



# BACKTRACKING

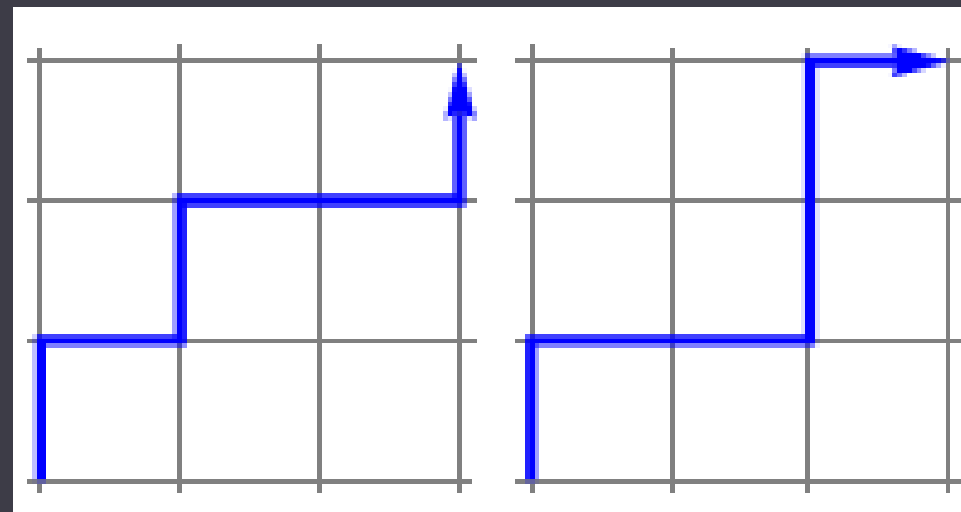
AULA 2 - BUSCAS

# Bruteforce

Muitos problemas requerem encontrar um certo objeto (ou contar/listar múltiplos objetos) que satisfaz uma certa propriedade ou que otimiza um certo valor, seja este objeto um número inteiro, um caminho num grid ou um par de posições num array.

Bruteforce se refere ao ato de testar todos os valores possíveis para o objeto e computar o resultado necessário

$$l \leq x \leq r$$

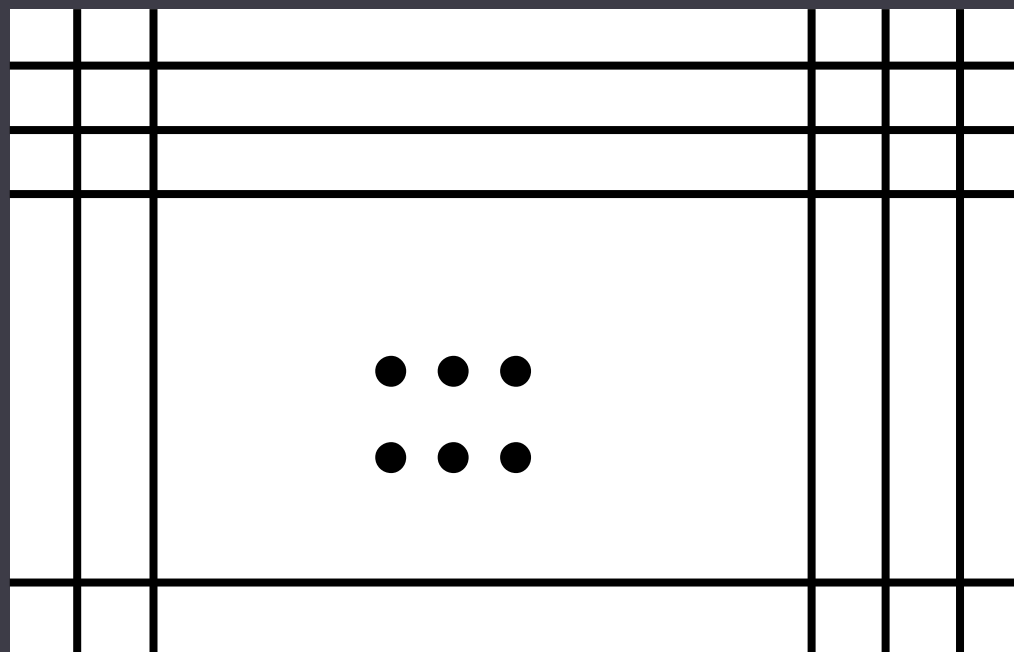


...  $\frac{\quad}{i}$  ...  $\frac{\quad}{j}$  ...

# Bruteforce

## Inteiros num range

$$l \leq x \leq r$$



### J - Cutting into Parts

Gym - 105437A [↗](#)

Monocarp has a rectangular sheet of paper that he wants to divide into exactly  $n$  parts.

To do this, he can make two types of cuts:

- A horizontal cut that starts from the left side of the sheet, ends on the right side of the sheet, passing through the entire sheet parallel to the top and bottom sides of the sheet;
- A vertical cut that starts from the top side of the sheet, ends on the bottom side of the sheet, passing through the entire sheet parallel to the left and right sides of the sheet.

Each type of cut can be made an arbitrary number of times. After each cut, none of the resulting pieces of the sheet are moved or removed; each new cut goes through the entire sheet.

Your task is to determine the minimum total number of horizontal and vertical cuts that Monocarp has to make to divide the sheet into exactly  $n$  parts.

#### Input

The first line contains an integer  $n$  ( $2 \leq n \leq 1\,000\,000$ ) — the number of parts that Monocarp wants to have.

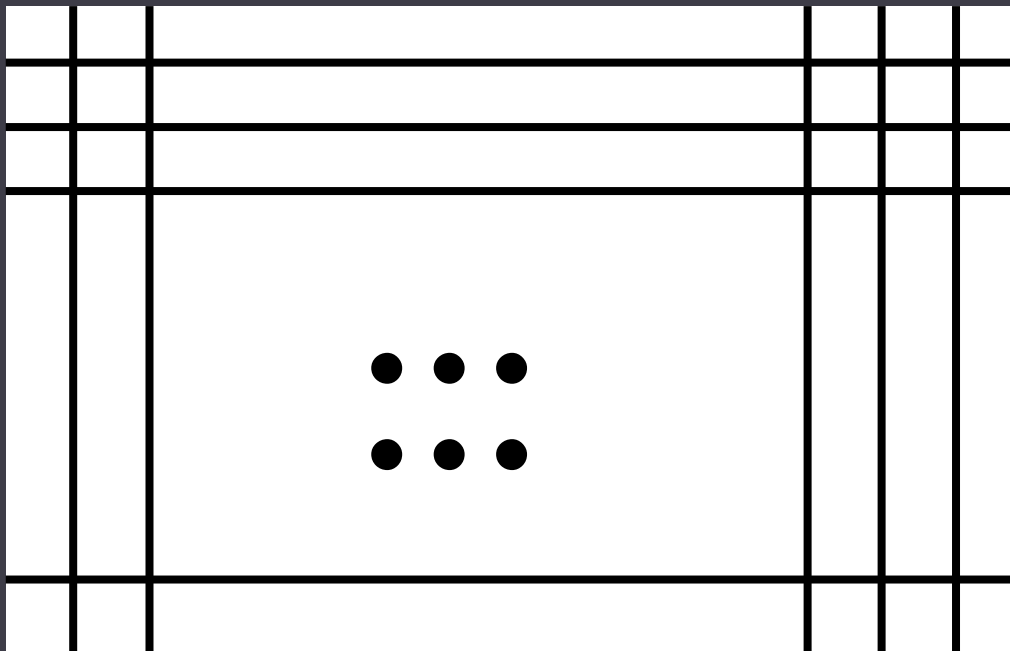
# Bruteforce

Inteiros num range

$$l \leq x \leq r$$

$$n/x$$

x



## J - Cutting into Parts

Gym - 105437A [↗](#)

Monocarp has a rectangular sheet of paper that he wants to divide into exactly  $n$  parts.

To do this, he can make two types of cuts:

- A horizontal cut that starts from the left side of the sheet, ends on the right side of the sheet, passing through the entire sheet parallel to the top and bottom sides of the sheet;
- A vertical cut that starts from the top side of the sheet, ends on the bottom side of the sheet, passing through the entire sheet parallel to the left and right sides of the sheet.

Each type of cut can be made an arbitrary number of times. After each cut, none of the resulting pieces of the sheet are moved or removed; each new cut goes through the entire sheet.

Your task is to determine the minimum total number of horizontal and vertical cuts that Monocarp has to make to divide the sheet into exactly  $n$  parts.

### Input

The first line contains an integer  $n$  ( $2 \leq n \leq 1\,000\,000$ ) — the number of parts that Monocarp wants to have.

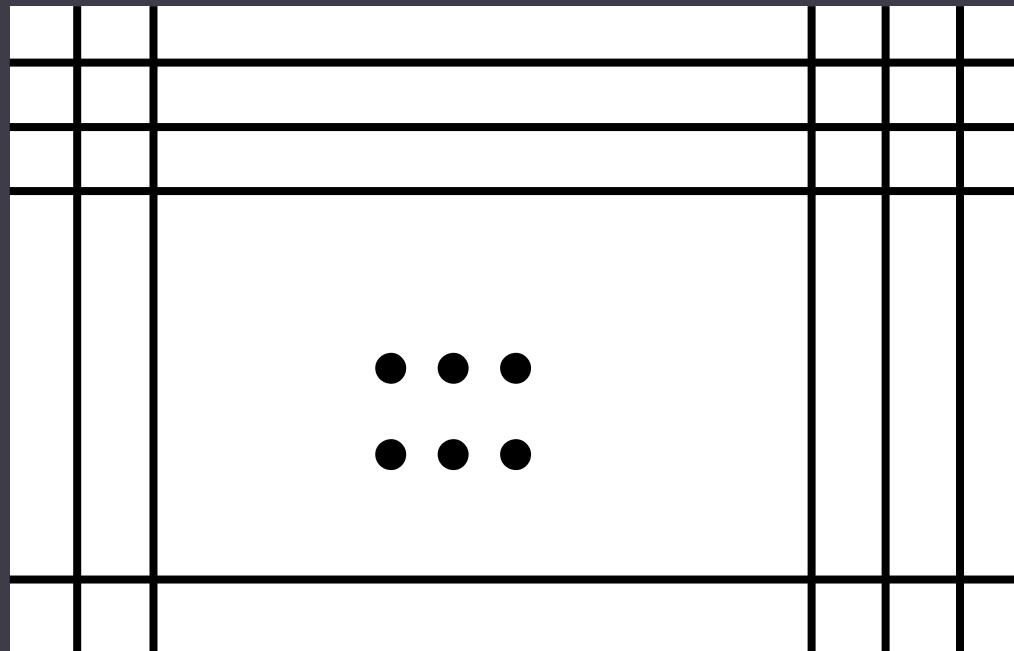
# Bruteforce

Inteiros num range

$$l \leq x \leq r$$

$$n/x$$

x



## J - Cutting into Parts

Gym - 105437A [↗](#)

Monocarp has a rectangular sheet of paper that he wants to divide into exactly  $n$  parts.

