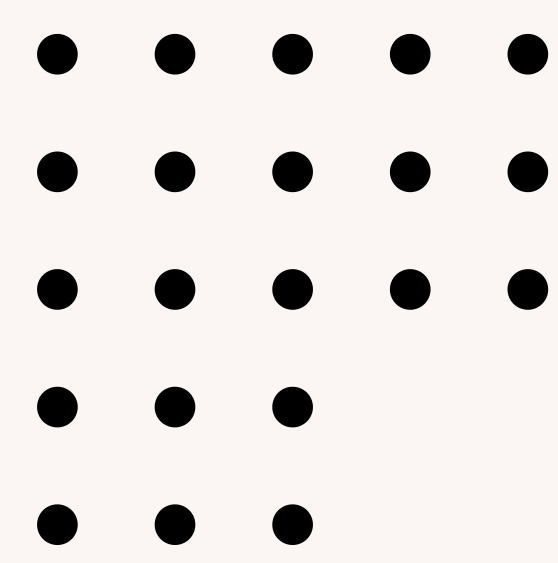
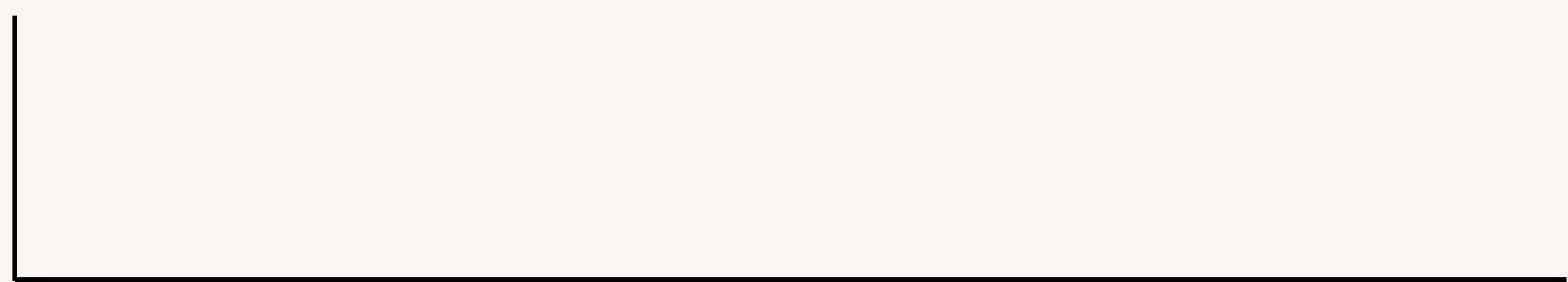




INTRODUÇÃO AO C++ E À ANÁLISE DE ALGORITMOS



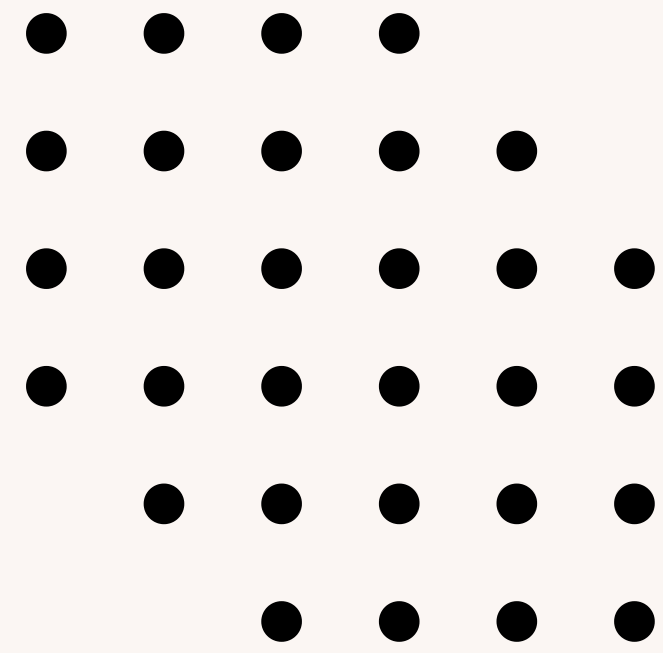
TÓPICOS DE ABORDAGEM

- Sintaxe do C++
 - Estrutura padrão
 - Entrada, saída e variáveis
 - Condicionais
 - Laços
 - Arrays



