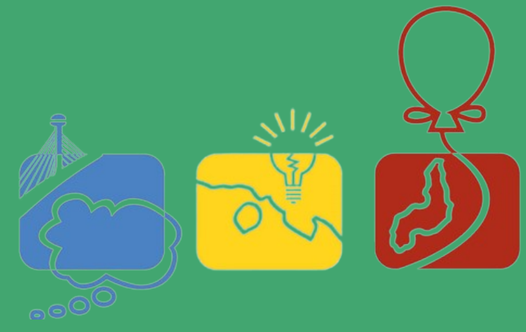
COMPLEXIDADE DE ALGORITMOS

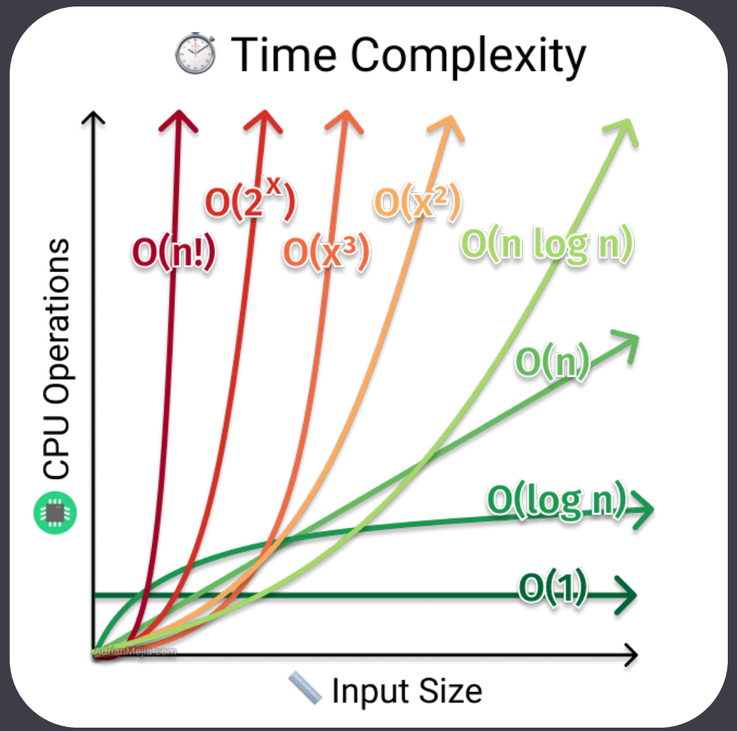
PCPI - AULA 2



Notação Big O

Para compararmos a velocidade dos nossos algoritos e determinar o quanto eles são rápidos, utilizamos a notação Big O.

Nós definimos a complexidade pela forma com que a quantidade de operações feitas pelo nosso programa aumenta em função do tamanho da entrada.



Notação Big O

Por exemplo, a complexidade do código abaixo é O(N), pois dependendo da entrada N o código executa uma quantidade de operações diferentes:

```
int N; cin >> N;
for(int i=0; i<N; i++)
    contador += 1;
```

Já esse outro código tem complexidade O(N^2) :

```
int N; cin >> N;
for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
      contador += 1;
```

Qual a complexidade desse código?

```
int N; cin >> N;

for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
    for(int k=0; k<N; k++)
      contador += 1;
```

Qual a complexidade desse código?

```
int N; cin >> N;

for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
    for(int k=0; k<N; k++)
      contador += 1;
```

O(N ^ 3)

Qual a complexidade desse código?

```
int N; cin >> N;

for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
    for(int k=0; k<N; k++)
      contador += 1;

for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
    contador += 1
```

Qual a complexidade desse código?

Sendo mais preciso, a complexidade seria:
O(N^3 + N^2)
Mas como N^3 é maior que N^2, consideramos só o maior para a complexidade

```
int N; cin >> N;

for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
    for(int k=0; k<N; k++)
      contador += 1;

for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
    contador += 1
```

O(N ^ 3)

Qual a complexidade desse código?

```
int N;
cin >> N;


cout << N / 2 << endl;
```

Qual a complexidade desse código?

```cpp
int N;
cin >> N;

cout << N / 2 << endl;
```

O(2) ???

Qual a complexidade desse código?

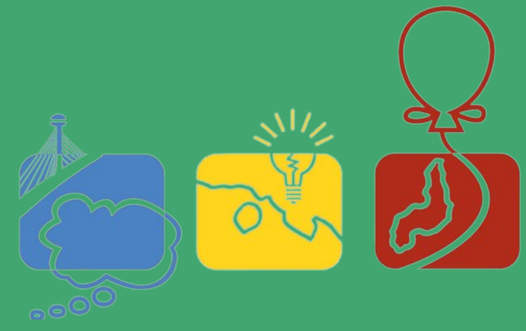
```cpp
int N;
cin >> N;

cout << N / 2 << endl;
```

Quando a quantidade de operações não depende do tamanho do input, dizemos que o código é constante, e dizemos que a complexidade é O(1) (mesmo que gaste mais do que uma operação)

O(1)

BIBLIOTECA STL C++

PCPI - AULA 2



STL - Standard Template Library

No C++ existe a biblioteca padrão STL, que já possui uma série de ***estruturas de dados**** e de ***algoritmos*** muito úteis já implementados que nós podemos usar em nossos códigos.