To do this, he can make two types of cuts:

- A horizontal cut that starts from the left side of the sheet, ends on the right side of the sheet, passing through the entire sheet parallel to the top and bottom sides of the sheet;
- A vertical cut that starts from the top side of the sheet, ends on the bottom side of the sheet, passing through the entire sheet parallel to the left and right sides of the sheet.

Each type of cut can be made an arbitrary number of times. After each cut, none of the resulting pieces of the sheet are moved or removed; each new cut goes through the entire sheet.

Your task is to determine the minimum total number of horizontal and vertical cuts that Monocarp has to make to divide the sheet into exactly  $n$  parts.

### Input

The first line contains an integer  $n$  ( $2 \leq n \leq 1\,000\,000$ ) — the number of parts that Monocarp wants to have.

# Bruteforce

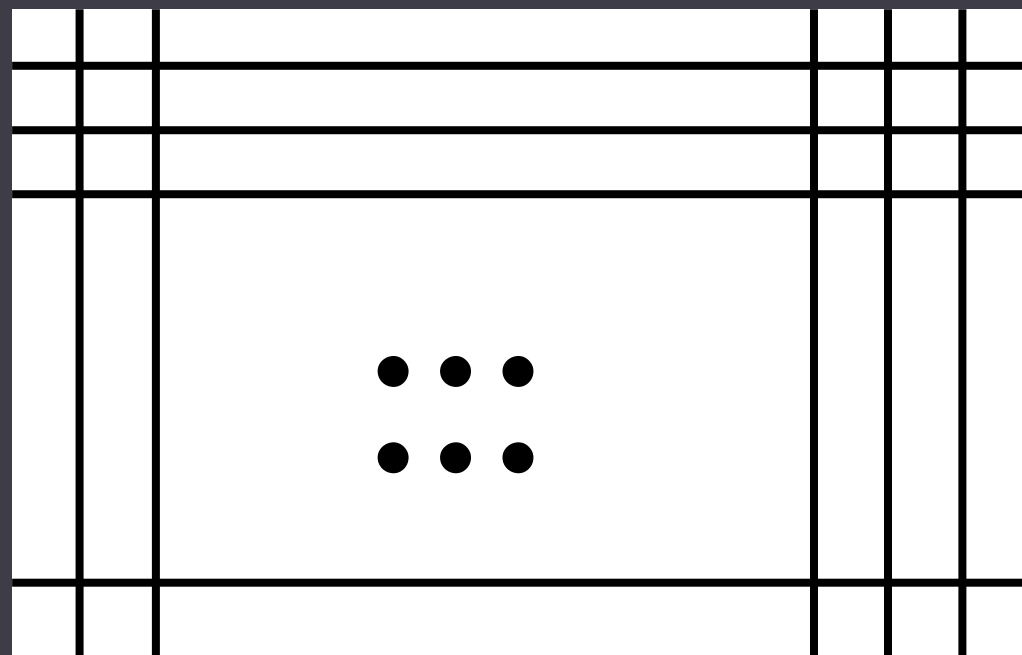
## Observações

- Para estimar complexidade:
  - Contar candidatos
  - Contar tempo de checagem de um candidato

# Bruteforce

Inteiros num range

- Para estimar complexidade:
  - Contar candidatos
  - Contar tempo de checagem de um candidato

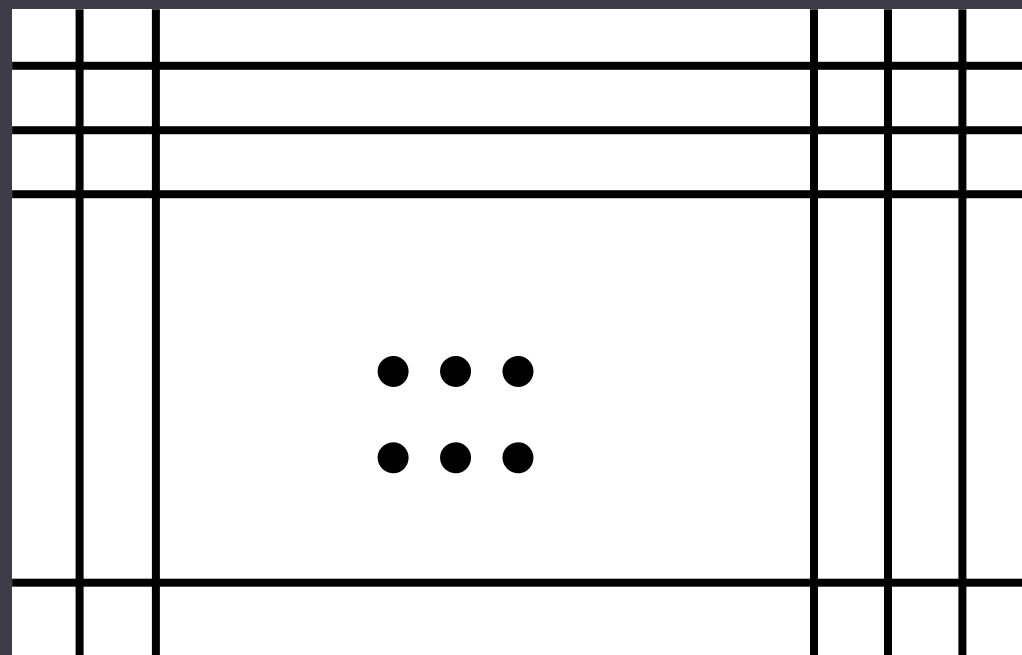


- Candidatos: ?
- Checagem: ?

# Bruteforce

Inteiros num range

- Para estimar complexidade:
  - Contar candidatos
  - Contar tempo de checagem de um candidato



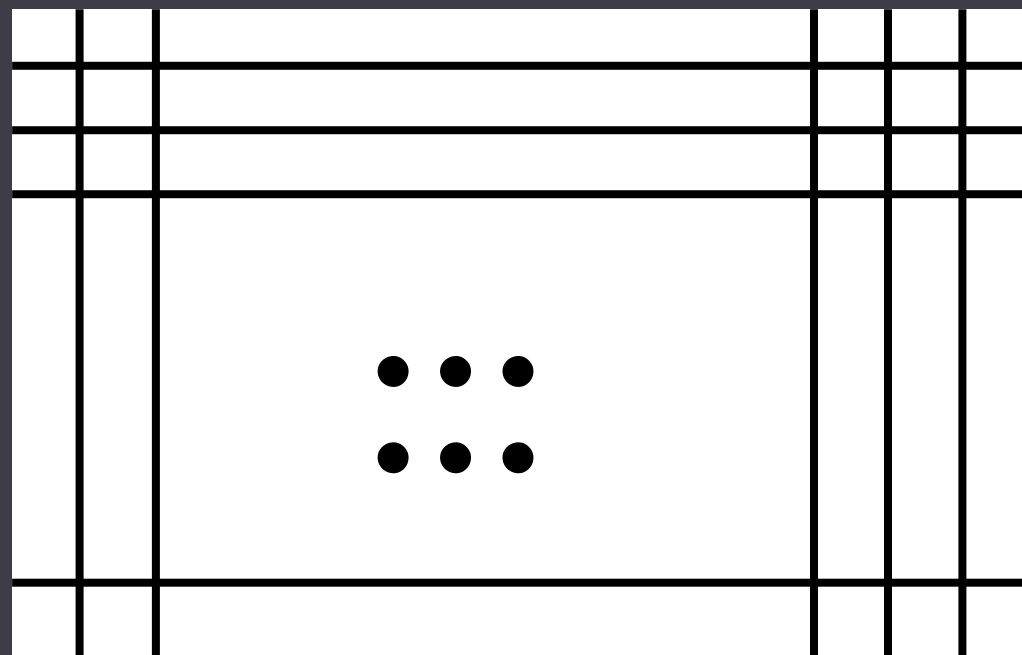
- Candidatos:  $O(n)$
- Checagem:  $O(1)$