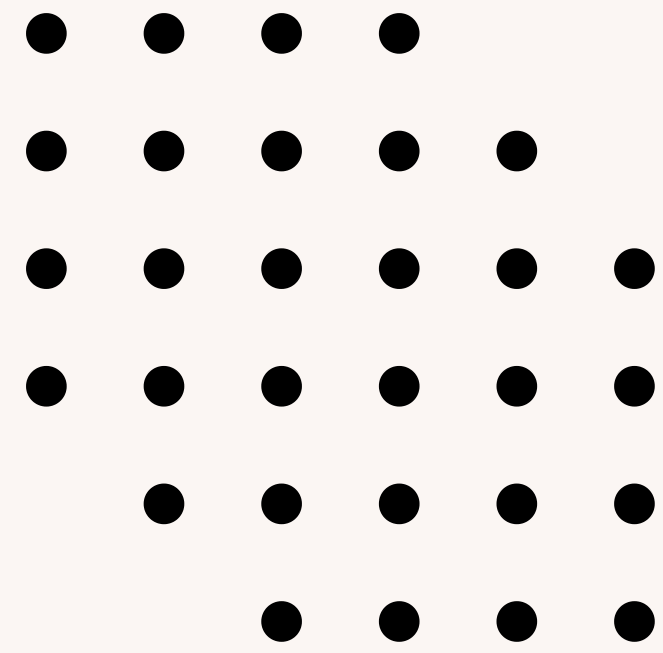
TÓPICOS DE ABORDAGEM

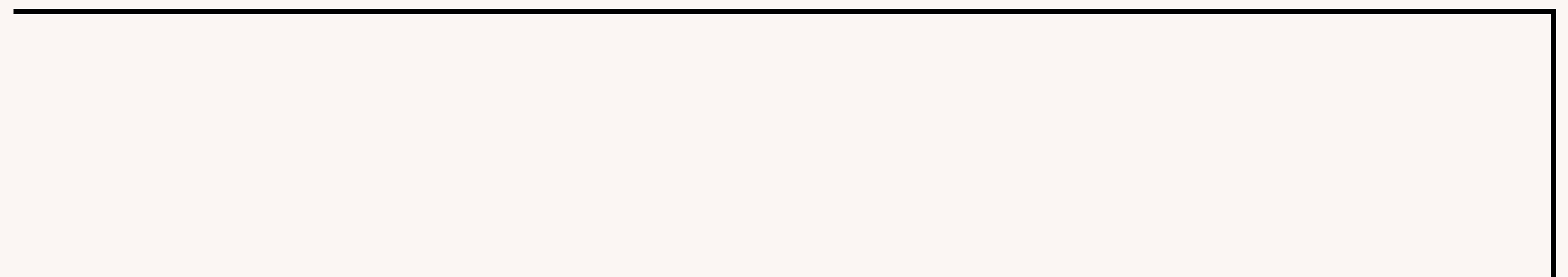
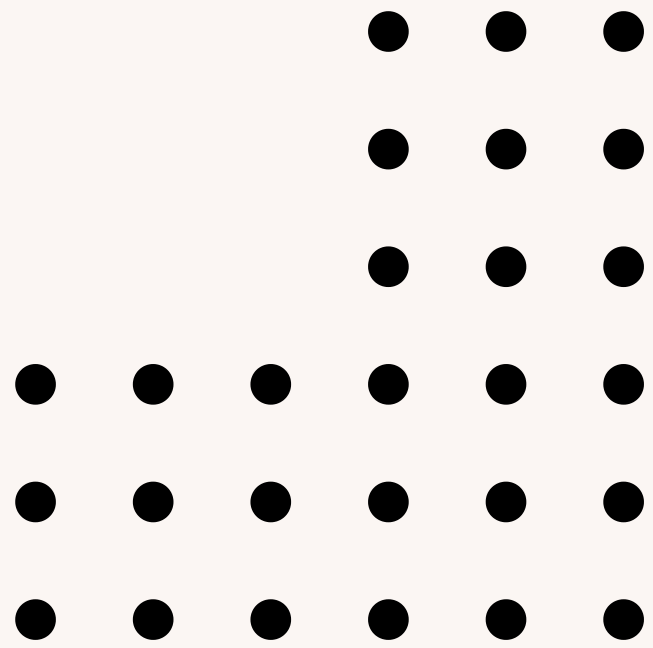
- Estrutura das competições
 - Estrutura do problema
 - Vereditos



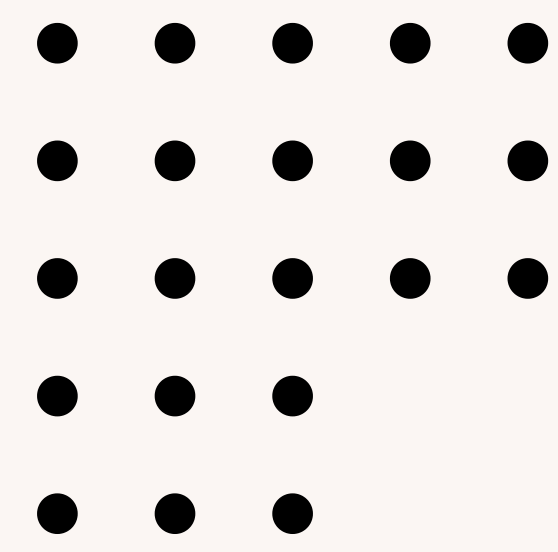
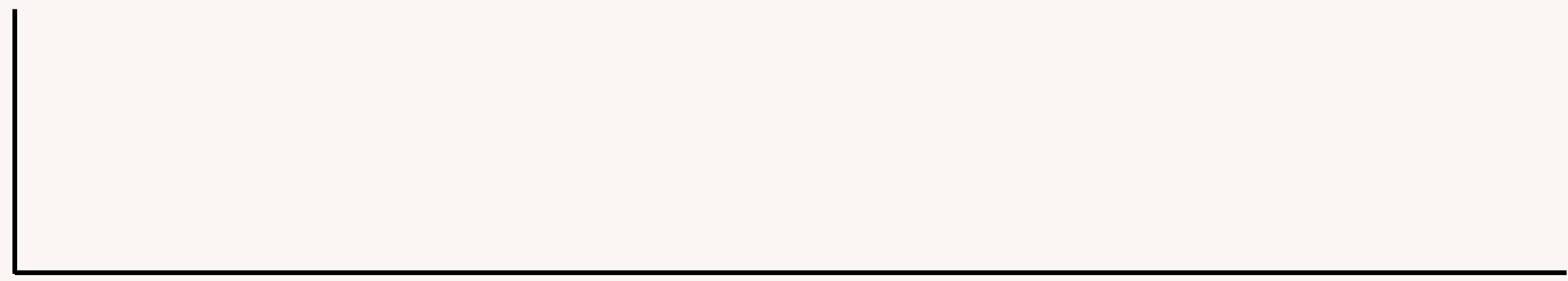
TÓPICOS DE ABORDAGEM

- Análise de algoritmos
 - Notação BIG O
 - Estimando tempo de execução





SINTAXE DO C++



ESTRUTURA PADRÃO

Biblioteca - Importa código de outros lugares, a `bits/stdc++.h` é uma biblioteca que inclui todas as outras

Namespace - Faz com que a gente tenha acesso a tudo da biblioteca padrão diretamente, sem especificá-la. Não é obrigatório, mas economiza muito código.

