



Clase 1: Librería Estándar

25 de marzo

¿Que es la librería estándar?

Es la librería que es parte de la especificación de C++ y está incluida en sus compiladores.

¿Que es la librería estándar?

Es la librería que es parte de la especificación de C++ y está incluida en sus compiladores.

Una herramienta superútil al momento de resolver problemas en programación competitiva.

¿Como se utiliza?

Se puede importar individualmente, o con la línea mágica vista en la primera clase:

```
#include <bits/stdc++.h>  
using namespace std;
```

Motivación

Queremos evitar los veredictos de TLEs que quizás obtuvieron la dos últimas semanas.

Motivación

Queremos evitar los veredictos de TLEs que quizás obtuvieron la dos últimas semanas.

Las estructuras de la librería estándar nos permiten hacer ciertas operaciones con una **complejidad** reducida.

¿Qué es la complejidad? ¿Y de qué nos sirve 🤔?

Es general es una aproximación del tiempo que se demora un algoritmo en finalizar en función del tamaño de su entrada.

¿Qué es la complejidad? ¿Y de qué nos sirve 🤔?

Es general es una aproximación del tiempo que se demora un algoritmo en finalizar en función del tamaño de su entrada.

$$O(n), O(n^2), O(n \log n)$$

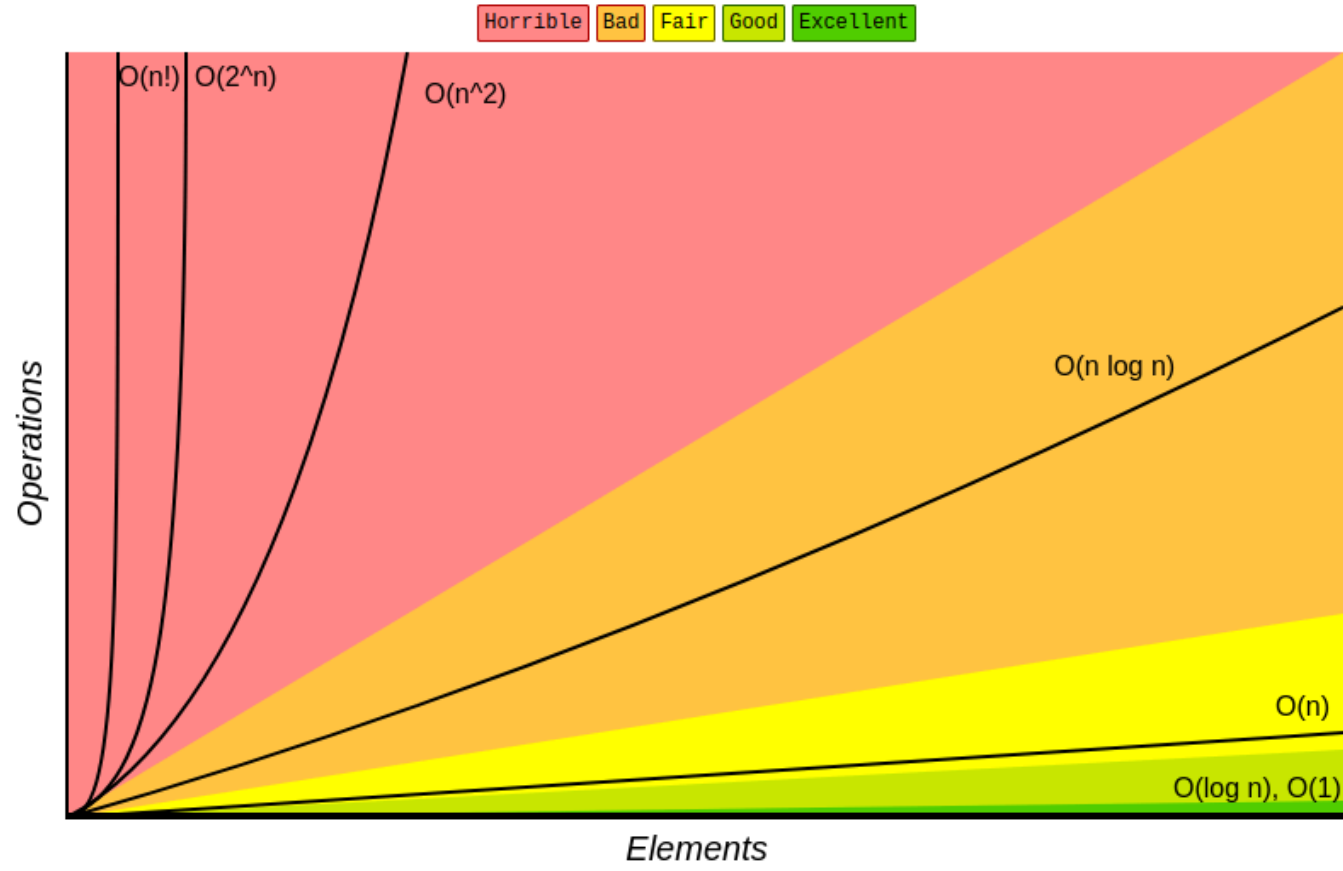
¿Qué es la complejidad? ¿Y de qué nos sirve 🤔?