# Bruteforce

Inteiros num range

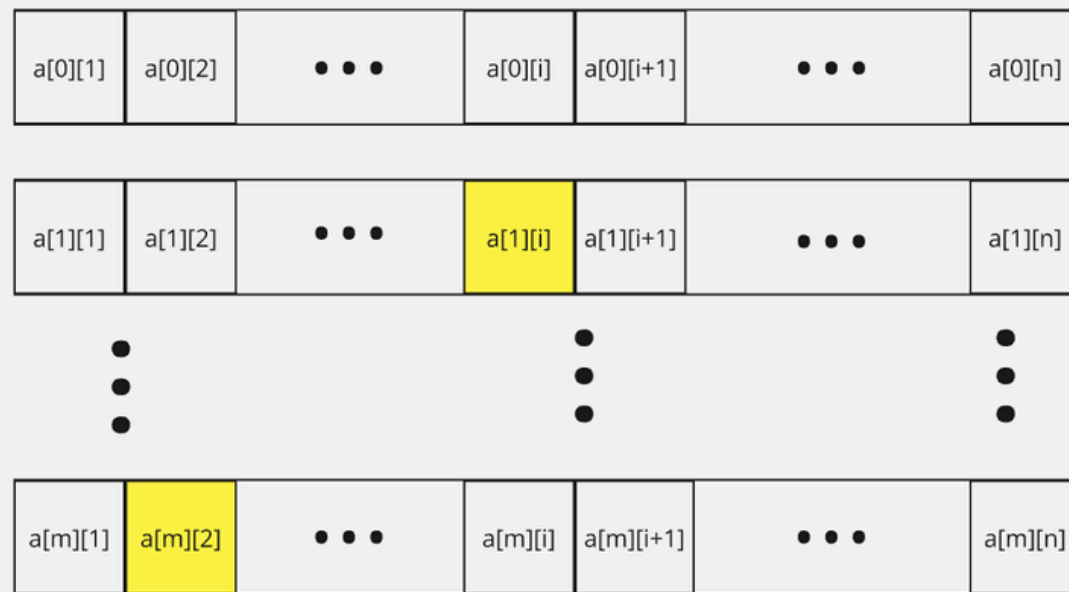
- Para estimar complexidade:
  - Contar candidatos
  - Contar tempo de checagem de um candidato



- Candidatos:  $O(n)$
- Checagem:  $O(1)$

# Bruteforce

## Arrays



- Array de tamanho  $n$
- $m$  modificações sucessivas
- Computar somatório, para todos os pares de arrays no histórico, do número de elementos distintos no par

## F - Serval and Toxel's Arrays [CodeForces - 1789C](#)

Toxel likes arrays. Before traveling to the Paldea region, Serval gave him an array  $a$  as a gift. This array has  $n$  **pairwise distinct** elements.

In order to get more arrays, Toxel performed  $m$  operations with the initial array. In the  $i$ -th operation, he modified the  $p_i$ -th element of the  $(i - 1)$ -th array to  $v_i$ , resulting in the  $i$ -th array (the initial array  $a$  is numbered as 0). During modifications, Toxel guaranteed that the elements of each array are still **pairwise distinct** after each operation.

Finally, Toxel got  $m + 1$  arrays and denoted them as  $A_0 = a, A_1, \dots, A_m$ . For each pair  $(i, j)$  ( $0 \leq i < j \leq m$ ), Toxel defines its value as the number of distinct elements of the concatenation of  $A_i$  and  $A_j$ . Now Toxel wonders, what is the sum of the values of all pairs? Please help him to calculate the answer.

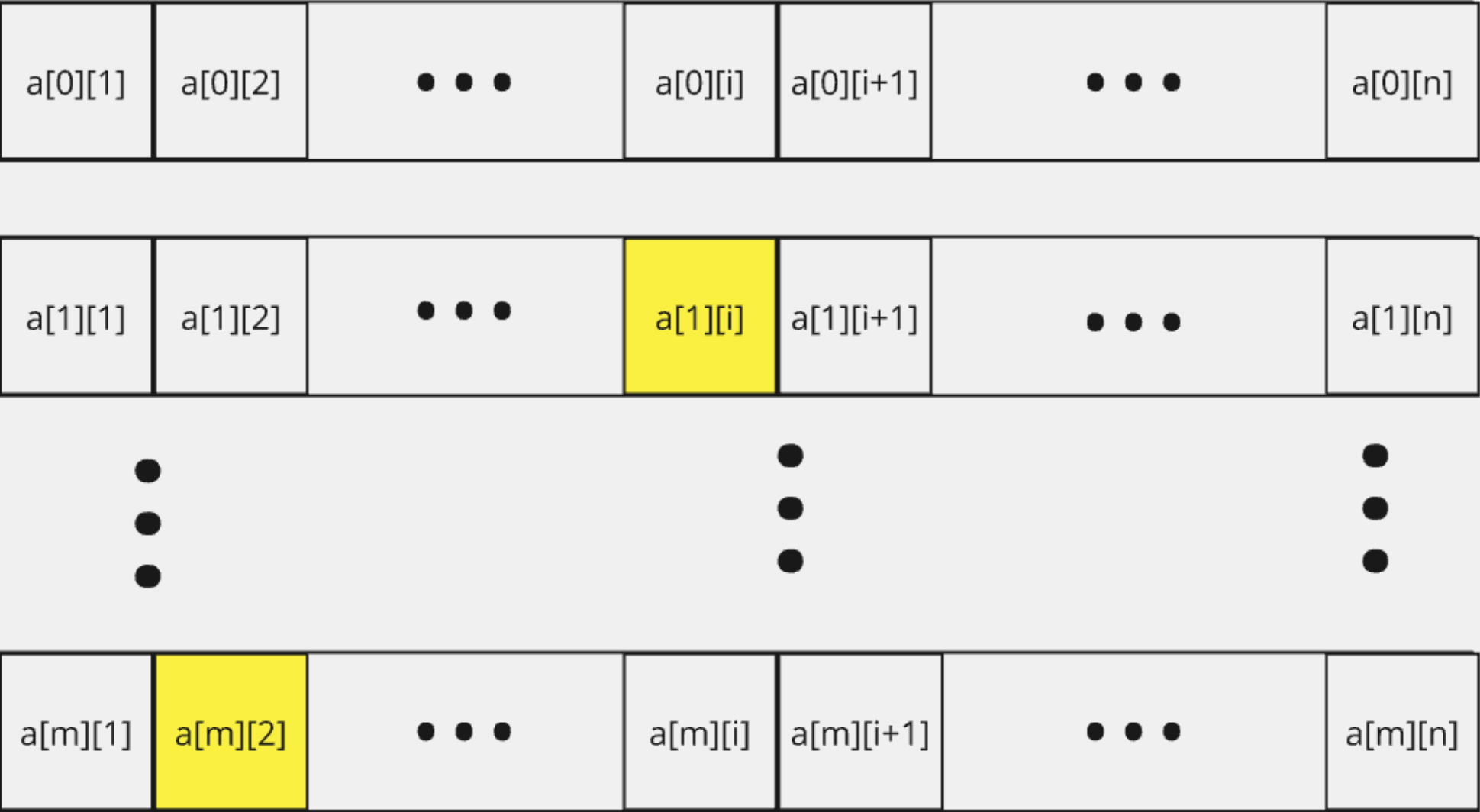
### Input

Each test contains multiple test cases. The first line contains the number of test cases  $t$  ( $1 \leq t \leq 10^4$ ). The description of the test cases follows.

The first line of each test case contains two integers  $n$  and  $m$  ( $1 \leq n, m \leq 2 \cdot 10^5$ ) — the length of the array and the number of operations.

# Bruteforce

## Arrays



- Array de tamanho  $n$
- $m$  modificacoes sucessivas
- Computar somatório, pra todos os pares de arrays no histórico, do número de elementos distintos no par

### F - Serval and Toxel's Arrays [CodeForces - 1789C](#)

Toxel likes arrays. Before traveling to the Paldea region, Serval gave him an array  $a$  as a gift. This array has  $n$  pairwise distinct elements.

In order to get more arrays, Toxel performed  $m$  operations with the initial array. In the  $i$ -th operation, he modified the  $p_i$ -th element of the  $(i - 1)$ -th array to  $v_i$ , resulting in the  $i$ -th array (the initial array  $a$  is numbered as 0). During modifications, Toxel guaranteed that the elements of each array are still pairwise distinct after each operation.

Finally, Toxel got  $m + 1$  arrays and denoted them as  $A_0 = a, A_1, \dots, A_m$ . For each pair  $(i, j)$  ( $0 \leq i < j \leq m$ ), Toxel defines its value as the number of distinct elements of the concatenation of  $A_i$  and  $A_j$ . Now Toxel wonders, what is the sum of the values of all pairs? Please help him to calculate the answer.