Main - Todo o código executável ficará aqui

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     cout << "Hello World!";
6 }
```

VARIÁVEIS

Variáveis guardam valores. Existem vários tipos de variáveis.

int/long long - Guardam inteiros que possuem até 32/64 bits.

double/long double - Armazenam números decimais

string - armazena texto

char - armazena um caractere

bool - armazena verdadeiro ou falso e outros...

```
1 int x = 5;  
2 long long y = 10000000000000;  
3 double z = 10.5;  
4 long double w = 5e-9;  
5 string s = "texto";  
6 char c = 'c';  
7 bool b = true;
```

ENTRADA/SAÍDA

Usamos cout (console out) para imprimir coisas na tela.

Usamos cin (console in) para que o usuário digite um valor para ser armazenado nas variáveis.

```
1  int x, y, z;
2  string s;
3  cout << "texto";
4  cout << 'c';
5  cout << 2;
6  cout << 3.5 << ' ' << 50;
7  cin >> x >> y >> z;
8  cin >> s;
```


CONDICIONAIS

Usamos para que o nosso código faça escolhas.

IF(condição) - Se a condição for verdade, execute este código

ELSE IF (condição) - Senão, se a nova condição for verdade então execute este código

ELSE - Se nenhuma condição for verdade execute este código

```
1  if(a < b) {  
2      cout << "a é menor que b";  
3  } else if (a == b) {  
4      cout << "b é igual ao a";  
5  } else {  
6      cout << "a é maior que b";  
7  }
```

COMPARADORES

Para comparar objetos e usar nas condições

$a < b$ - verdadeiro se a é menor que b

$a > b$ - verdadeiro se a maior que b

$a == b$ - verdadeiro se a igual a b

$a >= b$ - verdadeiro se a maior igual a b

$a <= b$ - verdadeiro se a menor igual a b

$a != b$ - verdadeiro se a diferente de b

```
1  if(a < b);  
2  if(a > b);  
3  if(a == b);  
4  if(a >= b);  
5  if(a <= b);  
6  if(a != b);
```

OPERADORES LÓGICOS

Para combinar condições

$a \ \&\& \ b$ - verdadeiro se a E b forem verdadeiros

$a \ || \ b$ - verdadeiro se a OU b forem verdadeiros

$!a$ - o contrário do a (verdadeiro se a for falso e vice-versa)

$a \ ^ \ b$ - verdadeiro se exatamente um dos a , b for verdadeiro

```
1  if(a && b)
2  if(a || b)
3  if(!a)
4  if(a ^ b);
```

LAÇOS – WHILE E DO WHILE

Usamos laços para executar código repetidamente.

WHILE(condição) - Enquanto a condição for verdade, repita este código

DO - WHILE(condição) - Parecido, mas executa o código antes de checar se pode repetir.

```
1  int a = 5, b = 10;  
2  while(a <= b) {  
3      a++; // equivale à: a = a + 1  
4  }
```

```
1  do {  
2  
3  } while(a < b);
```

LAÇOS – FOR

FOR(início; condição; iteração)

início - Comando que será executado quando o laço iniciar

condição - O laço irá repetir enquanto a condição for verdade

iteração - Comando que será executado toda vez que o laço repetir