Vamos ver aqui alguns dos que mais vamos utilizar na programação competitiva.

** Estrutura de dados: podemos pensar em uma estrutura de dados como uma forma de organizar dados. Isso inclui como eles estão armazenados, como acessamos, como adicionamos e como removemos eles da estrutura*

Sintaxe das Estruturas do STL

estrutura<tipo> nome;

Primeiro informamos **qual estrutura** queremos utilizar.

Uma estrutura de dados é uma forma de “organizar” nossos dados, então devemos informar o **tipo que nossa estrutura vai armazenar**.

Precisamos informar um **nome para identificar** nossa estrutura.

E opcionalmente podemos ainda passar alguns parâmetros para inicializar nossa estrutura, assim: estrutura<tipo> nome (N);

Funções Size e Empty

```
estrutura.size();
estrutura.empty();
```

A maioria das estruturas possui as funções size( ) e empty( ). Elas são bem simples:

- **size:** retorna o tamanho da estrutura, quantos elementos ela está guardando.
- **empty:** retorna se a estrutura está vazia ou não, é equivalente a estrura.size( ) == 0.

Vector

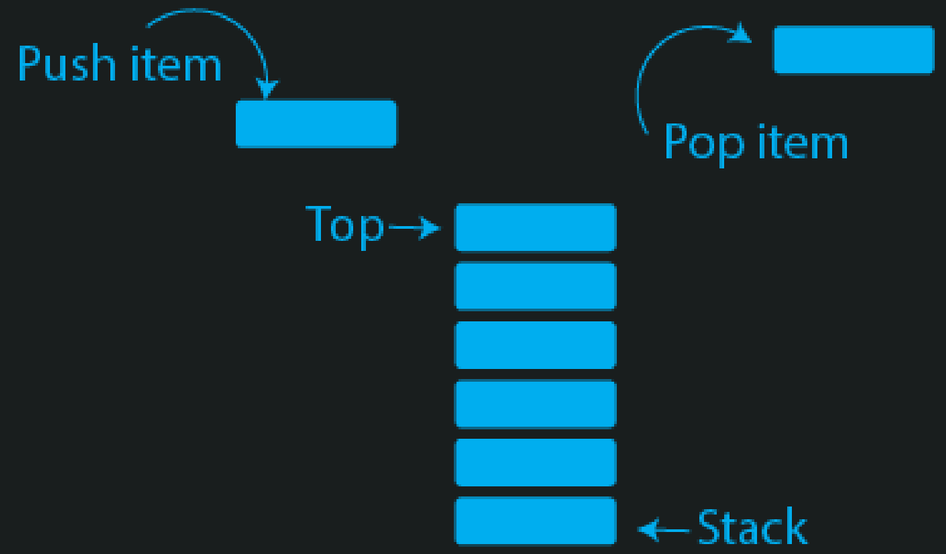
O vector nada mais é que um **Array Dinâmico**! Isto é, ele não tem um tamanho fixo, podemos adicionar e remover elementos dele a qualquer momento!

```cpp
// Cria um vector vazio
vector<int> lista;

// Adiciona um elemento no final do vector
lista.push_back(10);
lista.push_back(15);

// Remove o elemento no final do vector
lista.pop_back();

// Cria um vector com tamanho 10
// e com todas as posições iguais a -1
vector<int> lista2 (10, -1);

// Podemos acessar o Vector como um array
lista2[0] = 5;
lista2[9] = 20;
```

Stack

A stack, ou Pilha, organiza os dados de forma que só podemos ver o elemento que está no “topo” da pilha, além disso só podemos remover e adicionar elementos no topo da pilha.