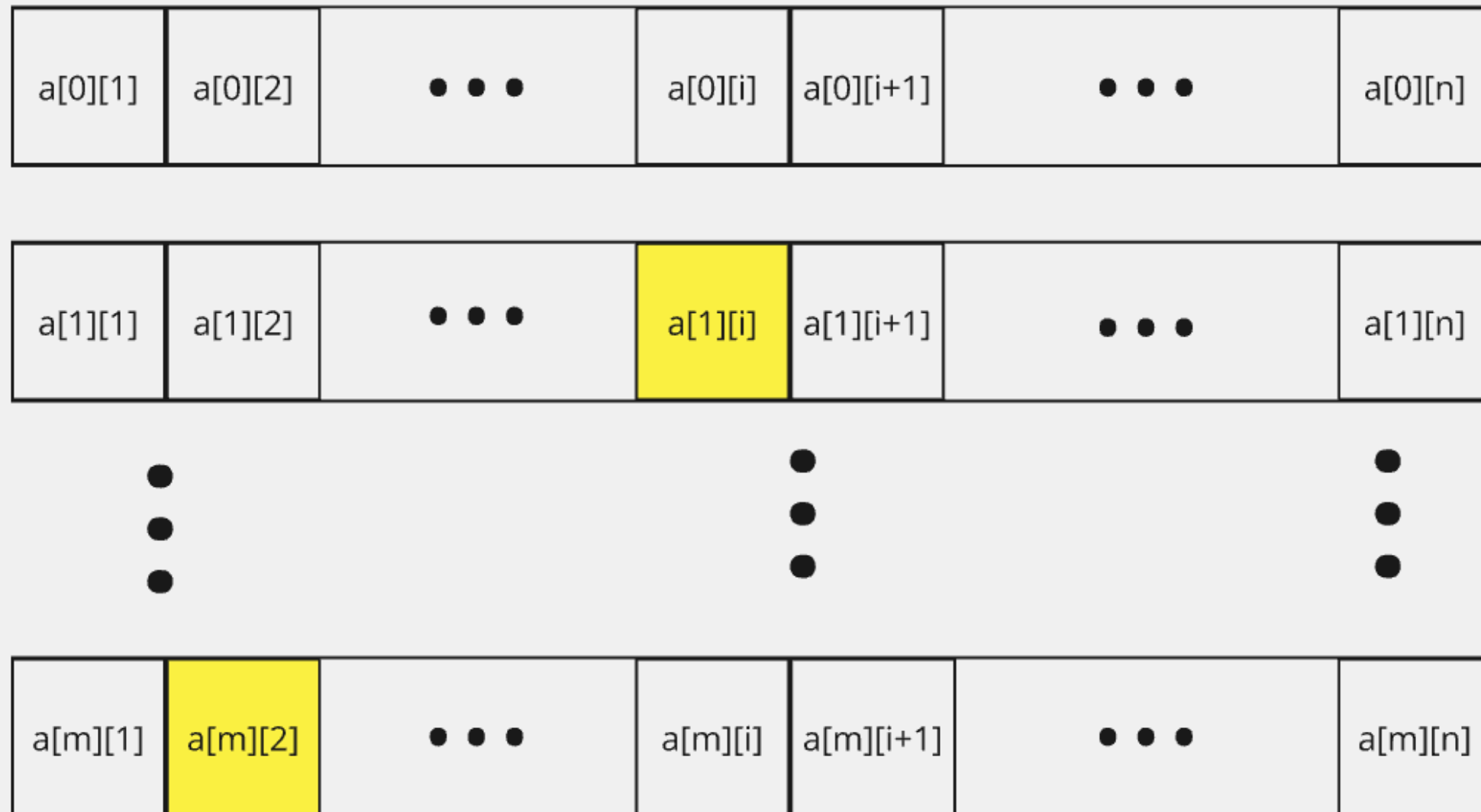
#### Input

Each test contains multiple test cases. The first line contains the number of test cases  $t$  ( $1 \leq t \leq 10^4$ ). The description of the test cases follows.

The first line of each test case contains two integers  $n$  and  $m$  ( $1 \leq n, m \leq 2 \cdot 10^5$ ) — the length of the array and the number of operations.

# Bruteforce

## Arrays

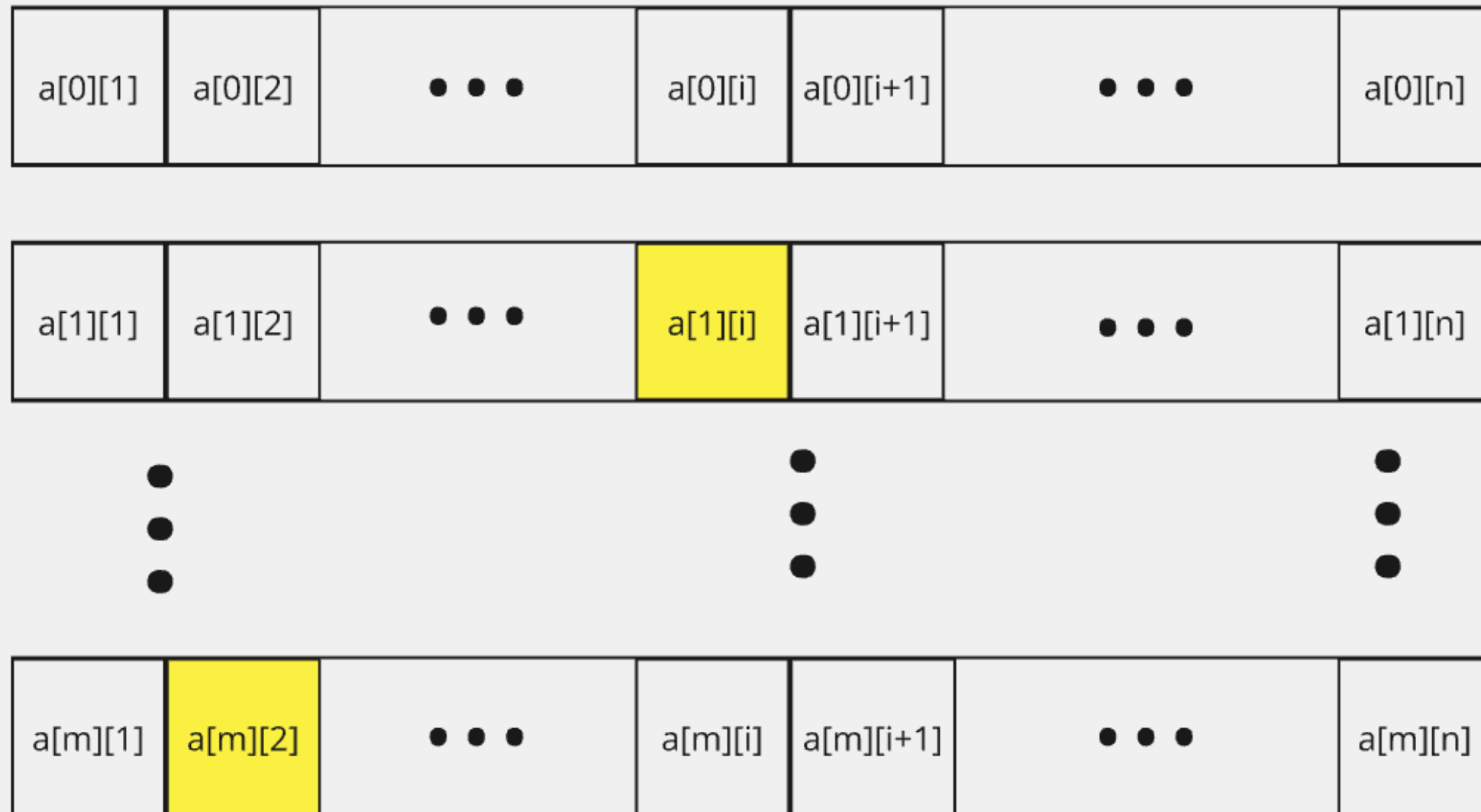


- Array de tamanho  $n$
- $m$  modificacoes sucessivas
- Computar somatório, pra todos os pares de arrays no histórico, do número de elementos distintos no par

Candidatos?

# Bruteforce

## Arrays



- Array de tamanho  $n$
- $m$  modificações sucessivas
- Computar somatório, pra todos os pares de arrays no histórico, do número de elementos distintos no par

Candidatos?

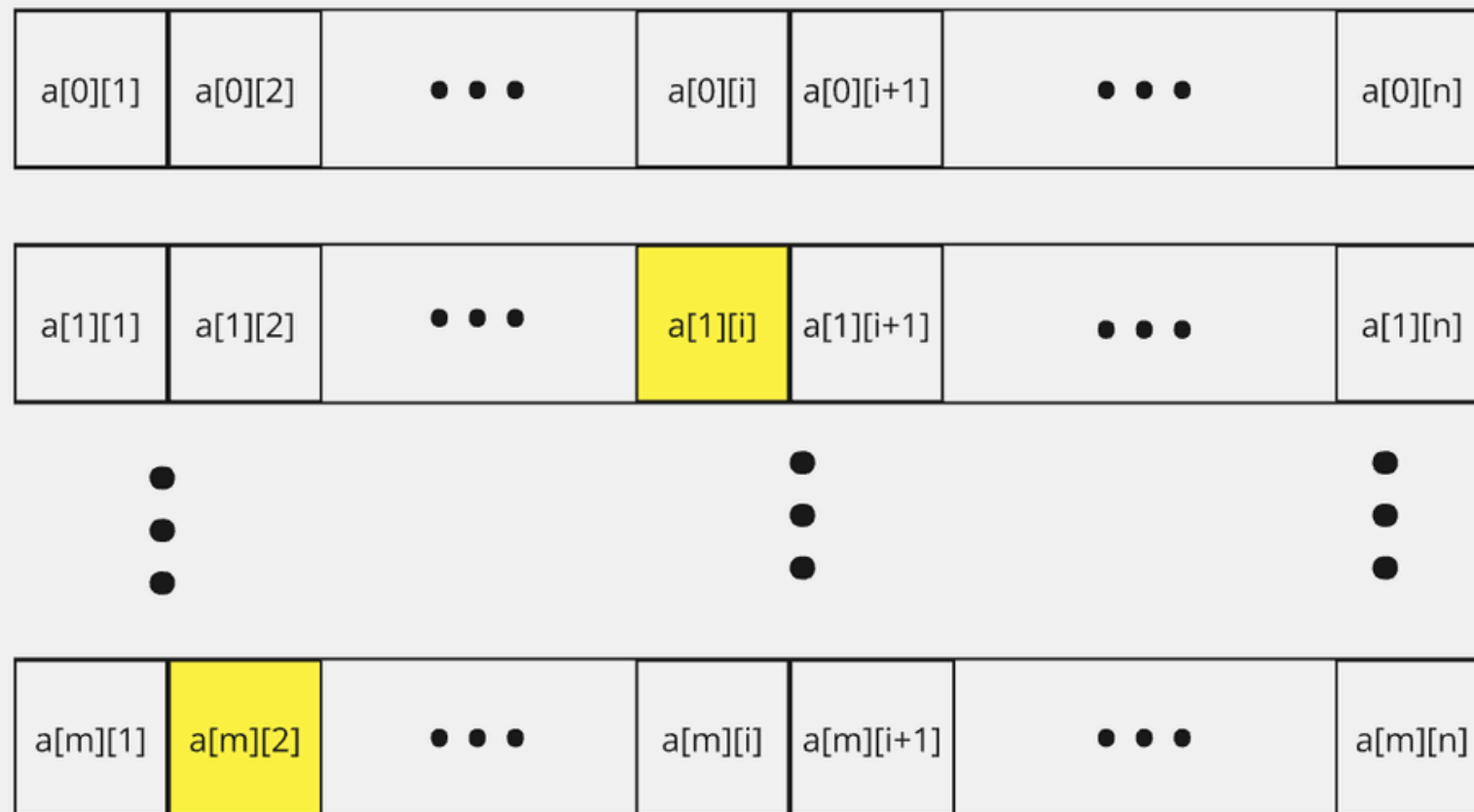
Pares de arrays  
no histórico!