```
1  for(int i=0; i<=50; i++) {  
2      // execute esse código 50 vezes  
3      cout << i;  
4  }
```

ARRAYS

Usamos arrays para armazenar listas de coisas.

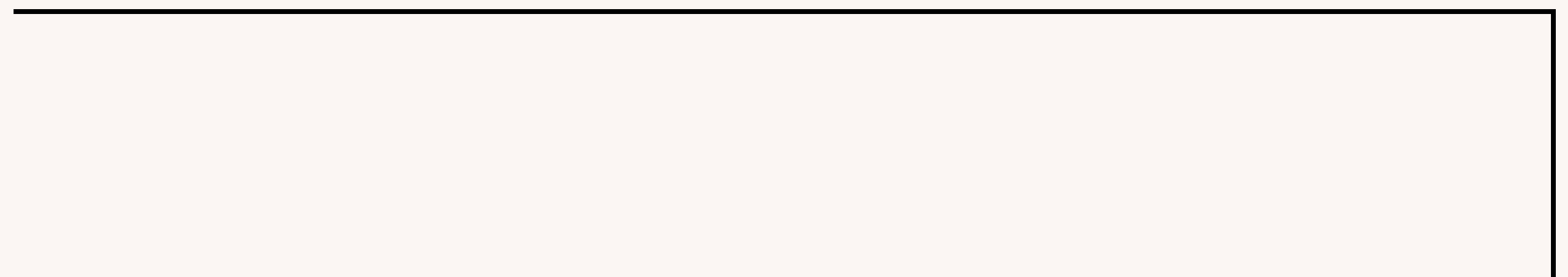
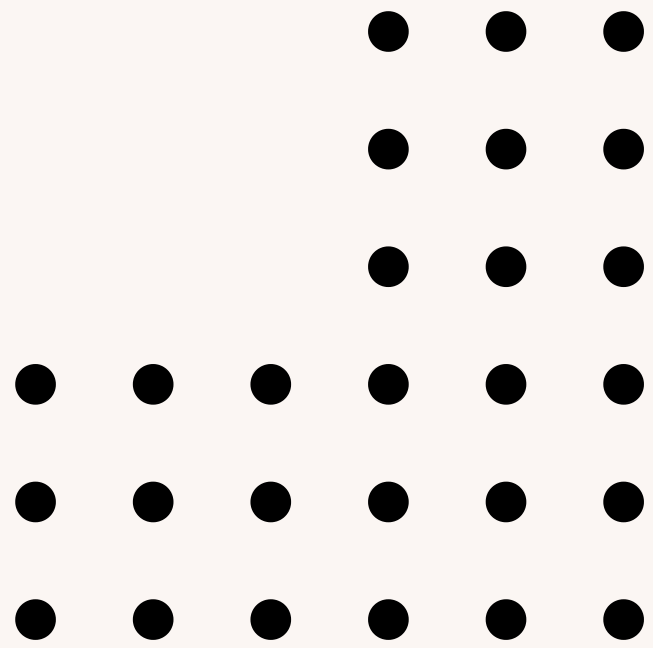
Ex:

`int arr[50]` - uma lista com 50 inteiros

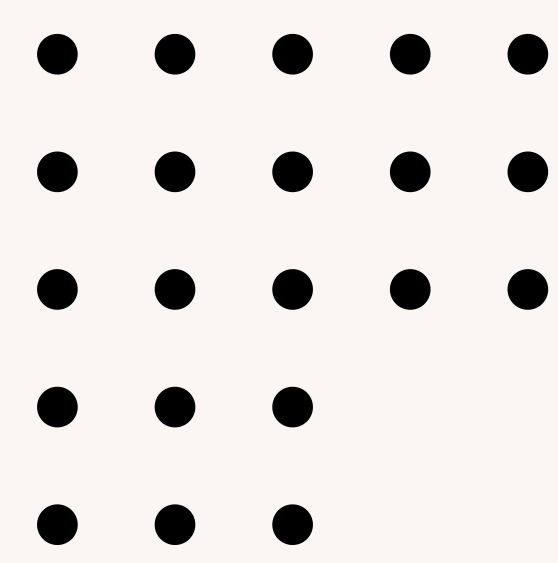
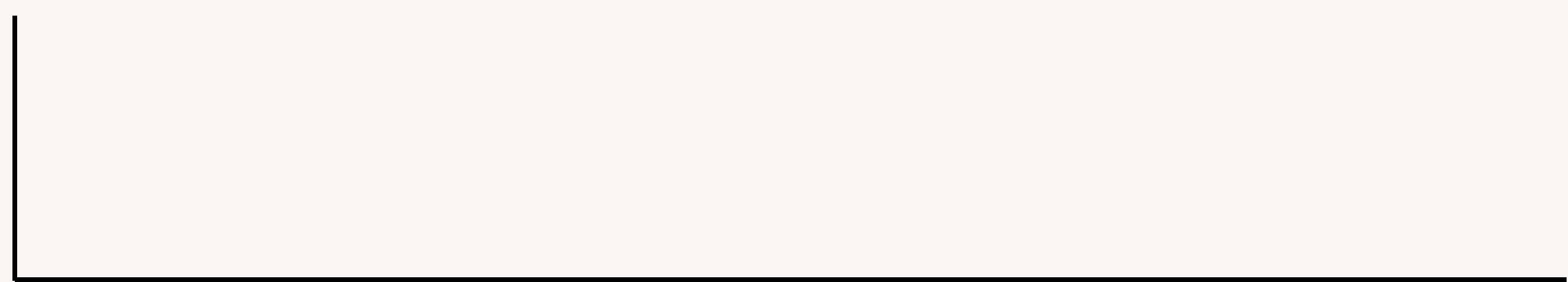
Para acessar o primeiro elemento da lista usamos `arr[0]`, o segundo elemento é o `arr[1]` e assim por diante...

As posições vão sempre de 0 até `TAMANHO-1`, tome cuidado com isso.

```
1  int arr[50]; // declaração
2
3  arr[0]; // <-- primeiro elemento da lista
4  arr[1]; // <-- segundo elemento
5  arr[49]; // <-- ultimo elemento
6  arr[50]; // <-- ILEGAL:
```

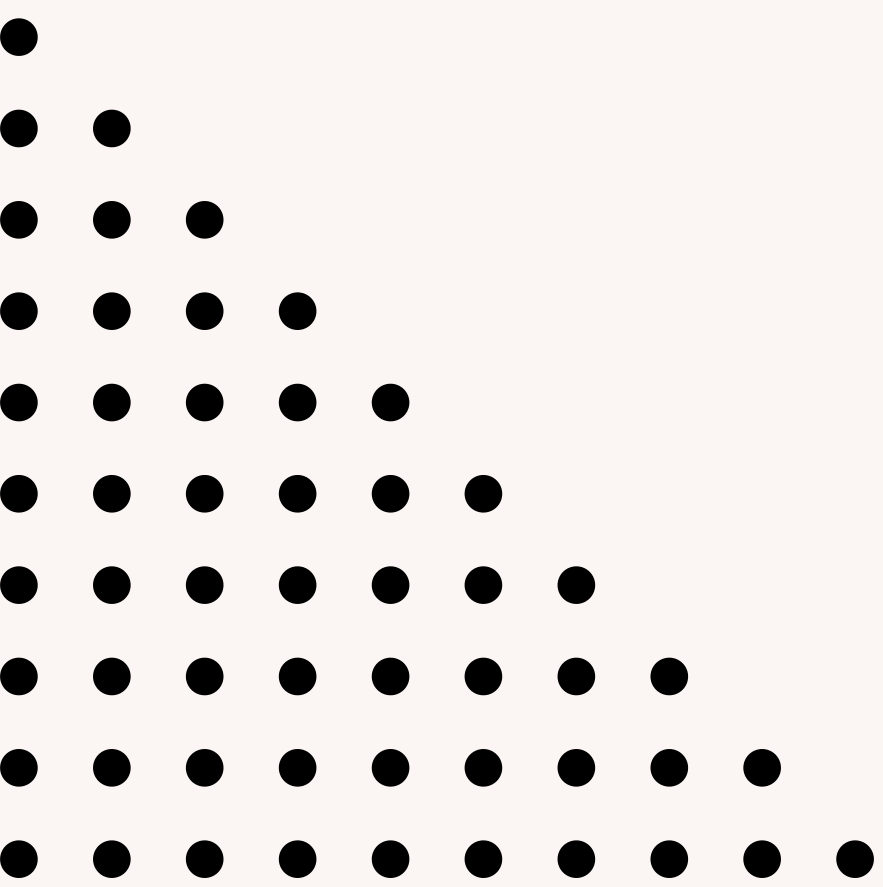


COMO FUNCIONAM OS PROBLEMAS?





OS PROBLEMAS DAS COMPETIÇÕES SÃO COMPOSTOS DE...

- 
- 1 Enunciado
 - 2 Descrição da entrada
 - 3 Descrição da saída
 - 4 Restrições (OBI e IOI)
 - 5 Exemplos

IMPRIMA EXATAMENTE 0 QUE O PROBLEMA PEDE

Nada de mensagens como “digite um número...”. O sistema irá achar que isso faz parte da sua resposta