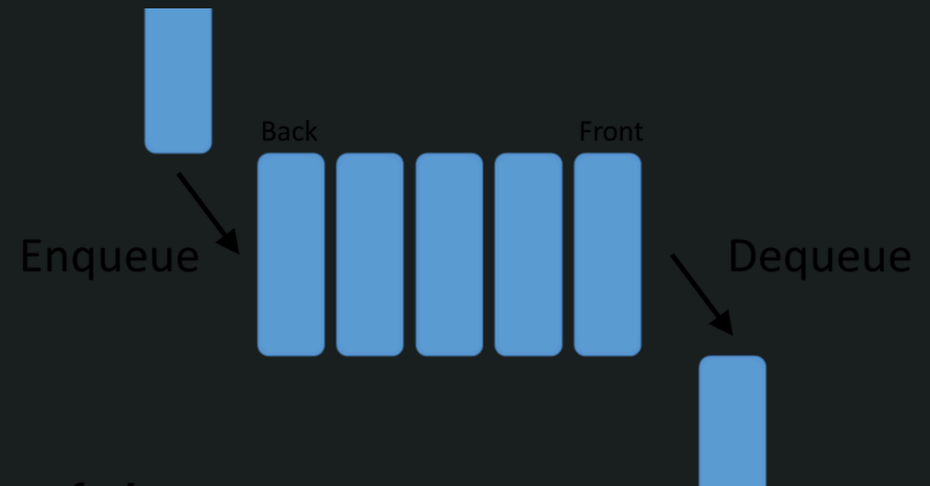
```
stack<int> pilha;

// Adiciona um elemento ao TOPO da pilha
pilha.push(5);
pilha.push(2);

// Remove o elemento no TOPO da pilha
pilha.pop();

// Olha o elemento no TOPO da pilha
int topo = pilha.top();
```



Queue

A queue, ou fila, funciona exatamente como uma fila, você consegue adicionar elementos no FINAL da fila e remover o elemento na FRENTE da fila. Você também pode ver o elemento na frente.

```cpp
queue<int> fila;

// Adiciona um elemento no FINAL da fila
fila.push(12);
fila.push(4);
fila.push(15);

// Remove o elemento da FRENTE da fila
fila.pop();

// Vê o elemento que está na FRENTE da fila
int frente = fila.front();
```



Priority Queue

A Fila de Prioridades é uma estrutura em que você pode adicionar elementos nela e ela vai sempre manter o elemento de maior prioridade no topo/frente da fila (geralmente o maior).

```cpp
priority_queue<int> prioridade;

// Adiciona um elemento na fila de prioridade
prioridade.push(8);
prioridade.push(15);
prioridade.push(2);

// Remove o elemento da FRENTE da fila (o de maior prioridade)
prioridade.pop();

// Vê o elemento que está na FRENTE da fila (o de maior prioridade)
int maior = prioridade.top();
```

Set

O set, ou conjunto, é uma estrutura que mantém os elementos ordenados. Você pode adicionar ou remover elementos do conjunto, e também verificar se um elemento existe no conjunto.

obs: não mantém elementos repetidos

```cpp
set<int> conjunto;

// Insere um elemento no conjunto
conjunto.insert(10);
conjunto.insert(4);
conjunto.insert(4);

// Verifica se um elemento está no conjunto
bool esta5 = conjunto.count(5);
bool esta4 = conjunto.find(4) != conjunto.end();

// Apaga um elemento do conjunto
conjunto.erase(4);
```

Map

O Map é uma estrutura poderosa, ele consiste em um par com uma chave e um conteúdo. Podemos acessar ele de forma semelhante a um vector, mas através da chave (apesar de não ser em nada parecido com um).

```
map<string, int> mapa;

mapa["Little Dice"] = 24;
mapa["SH12"] = 19;
mapa["Dann"] = 35;
mapa["IFPI"] = 114;
```

Pair

O pair, ou par, é uma estrutura bem simples, ele simplesmente guarda dois valores. Podemos acessar o primeiro e o segundo como variáveis.

```
pair<int, int>  par = {8, 7};
pair<int, string> p = {8, "txt"};

par.first = 4;
par.second += 2;
```

Deque

A estrutura deque é como um vector, mas aceita além das operações de push_back e pop_back as operações de push_front e pop_front.

```
deque<int> dq;

// Adiciona elementos no fim ou no início
dq.push_back(2);
dq.push_front(1);
dq.push_back(3);
dq.push_front(0);

// Remove elementos no fim ou no início
dq.pop_back();
dq.pop_front();

// Pode ser acessado como um array
dq[0] = 12;
dq[1] = 27;
```

Sort()

O sort é uma função que ordena um vector em ordem crescente. Você precisa passar o início e o fim do vetor (begin e end).

```
vector<int> lista = {4, 6, 2, 5, 1};

sort(begin(lista), end(lista));

for(int i=0; i<lista.size(); i++)
        cout << lista[i] << " ";

// saída:
// 1 2 4 5 6
```

Iterar por um vector, set ou map

Existe um iterador automático para esses estruturas:

```cpp
vector<int> lista;

for(auto x : lista)
        cout << x << " ";
cout << endl;

set<int> conj;

for(auto x: conj)
        cout << x << " ";
cout << endl;

map<string, int> mapa;

for(auto [chave, valor] : mapa)
        cout << chave << " " << valor << end
```

That’s all, Folks!