# Bruteforce

## Arrays

Complexidade:

- Candidatos:  $O(m^2)$
- Checagem:  $O(n \log(n))$



# Bruteforce

## Arrays

a[0][1]	a[0][2]	...	a[0][i]	a[0][i+1]	...	a[0][n]
a[1][1]	a[1][2]	...	a[1][i]	a[1][i+1]	...	a[1][n]
⋮			⋮			⋮
a[m][1]	a[m][2]	...	a[m][i]	a[m][i+1]	...	a[m][n]

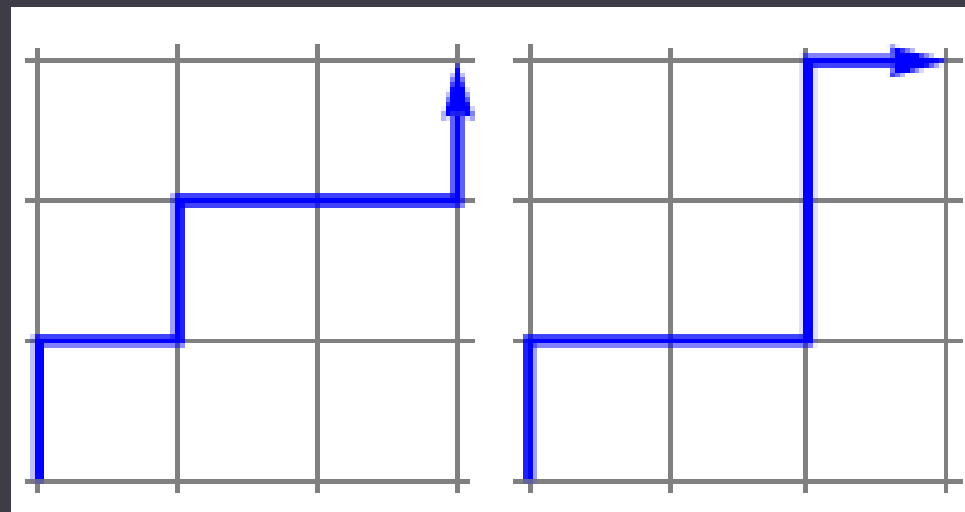
Complexidade:

- Candidatos:  $O(m^2)$
- Checagem:  $O(n \log(n))$
- No total  $O(m^2 n \log(n))$ 
  - $\simeq 1e16 = \text{IMPOSSÍVEL}$
  - Se a constraint fosse  $n, m \leq 100$  seria viável!

# Backtracking

O backtracking é um tipo de bruteforce, no qual se pode descartar um candidato mesmo que ele não esteja completamente montado, e costuma-se construir os candidatos incrementalmente

$$l \leq x \leq r$$



...  $\frac{\quad}{i}$  ...  $\frac{\quad}{j}$  ...



# Backtracking

## Sudoku

Você recebe um tabuleiro de sudoku parcialmente preenchido. Responda se é possível completá-lo.

				4				
				8				
		5	7		3	4		
		4		2		5		
9	2		1		4		8	7
		1		3		2		
		8	4		6	9		
				1				
				9				

# Backtracking

Sudoku

Candidato?

				4				
				8				
		5	7		3	4		
		4		2		5		
9	2		1		4		8	7
		1		3		2		
		8	4		6	9		
				1				
				9				

# Backtracking

## Sudoku

Candidato?

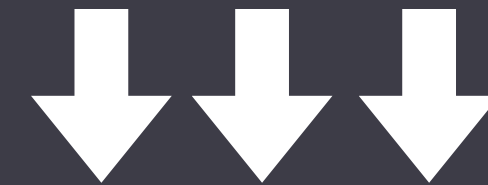
				4				
				8				
		5	7		3	4		
		4		2		5		
9	2		1		4		8	7
		1		3		2		
		8	4		6	9		
				1				
				9				

8	1	7	9	4	5	3	2	6
3	4	6	2	8	1	7	9	5
2	9	5	7	6	3	4	1	8
7	8	4	6	2	9	5	3	1
9	2	3	1	5	4	6	8	7
5	6	1	8	3	7	2	4	9
1	3	8	4	7	6	9	5	2
4	7	9	5	1	2	8	6	3
6	5	2	3	9	8	1	7	4

# Backtracking

Sudoku

Construção incremental:



1	2	9			
8	5	3			
7	4	6			
2	6	8			
5	9	4			
3	7	1			

# Backtracking

## Sudoku

Construção incremental:

- Passo: casa do tabuleiro
- Opções: números de 1 a 9
- Complexidade: ?



1	2	9	●		
8	5	3			
7	4	6			
2	6	8			
5	9	4			
3	7	1			

# Backtracking

## Sudoku

Construção incremental:

- Passo: casa do tabuleiro
- Opções: números de 1 a 9
- Complexidade:



1	2	9	●		
8	5	3			
7	4	6			
2	6	8			
5	9	4			
3	7	1			

# Backtracking

Sudoku

Construção incremental:

- Passo: casa do tabuleiro
- Opções: números de 1 a 9
- Complexidade:



1	2	9	●		
8	5	3			
7	4	6			
2	6	8			
5	9	4			
3	7	1			

# Backtracking

## Sudoku

```
1 bool filled[10][10];
2 int grid[10][10];
3 bool colval[10][10];
4 bool rowval[10][10];
5
6 int n;
7
8 bool solve(int i, int j) {
9     if(i == n and j == n) return true;
10    int nextj = (j+1)%n, nexti = i + (nextj == 0);
11
12    int dig = grid[i][j];
13
14    // is filled from input?
15    if(filled[i][j]) {
16        if(!colval[j][dig] and !rowval[i][dig]) {
17            colval[j][dig] = true, rowval[i][dig] = true;
18            if(solve(nexti, nextj)) return true;
19            colval[j][dig] = false, rowval[i][dig] = false;
20        }
21
22        return false;
23    }
24
25    for(dig=1; dig<=9; dig++) if(!colval[j][dig] and !rowval[i][dig]) {
26        colval[j][dig] = true, rowval[i][dig] = true;
27        if(solve(nexti, nextj)) return true;
28        colval[j][dig] = false, rowval[i][dig] = false;
29    }
30
31    return false;
32 }
```



# Backtracking

## Sudoku

```
1 bool filled[10][10];
2 int grid[10][10];
3 bool colval[10][10];
4 bool rowval[10][10];
5
6 int n;
7
8 bool solve(int i, int j) {
9     if(i == n and j == n) return true;
10    int nextj = (j+1)%n, nexti = i + (nextj == 0);
11
12    int dig = grid[i][j];
13
14    // is filled from input?
15    if(filled[i][j]) {
16        if(!colval[j][dig] and !rowval[i][dig]) {
17            colval[j][dig] = true, rowval[i][dig] = true;
18            if(solve(nexti, nextj)) return true;
19            colval[j][dig] = false, rowval[i][dig] = false;
20        }
21    }
22    return false;
23 }
24
25 for(dig=1; dig<=9; dig++) if(!colval[j][dig] and !rowval[i][dig]) {
26     colval[j][dig] = true, rowval[i][dig] = true;
27     if(solve(nexti, nextj)) return true;
28     colval[j][dig] = false, rowval[i][dig] = false;
29 }
30
31 return false;
32 }
```

Para estimar runtime:

- Complexidade de cada step
- Número de nós visitados na árvore implícita da recursão
- Número de candidatos relevantes
  - Se confia q a maioria dos candidatos grandes é válida, contar o número de candidatos válidos máximo pode levar a estimativas úteis
- Experiência com problemas/constraints parecidos

# Backtracking

```
1 data_structure candidate;
2
3 do_the_thing():
4     // incrementar contador se for um problema de contar candidatos validos
5     // appendar a um vetor se for um problema de enumerar candidatos validos
6     // habilitar um bool se for um problema de sim ou nao
7     // etc
8
9 // i -> numero do passo atual
10 backtracking(i):
11     if candidate.is_full_valid_candidate():
12         do_the_thing()
13         return
14
15     for option in curr_options():
16         if is_valid_step(option):
17             candidate.add_step(option)
18             backtracking(i+1)
19             candidate.remove_step(option)
```

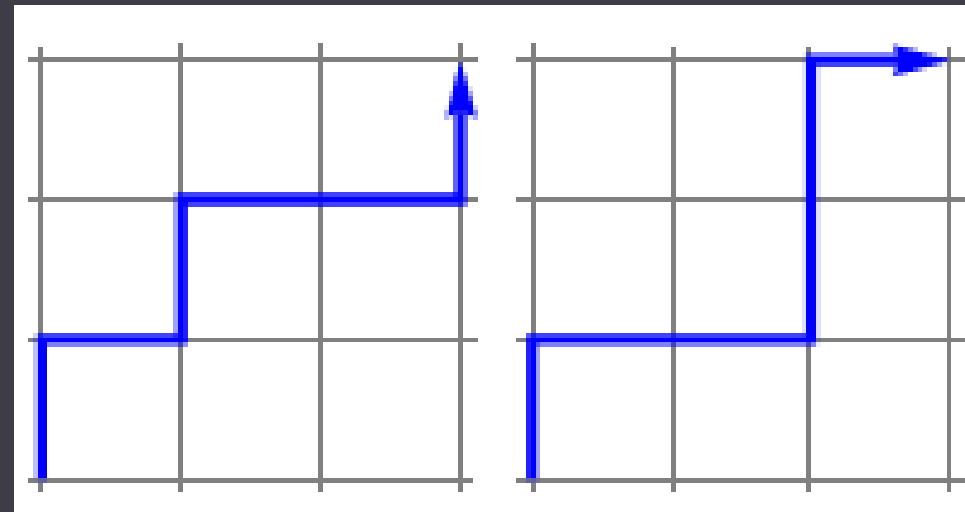
# Bruteforce/Backtracking

Pontos de atenção:

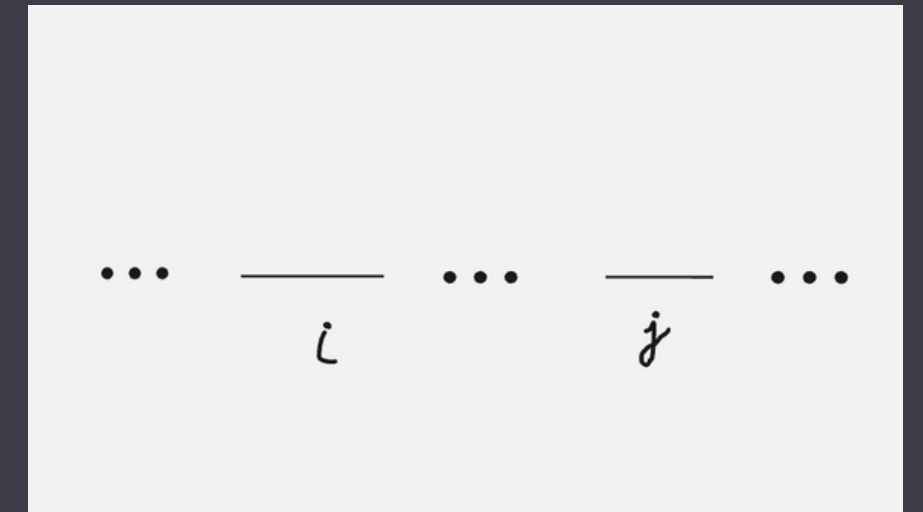
- Quais são os candidatos relevantes
- Complexidade/Estimativa
- Constraints

$$l \leq x \leq r$$

$$O(r - l + 1)$$



$$O(?)$$



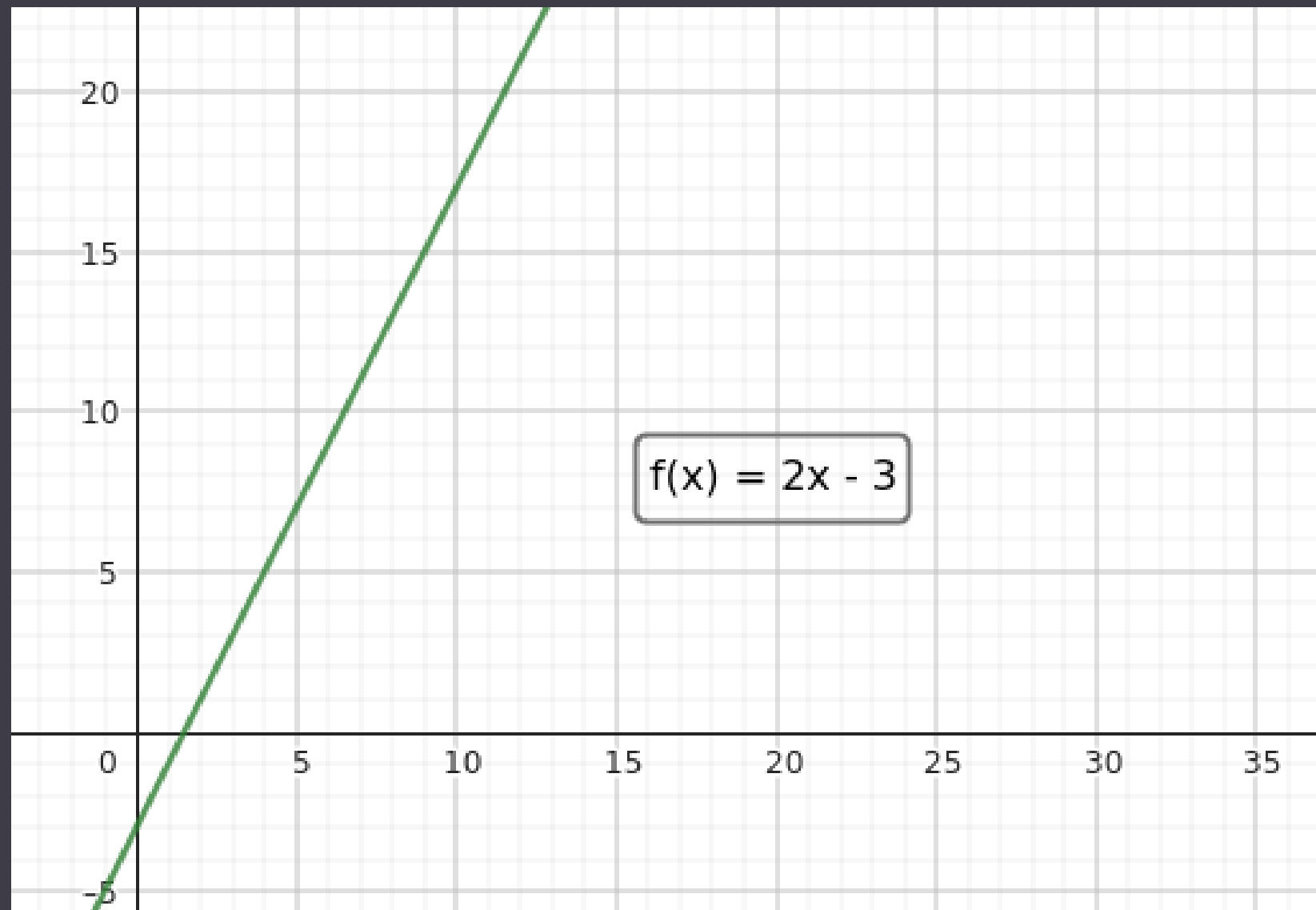
$$O(n^2)$$



# **BUSCA BINÁRIA**

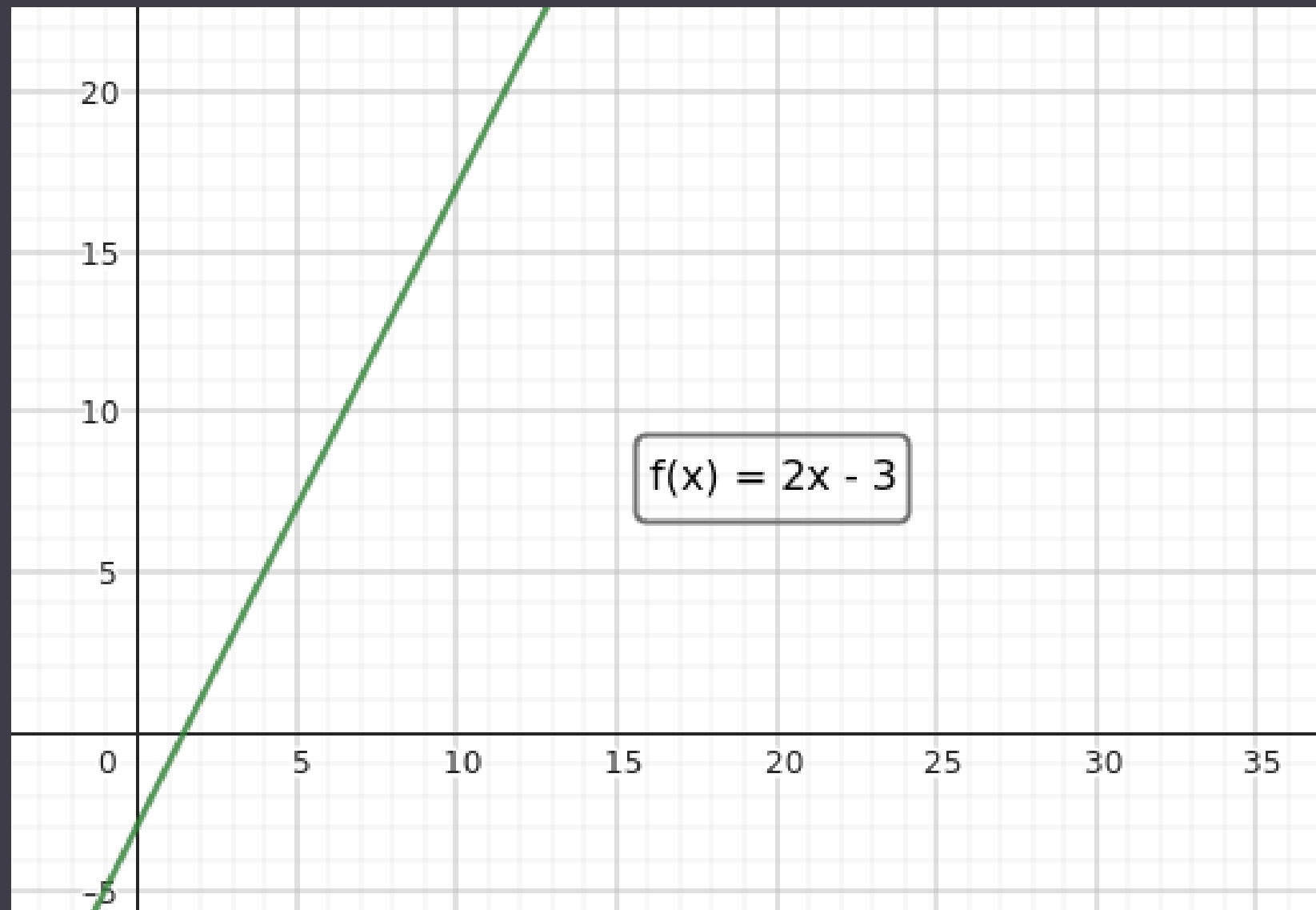
**AULA 2 - BUSCAS**

# Problema



Temos uma função monótona e queremos saber para qual maior  $x$  tal que  $f(x) < k$