```
1 int n;  
2 cout << "Digite um número, por favor: ";  
3 cin >> n;
```

RESTRIÇÕES

Você não precisa checar no seu código se as restrições são satisfeitas.

As restrições são uma garantia da pessoa que elaborou o problema: “seu código não será testado fora destes limites”

```
1  int n;  
2  cin >> n;  
3  if(n >= 100) {  
4      cout << "erro no n, está fora dos limites";  
5  }
```

VEREDITOS

Ao testar seu código, o sistema pode dar alguns vereditos:

Accepted - Você acertou a questão

Wrong answer - Seu código retornou uma resposta incorreta

Runtime Error - Seu código deu erro na execução

Time limit exceeded - Seu código não foi eficiente e estourou o limite de tempo

Memory limit exceeded - Seu código estourou o limite de memória

#59435797 | iangz's solution for [SPOJ-HELLO] [Problem A] ✕

Status	Time	Memory	Length	Lang	Submitted	Open	Share text ?	RemoteRunId
Accepted	10ms	5325kB	85	C++14 (gcc 8.3)	2025-03-22 21:04:03	✓	<input type="checkbox"/>	34345295

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     cout << "Hello World!";
6 }
```

Leave a comment

VEREDITOS

Ao testar seu código, o sistema pode dar alguns vereditos:

Accepted - Você acertou a questão

Wrong answer - Seu código retornou uma resposta incorreta

Runtime Error - Seu código deu erro na execução

Time limit exceeded - Seu código não foi eficiente e estourou o limite de tempo

Memory limit exceeded - Seu código estourou o limite de memória

#59435797 | iangz's solution for [SPOJ-HELLO] [Problem A] ✕

Status	Time	Memory	Length	Lang	Submitted	Open	Share text ?	RemoteRunId
Accepted	10ms	5325kB	85	C++14 (gcc 8.3)	2025-03-22 21:04:03	✓	<input type="checkbox"/>	34345295

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     cout << "Hello World!";
6 }
```

Leave a comment

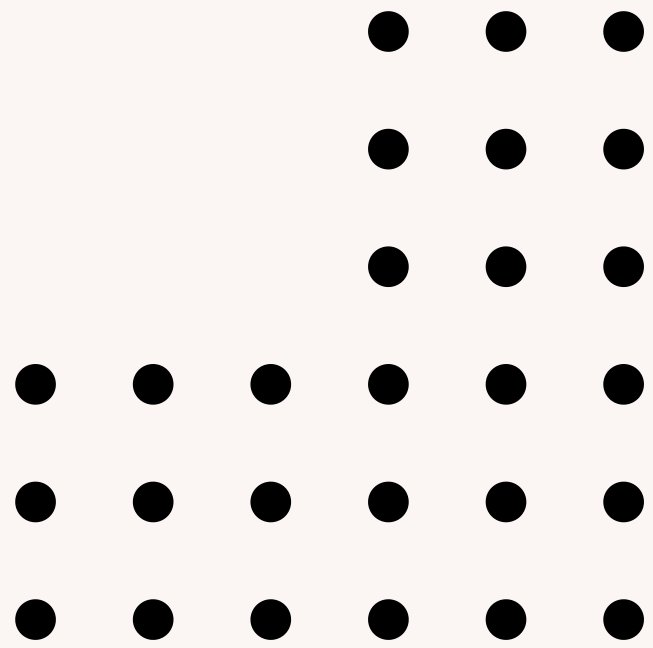
COMO PREVINIR...?

WA - Prove sua ideia, teste nos exemplos e construa mais testes.

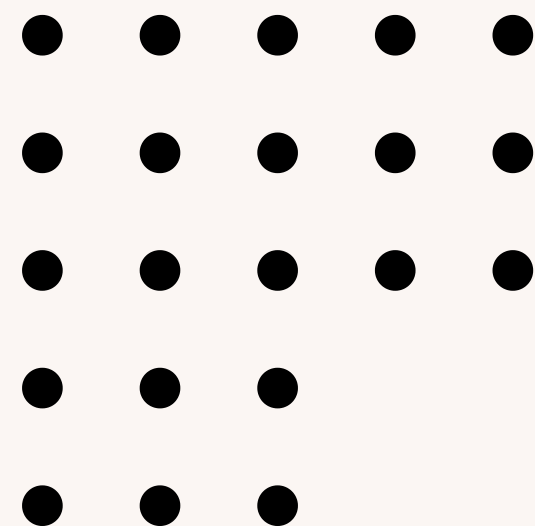
MLE - A experiência irá te permitir que você estime a quantidade de memória gasta com mais precisão. Ex: 10^8 inteiros aprox. 1GB RAM

TLE - Existem técnicas para estimar tempo de execução (próxima seção)





ANÁLISE DE ALGORITMOS



SOBRE

Devemos analisar quantos “passos” os nossos algoritmos estão executando para estimar o tempo de execução.

Você consegue dizer quantos passos o algoritmo do lado executa em função da variável n ? Selecione o texto escondido no retângulo azul para ter a resposta

