Intuition: (BackTrack)

The two-pointer technique starts with the widest container and moves the pointers inward based on the comparison of heights.
Increasing the width of the container can only lead to a larger area if the height of the new boundary is greater. By moving the pointers towards the center, we explore containers with the potential for greater areas.

Explanation:

1. The function helper is a recursive function that performs the backtracking. It takes several parameters:
    - start represents the index of the current cookie we are considering.
    - cookies is a constant reference to the vector of cookies.
    - temp is a reference to the vector that stores the current distribution of cookies among the children.
    - k represents the number of children.
    - ans is a reference to the variable that stores the minimum difference between the maximum and minimum number of cookies among the children.
2. The function starts by checking if we have processed all the cookies (base case). If start is equal to the size of the cookies vector, it means we have considered all cookies, and we can calculate the maximum number of cookies among the children. We iterate over the temp vector and update maxi with the maximum value.
3. After obtaining the maximum value, we update ans with the minimum between its current value and maxi. This ensures that we keep track of the minimum difference between the maximum and minimum number of cookies among the children.
4. Now comes the backtracking part. We iterate over the number of children, k, and perform the following steps:
    - Add the current cookie at index start to the ith child by incrementing temp[i] with cookies[start].
    - Recursively call the helper function with the updated start index and the modified temp vector.
    - After the recursive call, we backtrack by subtracting the current cookie from the ith child. This is done by decrementing temp[i] with cookies[start].
5. Finally, the distributeCookies function initializes ans with a maximum value and creates a temporary vector temp of size k initialized with zeros. It then calls the helper function with the starting index 0, the cookies vector, the temp vector, k, and the ans variable.
6. Once the helper function completes, the distributeCookies function returns the calculated ans, which represents the minimum difference between the maximum and minimum number of cookies among the children.

Crucial step:

1. temp[i] += cookies[start];: This line adds the start-th cookie to the i-th child's count. It updates the distribution temporarily for the current recursive call. This step represents "taking" the cookie and assigning it to a specific child.
2. helper(start + 1, cookies, temp, k, ans);: After adding the current cookie to a child, we make a recursive call to the helper function, incrementing the start index by 1. This means we move on to the next cookie and continue distributing it among the children. The recursive call explores all possible distributions of cookies by incrementally considering the next cookies.
3. temp[i] -= cookies[start];: This line performs backtracking. After the recursive call returns, we need to undo the changes made in step 1 to explore other possibilities.
   Subtracting cookies[start] from temp[i] reverts the distribution of the cookie we added to the i-th child. This step represents "back-tracking" or "undoing" the previous distribution.

The purpose of backtracking is to explore all possible combinations of cookie distributions among the children. By adding a cookie to a child, recursively exploring further distributions, and then backtracking by removing the cookie, we can systematically try out different scenarios without explicitly generating all permutations.

The backtracking step is crucial because it allows us to backtrack to a previous state and try a different choice. In this case, it enables us to consider different distributions of cookies among the children by undoing the changes made during the recursive call and trying out other possibilities.