# Problema

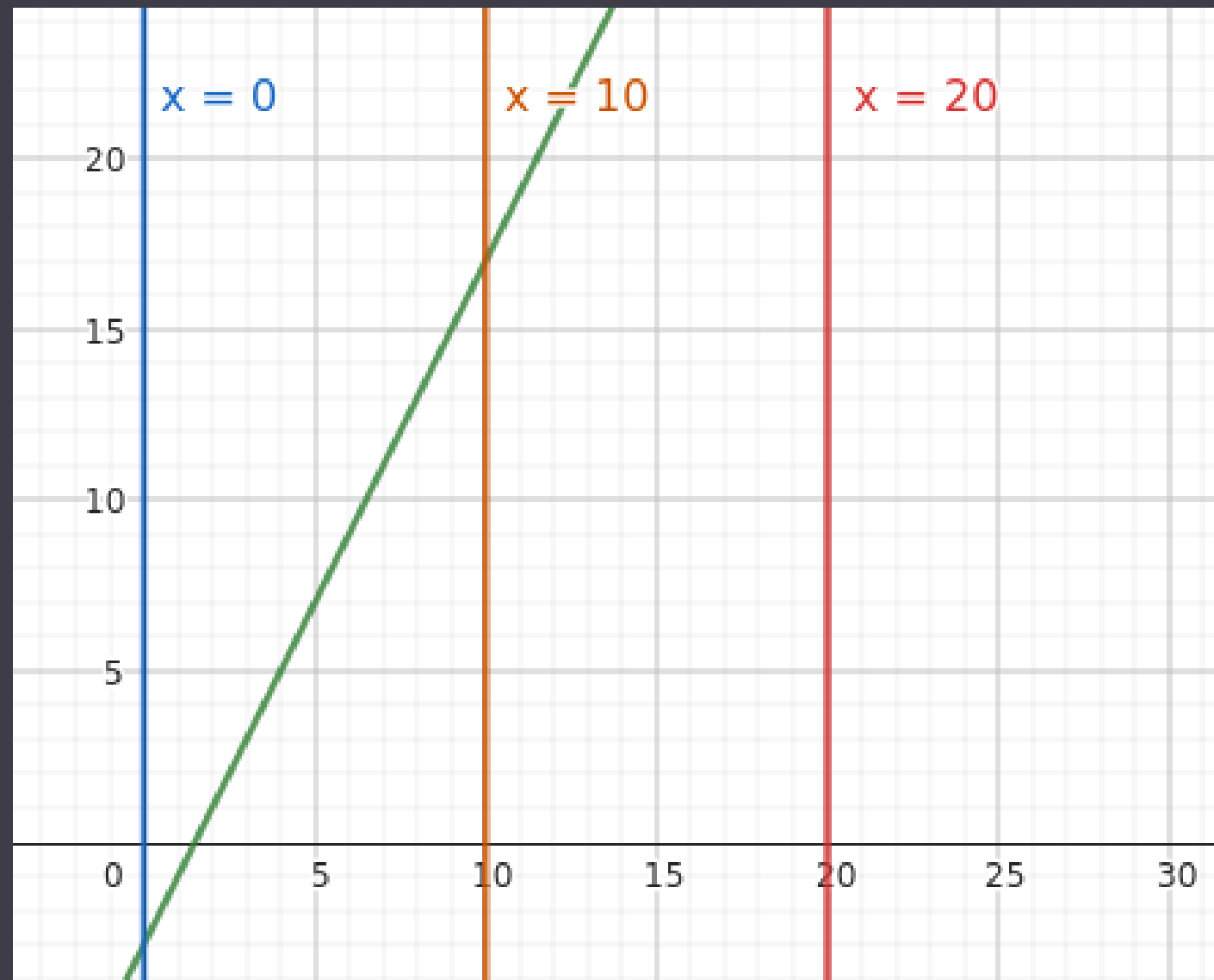


Temos uma função monótona e queremos saber para qual maior  $x$  tal que  $f(x) < k$

~~Solução simples: matemática~~

Solução geral: busca binária

# Problema



Criamos um intervalo em que temos certeza que a resposta está lá:  $[0, 20]$   
ex:  $k = 20$

Dividimos o intervalo em dois:  
 $[left, mid]$  e  $[mid, right]$

# Problema



Dividimos o intervalo em dois:  
[left, mid] e [mid, right]

Sabemos que eh monotonicamente  
crescente ( $a > 0$ )

Então se  $f(\text{mid}) < 20$ , sabemos que  
todo o intervalo  $[0, \text{mid}]$  não  
precisamos olhar mais, e olhamos  
apenas para  $[\text{mid} + 1, \text{right}]$ .



# Busca binária

A busca binária vai funcionar para qualquer tipo de função monótona.

## Código

$O(\log(n))$

$n$  é o tamanho do intervalo de busca

```
1 binSearch(l, r, f, k) -> integer:
2   best = -1
3   while l <= r do
4     mid = (l+r)/2
5     if f(mid) < k then
6       best = mid
7       l = mid + 1
8     else
9       r = mid - 1
10  return best
```

# Problema

A histogram is a polygon made by aligning  $N$  adjacent rectangles that share a common base line. Each rectangle is called a bar. The  $i$ -th bar from the left has width 1 and height  $H_i$ .

Your goal is to find the area of the largest rectangle contained in the given histogram, such that one of the sides is parallel to the base line.

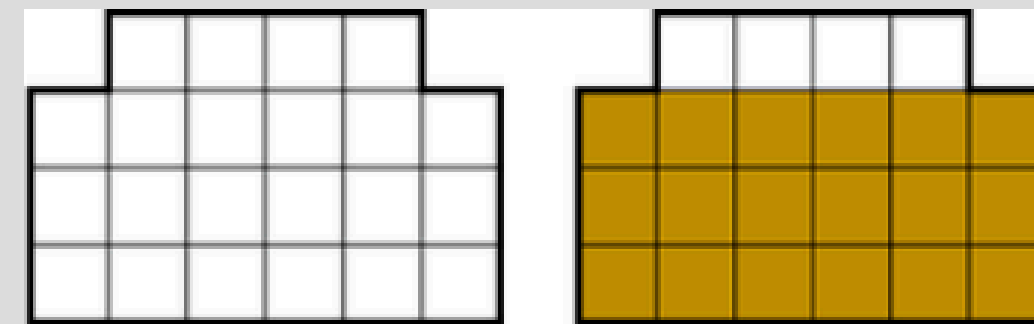


Figure 1. The histogram given in the example, with the largest rectangle shown on the right.

Actually, no, you have to find the largest **square**. Since the area of a square is determined by its side length, you are required to output the **side length** instead of the area.

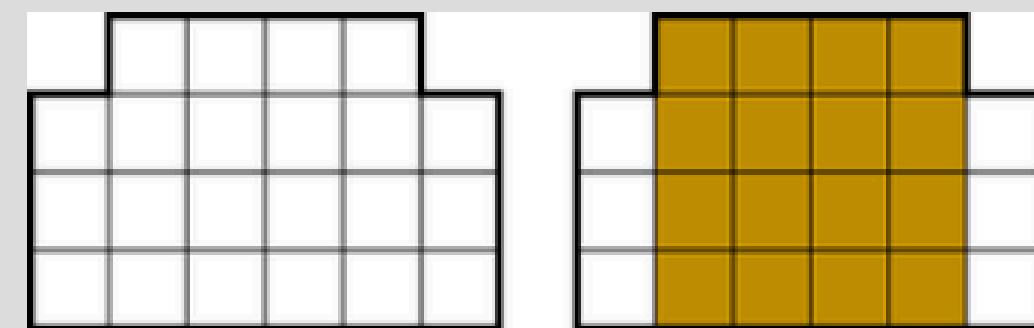


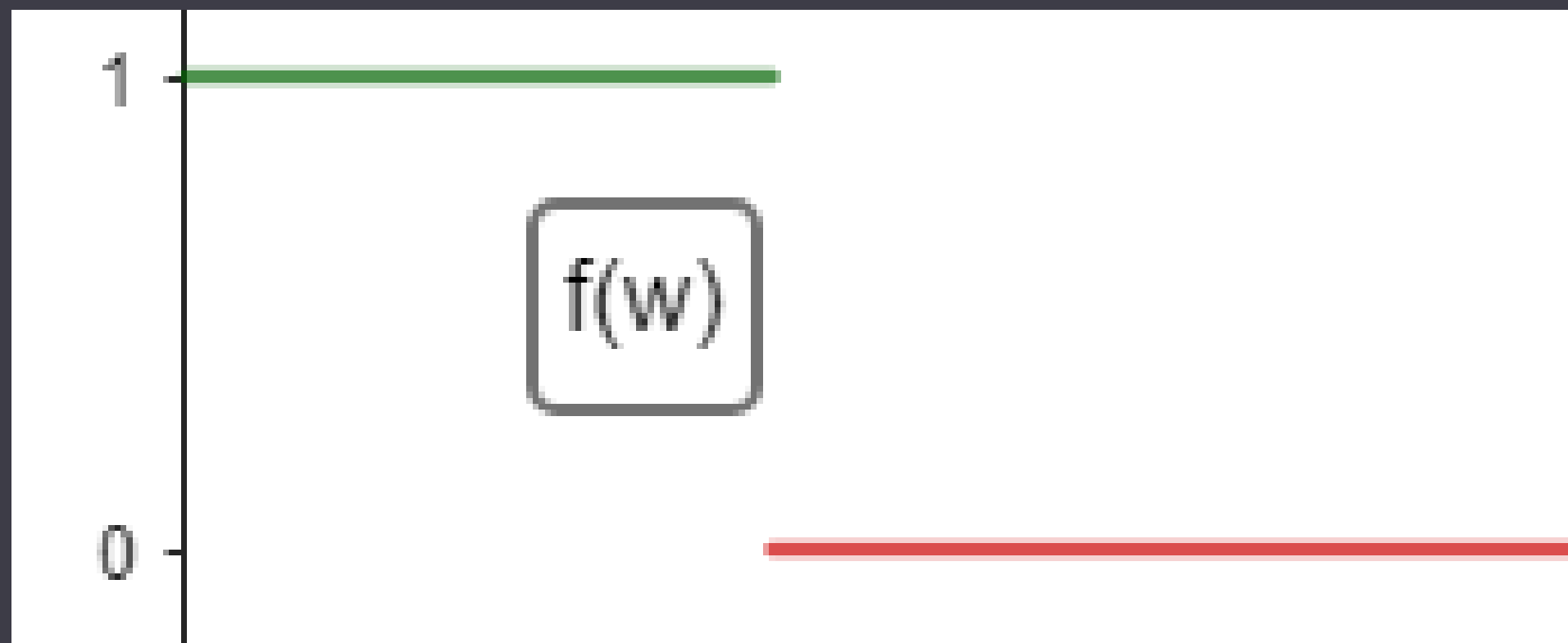
Figure 2. The histogram given in the example, with the largest **square** shown on the right.