```
1  for(int i=0; i<n; i++) {  
2      cout << "1";  
3      cout << "2";  
4  }
```

QUAL O MAIS RÁPIDO?

Estes códigos fazem a mesma coisa. Suponha que o vetor *a* existe. Resposta no retângulo azul

```
1  for(int i=0; i<n; i++) {  
2      int soma = 0;  
3      for(int j=0; j<=i; j++) {  
4          soma = soma + a[j];  
5      }  
6      cout << soma << ' ';  
7  }
```

```
1  int soma = 0;  
2  for(int i=0; i<n; i++) {  
3      soma = soma + a[i];  
4      cout << soma;  
5  }
```



ANALISANDO TEMPO DOS ALGORITMOS

Pense no pior caso

“Qual o caso que vai exigir mais do meu algoritmo?”

Olhe para as restrições!

Calcule quantos passos serão executados no pior caso.

Ex: Se seu algoritmo executa n^2 passos, e o pior caso é quando $n = 10^5$. Então seu algoritmo executa no máximo 10^{10} passos

Use regra de três para estimar o tempo de execução no pior caso.

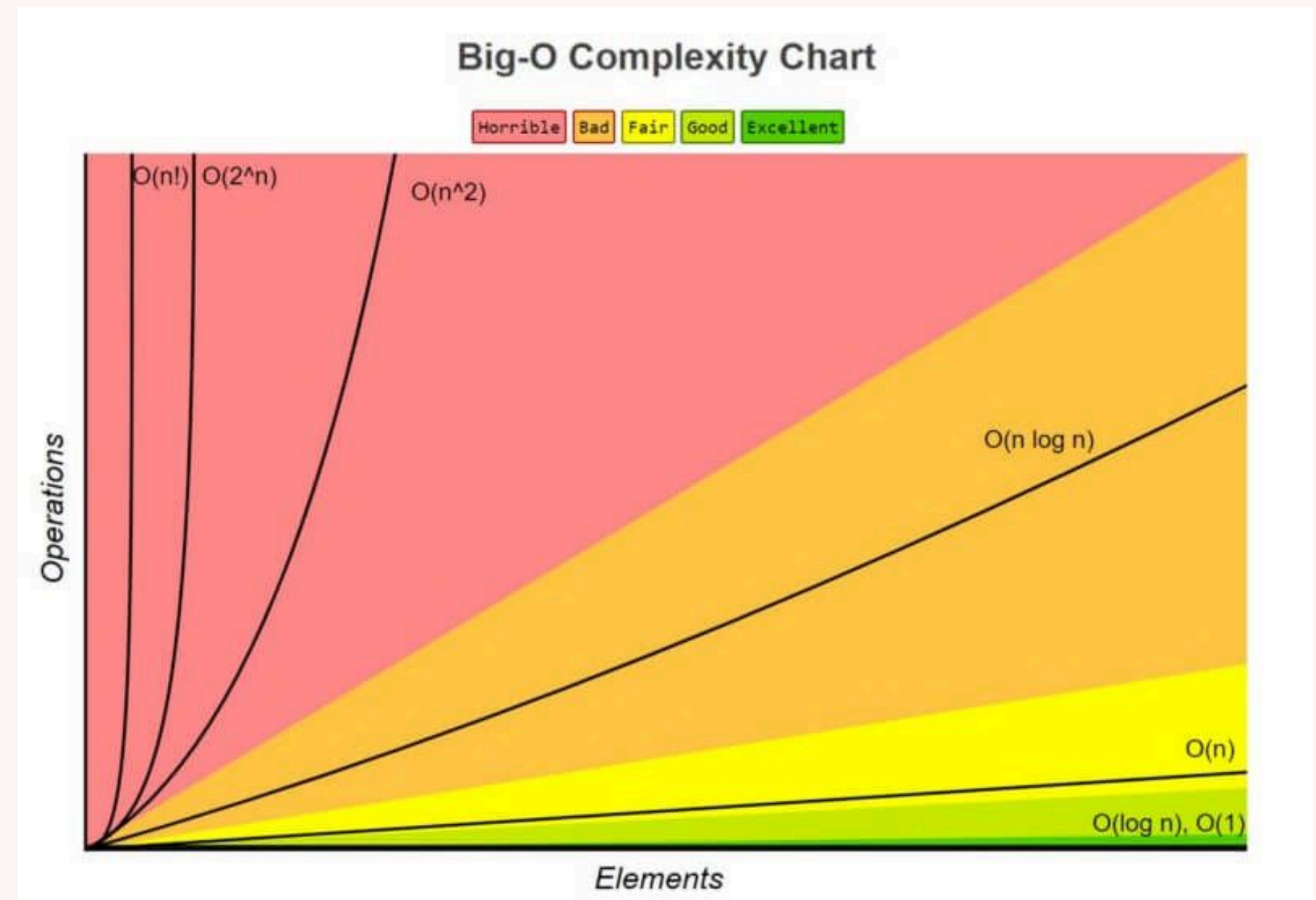
