Cutting the Cake

Here we show how the Codility Challenge codenamed Silicium-2014 can be solved. You can still give it a try, but no certificate will be granted. The problem asks you to find the K-th piece of a cake in terms of size.

**Slow solution** $O(N^2 \log N)$

The simplest solution is to calculate the area of each piece of the cake separately. After that, it is sufficient to sort all the areas and to choose $K$-th from them.

```python
# calculate the lengths of the pieces
def calculateLengths(X, Y, A, B):
    N = len(A)
    width = [0] * (N + 1)
    width[0] = A[0]
    for i in xrange(1, N):
    width[N] = X - A[N - 1]
    height = [0] * (N + 1)
    height[0] = B[0]
    for i in xrange(1, N):
        height[i] = B[i] - B[i - 1]
    height[N] = Y - B[N - 1]
    return width, height

# calculate the areas
def slowSolution(X, Y, K, A, B):
    N = len(A) + 1
    width, height = calculateLengths(X, Y, A, B)
    pieces = [0] * (N * N)
    for i in xrange(N):
        for j in xrange(N):
            pieces[i + (j - 1) * N] = width[i] * height[j]
    # sort areas and choose K-th
    pieces.sort()
    return pieces[N * N - K]
```

The time complexity of the above algorithm is $O(N^2 \log N)$ due to the sorting time of all the elements. This approach is far from optimal.
Golden solution $O(N \log(N + X + Y))$

The size of the piece of cake we are looking for can be found by a binary search of its area. The area is between 1 and the size of the biggest piece of the cake. In each iteration of the binary search, the interval is halved. We select the middle element $s$ of the interval and, depending on the number of pieces that are greater than or equal to $s$, choose the left or right interval for the next iteration.

```
2: Golden solution — $O(N \log(N + X + Y))$.
1  def goldenSolution(X, Y, K, A, B):
2      N = len(A)
3      width, height = calculateLengths(X, Y, A, B)
4      width.sort()
5      height.sort()
6      beg = 1
7      end = width[N] * height[N]
8      result = 0
9      # binary search by the area
10     while beg <= end:
11         mid = (beg + end) // 2
12         if greater_eq(X, Y, mid, width, height) >= K:
13             beg = mid + 1
14             result = mid
15         else:
16             end = mid - 1
17         return result
```

As the interval is halved in every iteration, the number of all divisions can be estimated by $O(\log(X + Y))$. All that remains is the question of how to calculate the number of pieces that are greater than or equal to $s$.

Counting pieces of the cake

The widths and heights of the pieces are sorted into non-decreasing order. We calculate the number of pieces starting from the smallest widths. Let’s assume that we know the number of pieces greater than or equal to $s$ for some fixed width. How can this number change for larger width? It can only increase, because all heights stay the same and the width gets larger.

```
3: The number of cakes — $O(N)$. 
1  def greater_eq(X, Y, mid, width, height):
2      N = len(width)
3      result = 0
4      j = N - 1
5      for i in xrange(N):
6          while j >= 0 and width[i] * height[j] >= mid:
7              j -= 1
8          result += N - 1 - j
9      return result
```

The time complexity of the above function is linear due to the amortized cost. The variable $j$ cannot be decreased more than $N$ times, and it is decreased by 1 in every iteration of the while loop.

The time complexity of the whole algorithm is $O(N \log(N + X + Y))$, due to the binary search of the result and sorting all the widths and heights.