## 1405. Sticks

*The Law of the Jungle is very clear: every wolf, once he has started a family, may leave his Pack. But as soon as his cubs grow up and become strong enough, he must bring them to the Council of the Pack, which is usually held once a month during the full moon, and present them to all the other wolves.*

Father Wolf waited until his cubs had grown a little and started to run about. Then, on one of the nights when the Pack gathered, he led them – together with Mowgli and Mother Wolf – to the Council Rock. It was the top of a hill strewn with large boulders, behind which a hundred wolves could easily hide. Akela, the great gray lone wolf, chosen as leader of the whole Pack for his strength and agility, called out from his rock:

— The Law is known to you, the Law is known to you! Look well, wolves!

Father Wolf pushed the Frog, Mowgli, into the center of the circle. Mowgli sat down on the ground, laughed, and began to play with some sticks.

He came up with a simple task: to make a rectangle of the maximum possible area from these sticks – not necessarily using all of them.

**Input.** The first line contains the number of sticks *n* (1 ≤ *n* ≤ 16). The second line contains their lengths – positive integers in the range from 1 to 108.

**Output.** Print the maximum possible area of a rectangle that can be made from the given set of sticks, or  if forming a rectangle is impossible.

|  |  |
| --- | --- |
| **Sample input** | **Sample output** |
| 87 1 5 2 3 2 4 5 | 49 |

**SOLUTION**

**dynamic programming - masks**

# Algorithm analysis

Let S be the given set of sticks. It is required to find two disjoint subsets A and B of S such that the sticks in each of them can be divided into two subsets with equal total lengths.

For every subset *mask* ⊆ S, compute the total length of its sticks and store it in sum[*mask*].

Iterate over all subsets I  S. For each subset I, go through all of its subsets J  I. If there exists such a subset J that 2 \* sum[J] = sum[I] (that is, the total length of sticks in J is exactly half of the total length of sticks in I), then the subset I can be used as a pair of opposite sides of a rectangle. In this case, set can[I] = ***true***.

The entire procedure runs in O(3*n*).

Iterate over all subsets of the set of sticks. If for some subset I there exists a subset J  I such that can[J] = ***true*** and can[I xor J] = ***true*** (that is, the subsets J and I xor J are disjoint), then we obtain one of the possible distributions of sticks between the horizontal and vertical sides of the rectangle.

The side lengths of such a rectangle are sum[J] / 2 and sum[I xor J] / 2. Among all such possible rectangles, choose the one with the maximum area.

# Algorithm implementation

The lengths of the sticks are stored in the array *a*. For each subset of sticks *mask*, store their total length in the array sum[*mask*]. The value can[*mask*] is set to ***true*** if the subset *mask* can be divided into two parts with equal total lengths – these will correspond to the opposite sides of a rectangle.

#define MAX 16

int a[MAX];

int sum[1 << MAX];

bool can[1 << MAX];

The main part of the program. Read the input data.

scanf("%d",&n);

for (i = 0; i < n; i++)

 scanf("%d", &a[i]);

Iterate over all subsets of sticks and compute the total length of each subset.

for (i = 0; i < 1 << n; i++)

 for (j = 0; j < n; j++)

 if (i & (1 << j)) sum[i] += a[j];

Then, iterate through all possible subsets of the set of sticks. If for some subset I there exists a subset J  I such that the total length of sticks in J is exactly half of the total length of sticks in I, set can[I] = ***true***. In this case, the sticks from subset I can be used as opposite sides of a rectangle.

for (i = 1; i < 1 << n; i++)

{

 int s = sum[i];

 for (j = i; j > 0; j = (j - 1) & i)

 if (sum[j] \* 2 == s)

 {

 can[i] = true;

 break;

 }

}

Iterate over all subsets of the set of sticks. If for some subset I there exists a subset J  I such that can[J] = ***true*** and can[I xor J] = ***true***, then we obtain one of the possible distributions of sticks between the horizontal and vertical sides of the rectangle. The side lengths of such a rectangle are sum[J] / 2 and sum[I xor J] / 2.

res = 0;

for (i = 1; i < 1 << n; i++)

 for (j = i; j > 0; j = (j - 1) & i)

 if (can[j] && can[i ^ j])

 res = max(res, sum[j] / 2 \* 1LL \* sum[i ^ j] / 2);

Print the area of the rectangle with the maximum size found.

printf("%lld\n",res);