Um computador executa aproximadamente 10^8 operações por segundo. Um algoritmo que executa 10^{10} operações demoraria 100 segundos.

NOTAÇÃO BIG O

Vai facilitar nossas contas.

Serve para separar os algoritmos em “grupos” com taxa de crescimento no tempo parecidos.

intuitivamente dizemos que um algoritmo é $O(x)$ quando ele executa “por volta de” x passos (ou menos).



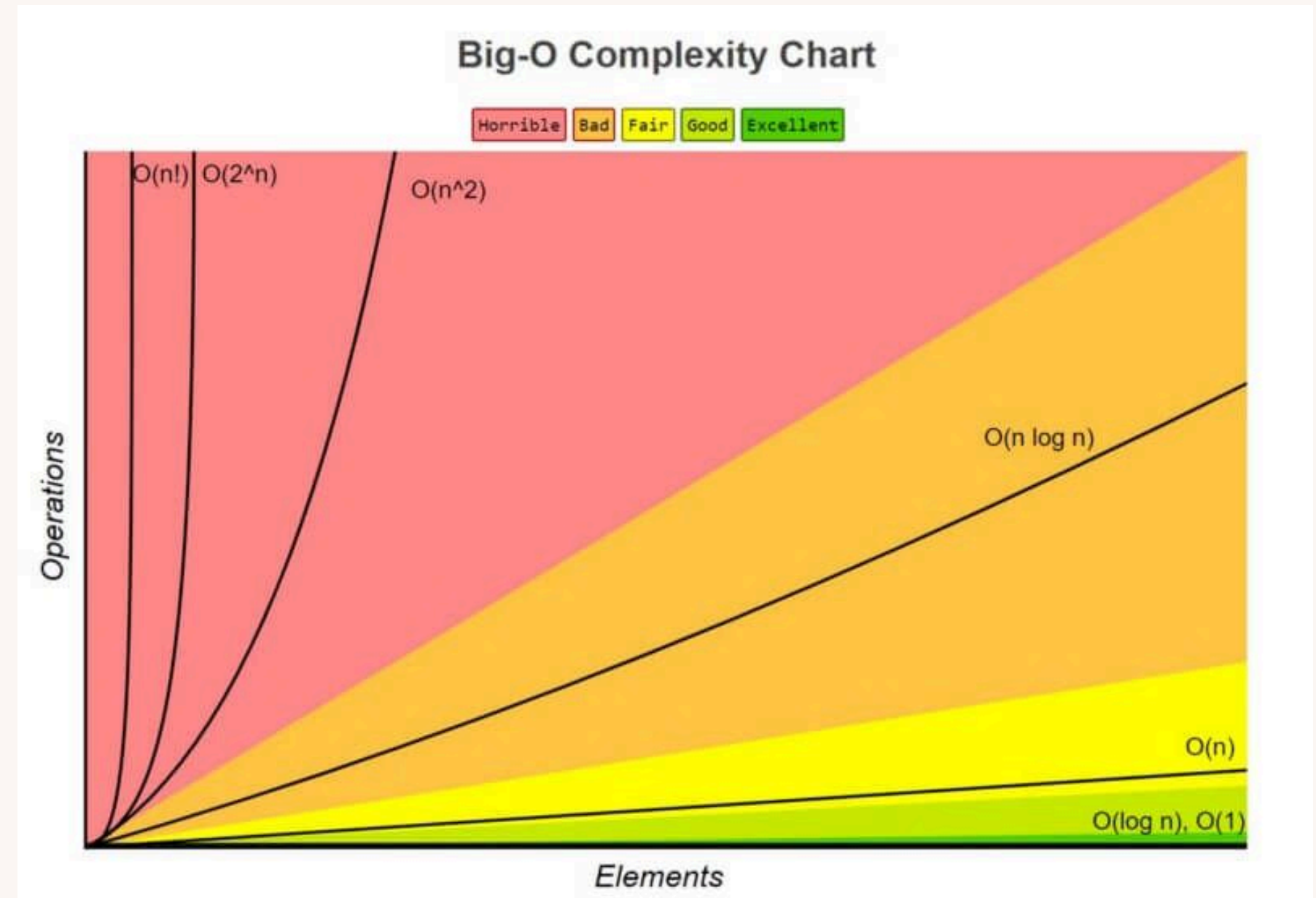
PROPRIEDADES

Apenas o termo que “cresce mais” importa.

Ex: um algoritmo que executa $n^2 + n$ passos pode ser considerado $O(n^2)$.

Podemos ignorar constantes

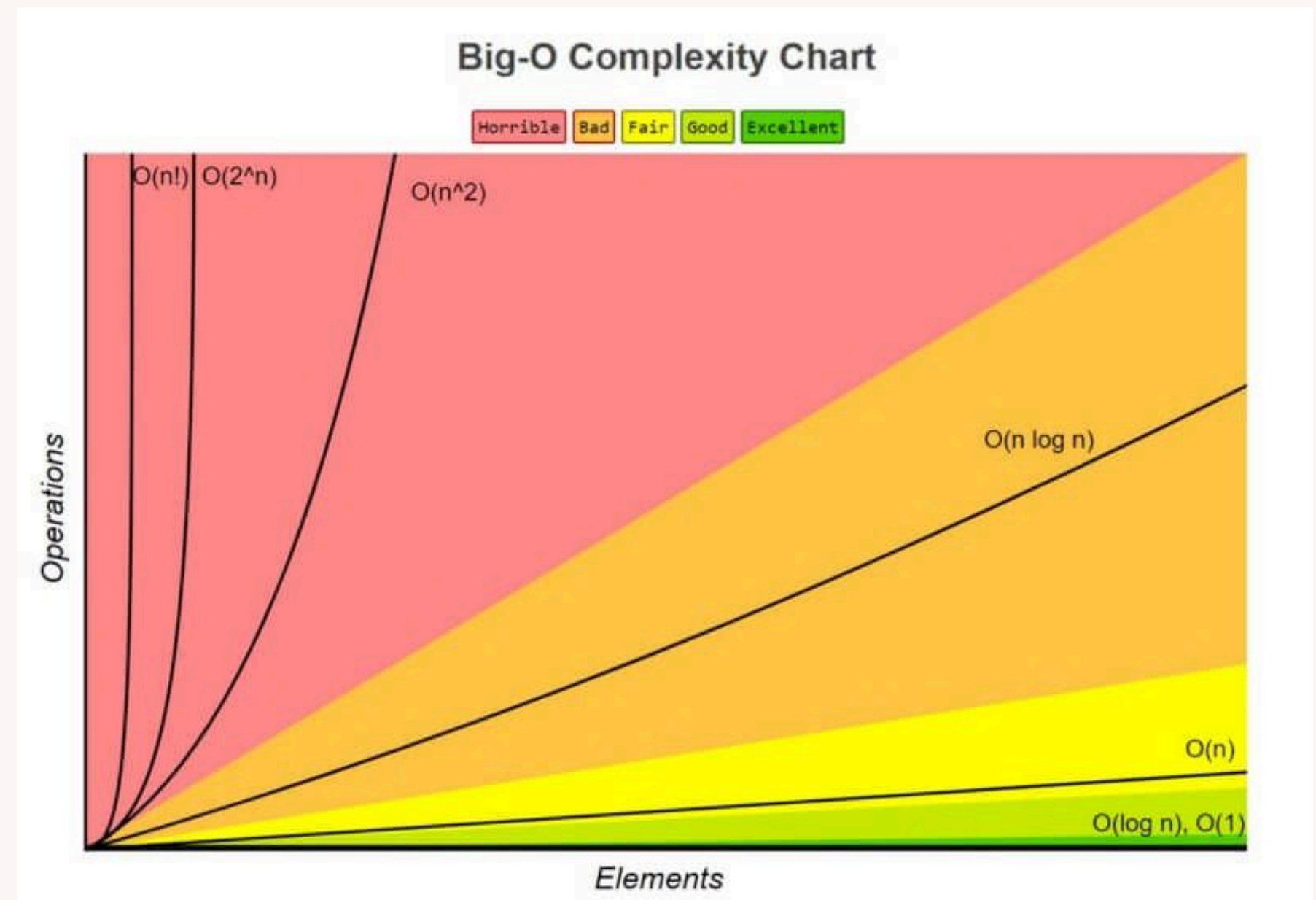
Ex: um algoritmo que executa $4n$ passos é $O(n)$.



FACILITA AS CONTAS

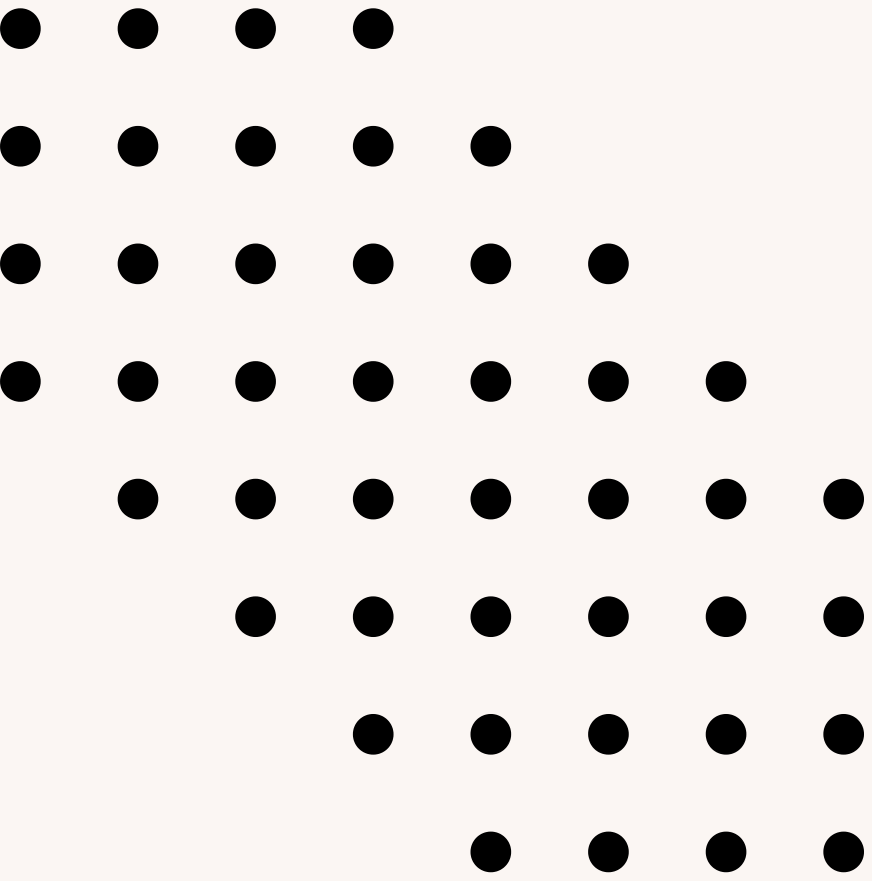
Se um algoritmo é $O(n)$ podemos assumir que ele executa n passos.

Perdemos precisao da estimativa ao fazer isso, cuidado!



CLASSES COMUNS

n	Worst AC Algorithm	Comment