Apesar da solução esperada ser stack monotônica, vamos pensar numa solução alternativa pra fins de aprendizado

Seja  $f(w)$  a função que retorna 1 se é possível e 0 se impossível com  $w$ .

# Busca binária na resposta

Seja  $f(w)$  a função que retorna 1 se é possível e 0 se impossível com  $w$ .



Podemos perceber que  $f(w)$  é monótona e parecida com isso.

Queremos  $w$  mais a direita t.q  $f(w) = 1$

**Busca binária !!**

# Como fazer $f(w)$ ?

Como saber se existe um quadrado de lado  $w$ ?

# Como fazer $f(w)$ ?

Como saber se existe um quadrado de lado  $w$ ?

$w$  elementos seguidos  $\geq w$

```
1 hasSquareSideW(A[0,n-1], W) -> boolean:
2   l = 0
3   for r = 0 to n - 1 do
4     if A[r] < W then
5       l = r + 1
6     if r - l + 1 == W then
7       return true
8   return false
```

Complexidade com busca binária:  $O(n \log \text{MAXW})$

# STL: upper e lower bound

Retornam iterators!

`vector<int>::iterator`

```
// 10 10 10 20 20 20 30 30
//           ^
//                   ^
```

Quantidade de  
elementos em  
 $O(\log n)$

constante melhor que map!

```
auto lb = lower_bound (v.begin(), v.end(), 20);
auto ub = upper_bound (v.begin(), v.end(), 20);
int qt = ub - lb;
```

lower\_bound: primeiro elemento  $\geq x$

upper\_bound: primeiro elemento  $> x$



# SUBSET SUM

AULA 2 - BUSCAS



# Problema

Temos um conjunto de  $N \leq 20$  inteiros e queremos saber se existe algum subconjunto que somam  $k$ .

Input:

4 5

1 2 3 2

Output:

Yes

# Problema

Temos um conjunto de  $N \leq 20$  inteiros e queremos saber se existe algum subconjunto que somam  $k$ .

Input:

4 5

1 2 3 2

Output:

Yes

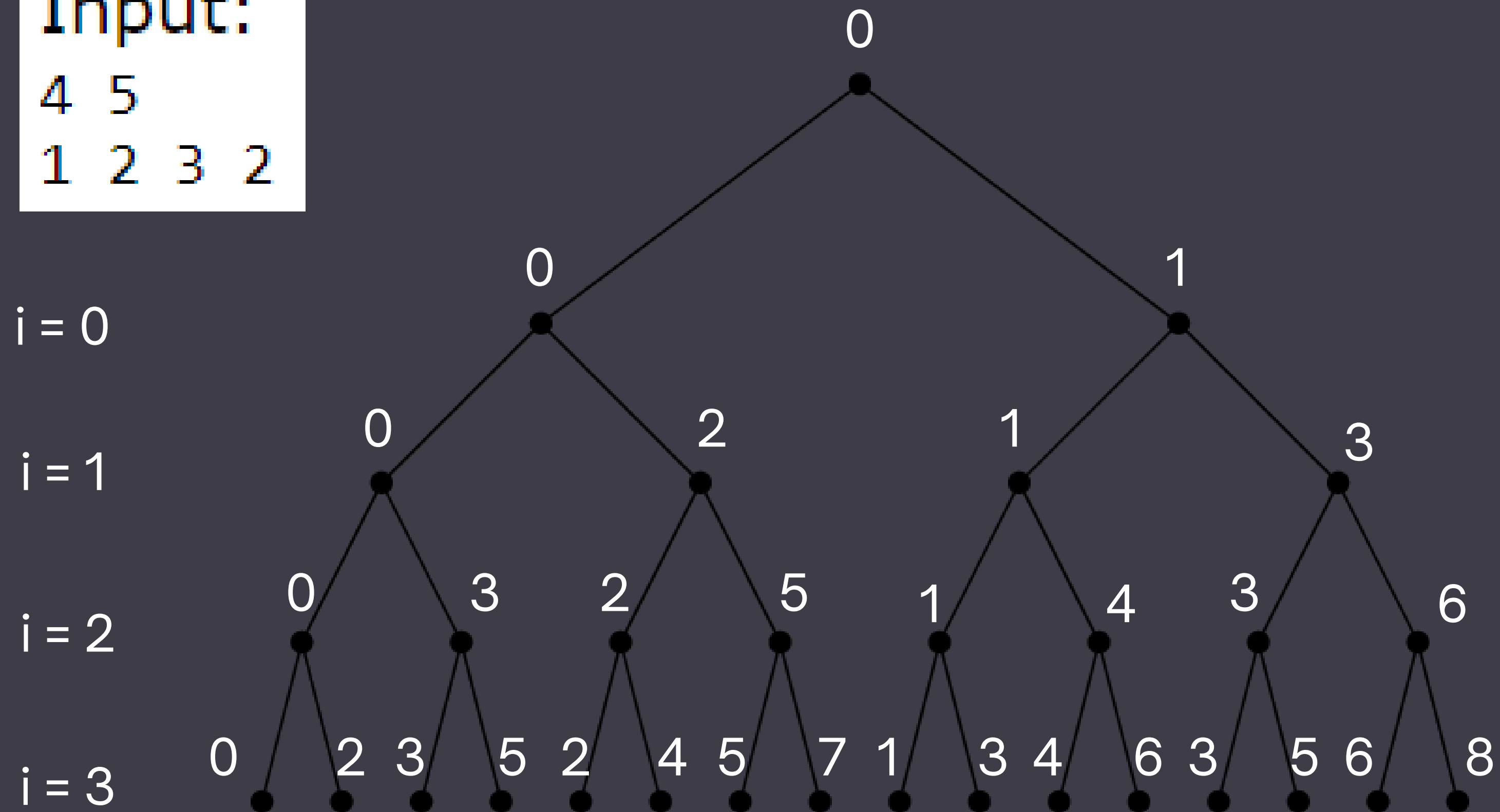
Testar todos subconjuntos =  $O(2^n)$

$2^{20} \approx 1e6$  (passa no TL de 1s)

Input:

4 5

1 2 3 2



# Código

```
1 rec(i, sum, k, A[0,n-1]) -> boolean:
2   if sum == k then
3       return true
4   if i == n then
5       return false
6   if rec(i+1, sum, k, A) then
7       return True
8   return rec(i+1, sum + A[i], k, A)
9
10
11 subsetSum(A[0,n-1], k) -> boolean:
12   return rec(0, 0, k, A)
```

Como fazer para gerar todos  
os subsets com soma k?



# MEET IN THE MIDDLE

AULA 2 - BUSCAS

# Problema

Temos um conjunto de  $N \leq 40$  inteiros e queremos saber se existe algum subconjunto que somam  $k$ .

Input:

4 5

1 2 3 2

Output:

Yes

# Problema

Temos um conjunto de  $N \leq 40$  inteiros e queremos saber se existe algum subconjunto que somam  $k$ .

Input:

4 5

1 2 3 2

Output:

Yes


Testar todos subconjuntos =  $O(2^n)$

$2^{40} \approx 1e12$  (não passa no TL de 1s)



# Solução?



# Solução?



**The Black Eyed Peas - Meet Me Halfway (Official Music Video)**  
569 mi de visualizações · há 14 anos

 Black Eyed Peas 

#TheBlackEyedPeas #MeetMeHalfway #Remastered #VevoCertified #Pop #OfficialMusicVideo.

Legendas

# Solução?

Conseguimos gerar todos os subconjuntos do problema dividindo em dois:

# Solução?

Conseguimos gerar todos os subconjuntos do problema dividindo em dois:

Ex:

```
Input:
4 5
1 2 3 2
```

# Solução?

Conseguimos gerar todos os subconjuntos do problema dividindo em dois:

Ex:

**Input:** *Grupo 1* : {1, 2}  
4 5  
1 2 3 2 *Grupo 2* : {3, 2}

Geramos a subset sum de cada grupo

*Grupo 1 :  $\{1, 2\}$*

*Grupo 2 :  $\{3, 2\}$*

*Subsets 1 :  $\{0, 1, 2, 3\}$*

*Subsets 2 :  $\{0, 3, 2, 5\}$*

O produto cartesiano nos dá todos os subsets do problema

mas não vamos fazer diretamente por que voltaria pro  $2^n$

*Subsets 1* :  $\{0, 1, 2, 3\}$

*Subsets 2* :  $\{0, 3, 2, 5\}$

O produto cartesiano nos dá todos os subsets do problema

(mas não vamos fazer diretamente por que voltaria pro  $2^n$ )

**Solução: Two Sum**

# Código

```
1 subsetSum_middle(A[0,n-1], k) -> boolean:  
2     sub_1 = gen_subsets(A[0,n/2])  
3     sub_2 = gen_subsets(A[n/2+1,n-1])  
4     return two_sum(sub_1, sub_2, k)
```

$$O( 2^{(n/2)} * (n/2) )$$

Two Sum em  $O(n \log n)$

$$N = 40 \approx 2e7$$

(passa, mas cuidado com constantes: preferir sort em vez de map/set)



**That's all, Folks!**

