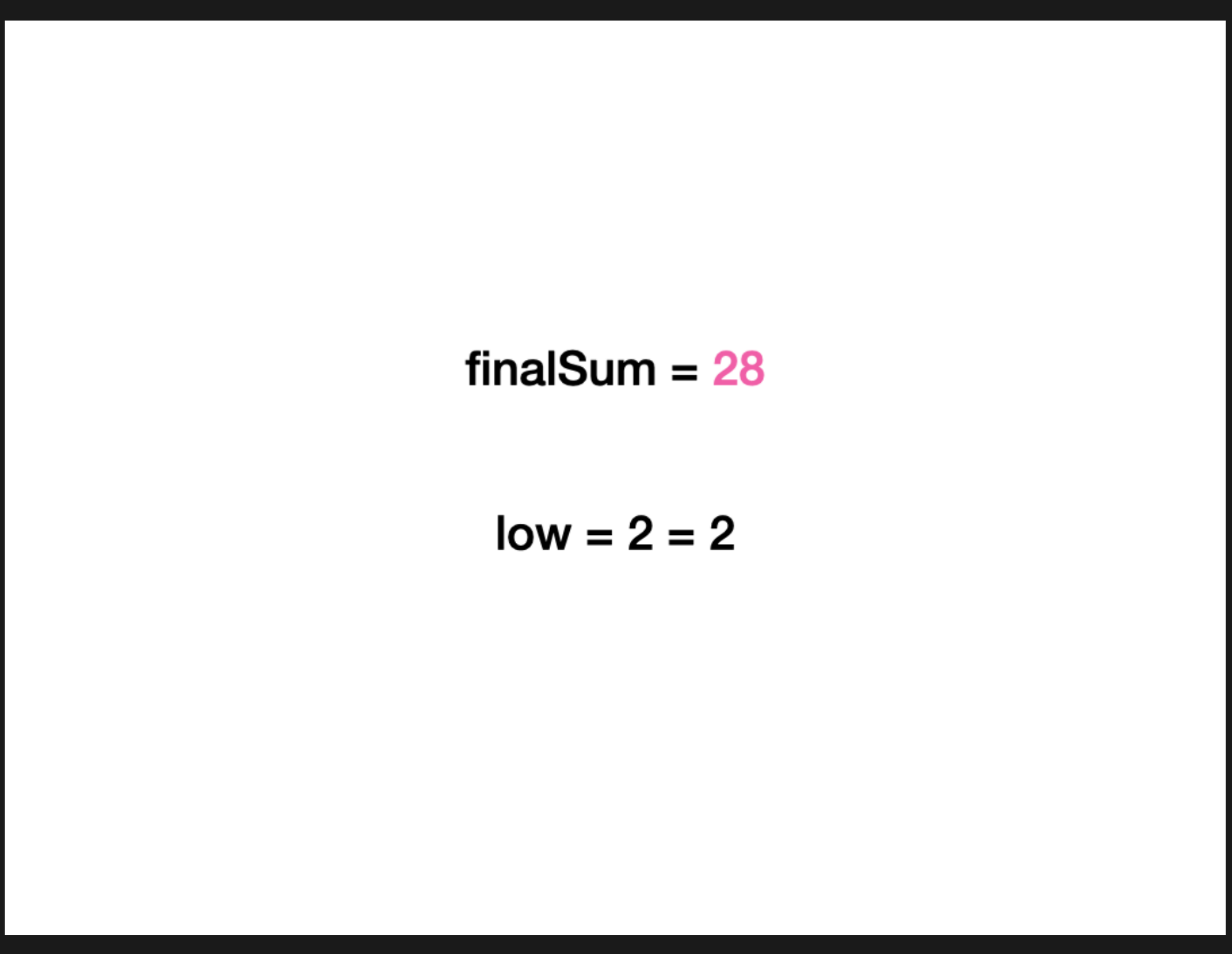
Back to the Original Problem

We are given S and asked to find the **maximum** possible K and construct a split of size K.

If we take the sum of the smallest K positive even integers `(2 + 4 + 6 + 8 + ... + 2*K)`, we'll obtain the **least** possible sum that has a split of size K. Let's denote this sum as `low`. A sum S will have a valid split of size K if $S \geq$ `low`.

To solve the problem, we'll first find the **maximum** K where $S \geq$ `low`. Starting with the split that sums to `low`, we'll add $S -$ `low` to the largest integer to obtain our final split for S.

Simulation



Time Complexity

For a sum S with a split of **maximum** size K, `low = 2 + 4 + 6 + 8 + ... + 2*K`. In the sum, there are $O(K)$ elements and the average element is $O(K)$, resulting in $S = O(K^2)$ and $K = O(\sqrt{S})$. Since our algorithm runs in $O(K)$, our final time complexity is $O(\sqrt{S})$.

Time Complexity: $O(\sqrt{S})$

Space Complexity

Since we construct a list of size K, our space complexity is also $O(\sqrt{S})$.

Space Complexity: $O(\sqrt{S})$

C++ Solution

```cpp
class Solution {
  public:
    vector<long long> maximumEvenSplit(long long finalSum) {
        vector<long long> ans;    // integers in our split
        if (finalSum % 2 == 1) {  // odd sum provides no solution
            return ans;
        }
        long long currentSum = 0;  // keep track of the value of low
        int i = 1;
        while (currentSum + 2 * i <=
               finalSum) {  // keep increasing size of split until maximum
            currentSum += 2 * i;
            ans.push_back(2 * i);
            i++;
        }
        ans[ans.size() - 1] +=
            finalSum - currentSum;  // add S - low to largest element
        return ans;
    }
};
```

Java Solution

```java
class Solution {
    public List<Long> maximumEvenSplit(long finalSum) {
        List<Long> ans = new ArrayList<Long>(); // integers in our split
        if (finalSum % 2 == 1) { // odd sum provides no solution
            return ans;
        }
        long currentSum = 0; // keep track of the value of low
        int i = 1;
        while (currentSum + 2 * i
            <= finalSum) { // keep increasing size of split until maximum
            currentSum += 2 * i;
            ans.add((long) 2 * i);
            i++;
        }
        int idx = ans.size() - 1;
        ans.set(idx,
            ans.get(idx) + finalSum
                - currentSum); // add S - low to largest element
        return ans;
    }
}
```

Python Solution

```python
class Solution:
    def maximumEvenSplit(self, finalSum: int) -> List[int]:
        ans = []  # integers in our split
        if finalSum % 2 == 1:  # odd sum provides no solution
            return ans
        currentSum = 0
        i = 1
        while (
            currentSum + 2 * i <= finalSum
        ):  # keep increasing size of split until maximum
            currentSum += 2 * i
            ans.append(2 * i)
            i += 1
        ans[len(ans) - 1] += finalSum - currentSum  # add S - low to largest element
        return ans
```

Javascript Solution

```javascript
/**
 * @param {number} finalSum
 * @return {number[]}
 */
var maximumEvenSplit = function (finalSum) {
  let ans = []; // integers in our split
  if (finalSum % 2 === 1) {
    // odd sum provides no solution
    return ans;
  }
  let currentSum = 0; // keep track of the value of low
  let i = 1;
  while (currentSum + 2 * i <= finalSum) {
    // keep increasing size of split until maximum
    currentSum += 2 * i;
    ans.push(2 * i);
    i++;
  }
  ans[ans.length - 1] += finalSum - currentSum; // add S - low to largest element
  return ans;
};
```