$\leq [10..11]$	$O(n!), O(n^6)$	e.g., Enumerating permutations (Section 3.2)
$\leq [17..19]$	$O(2^n \times n^2)$	e.g., DP TSP (Section 3.5.2)
$\leq [18..22]$	$O(2^n \times n)$	e.g., DP with bitmask technique (Book 2)
$\leq [24..26]$	$O(2^n)$	e.g., try 2^n possibilities with $O(1)$ check each
≤ 100	$O(n^4)$	e.g., DP with 3 dimensions + $O(n)$ loop, ${}_nC_{k=4}$
≤ 450	$O(n^3)$	e.g., Floyd-Warshall (Section 4.5)
$\leq 1.5K$	$O(n^{2.5})$	e.g., Hopcroft-Karp (Book 2)
$\leq 2.5K$	$O(n^2 \log n)$	e.g., 2-nested loops + a tree-related DS (Section 2.3)
$\leq 10K$	$O(n^2)$	e.g., Bubble/Selection/Insertion Sort (Section 2.2)
$\leq 200K$	$O(n^{1.5})$	e.g., Square Root Decomposition (Book 2)
$\leq 4.5M$	$O(n \log n)$	e.g., Merge Sort (Section 2.2)
$\leq 10M$	$O(n \log \log n)$	e.g., Sieve of Eratosthenes (Book 2)
$\leq 100M$	$O(n), O(\log n), O(1)$	Most contest problem have $n \leq 1M$ (I/O bottleneck)



FIM