Es general es una aproximación del tiempo que se demora un algoritmo en finalizar en función del tamaño de su entrada.

$$O(n), O(n^2), O(n \log n)$$

En la programación competitiva evitamos escribir soluciones recursivas, se buscan soluciones iterativas: estas son más fáciles de calcular.

Big-O Complexity Chart



```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < n; j++) {  
        a++;  
    }  
}
```

```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < n; j++) {  
        a++;  
    }  
}
```

$$n \cdot n \implies O(n^2)$$

```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < i; j++) {  
        a++;  
    }  
}
```

```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < i; j++) {  
        a++;  
    }  
}
```

$$\sum_{i=0}^n i = \frac{n \cdot (n + 1)}{2} \implies O(n^2)$$

```
for (int i = 1; i < n; i++) {  
    for (int j = 0; j < n; j += i) {  
        a++;  
    }  
}
```

```
for (int i = 1; i < n; i++) {  
    for (int j = 0; j < n; j += i) {  
        a++;  
    }  
}
```

$$\sum_{i=1}^n \frac{n}{i}$$



FROM THE MAKERS OF WOLFRAM LANGUAGE AND MATHEMATICA



Enter what you want to calculate



NATURAL LANGUAGE





MATH INPUT





$$\sum_{i=1}^n \frac{n}{i}$$

 NATURAL LANGUAGE

 MATH INPUT

Sum

$$\sum_{i=1}^n \frac{n}{i} = n H_n$$



Growth rate [\[edit \]](#)

These numbers grow very slowly, with [logarithmic growth](#), as can be seen from the integral test.^[15] More precisely, by the [Euler-Maclaurin formula](#),

$$H_n = \ln n + \gamma + \frac{1}{2n} - \varepsilon_n$$

where $\gamma \approx 0.5772$ is the [Euler-Mascheroni constant](#) and $0 \leq \varepsilon_n \leq 1/8n^2$ which approaches 0 as n goes to infinity.^[16]



$$O(n \log n)$$

Ahora si que si.

Los problemas que entregamos tienen información clave que les debería ayudar a concluir si su solución pasa:

Ahora si que si.

Los problemas que entregamos tienen información clave que les debería ayudar a concluir si su solución pasa:

Time Limit

Ahora si que si.

Los problemas que entregamos tienen información clave que les debería ayudar a concluir si su solución pasa:

Time Limit

En programación competitiva se asume que en general los computadores se demoran 1 segundo en realizar 10^8 operaciones.

$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n) \text{ o } O(n)$

¿Qué tiene la librería estándar?

Estructuras

- Vectores
- Sets y Mapas
- Hashsets y Hashmaps
- Queues y Stacks
- Priority Queues

Algoritmos

- `sort()`, `find()`, `lower_bound()`,

Vector

Es un arreglo dinámico.

```
vector<int> vec(n) // crea un vector en  $O(n)$   
vec[i] = 0 // accede en  $O(1)$   
vec.push_back(2) // agrega un elem. al final en  $O(1)$  amort.  
vec.pop_back() // elimina el ultimo elem. en  $O(1)$   
vec.erase(i) // elimina el valor en i en  $O(n)$   
vec.assign(m, 0) // reemplaza m valores con 0 en  $O(m)$ 
```

Iteradores

Algunas estructuras de la librería estándar tienen una herramienta útil que son iteradores. Apuntan a algún elemento del contenedor:

Iteradores

Algunas estructuras de la librería estándar tienen una herramienta útil que son iteradores. Apuntan a algún elemento del contenedor:

```
auto it = vec.begin() // retorna un iterador al inicio O(1)
it = vec.end() // retorna un iterador pasado el final O(1)
auto x = *it // lee el elemento en it en O(1)
```

Iteradores

Algunas estructuras de la librería estándar tienen una herramienta útil que son iteradores. Apuntan a algún elemento del contenedor:

```
auto it = vec.begin() // retorna un iterador al inicio O(1)
it = vec.end() // retorna un iterador pasado el final O(1)
auto x = *it // lee el elemento en it en O(1)
```

Estos iteradores se pueden operar, como por ejemplo:

```
it++; it--; vec.begin() + i;
```

Iteradores

Algunas estructuras de la librería estándar tienen una herramienta útil que son iteradores. Apuntan a algún elemento del contenedor:

```
auto it = vec.begin() // retorna un iterador al inicio 0(1)
it = vec.end() // retorna un iterador pasado el final 0(1)
auto x = *it // lee el elemento en it en 0(1)
```

Estos iteradores se pueden operar, como por ejemplo:

```
it++; it--; vec.begin() + i;
```

Cuidado, si los miras feo te van a dar compilation error.

Iteradores

Estos iteradores se pueden utilizar como argumentos a otras funciones:

Iteradores

Estos iteradores se pueden utilizar como argumentos a otras funciones:

```
sort(vec.begin(), vec.end()) // lo ordena en  $O(n \log n)$   
reverse(vec.begin(), vec.end()) // lo da vuelta en  $O(n)$   
lower_bound(vec.begin(), vec.end(), x) // obtiene un  
iterador a el primer valor que *no* es menor a x en  $O(\log n)$ 
```

String



String



String



Los string son basicamente:

```
string = vector<char>;
```

String



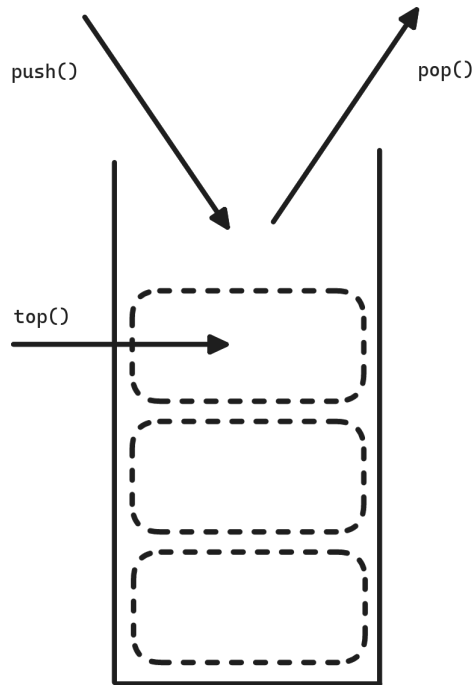
Los string son basicamente:

```
string = vector<char>;
```

Pero tiene un par de cosas adicionales:

```
s = "ho!a" + "mundo"; // sumar en 0(n)  
cout << s; // imprimir en 0(n)  
cin >> s; // ingresar en 0(n)
```

Stack



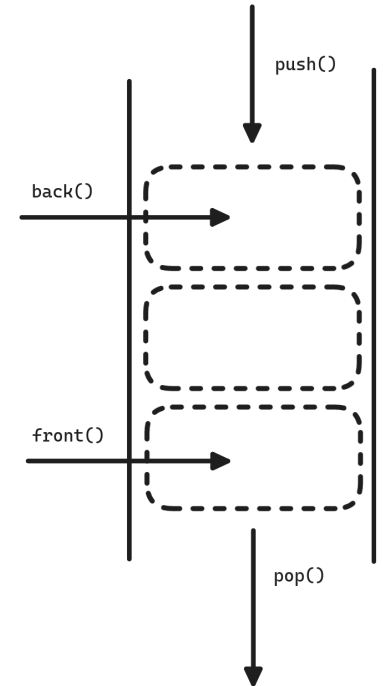
La libreria estandar contiene un stack que se puede utilizar.

Sin embargo un vector es equivalente.

Queue

La librería estandar tiene un queue.

```
queue<int> q;  
s.push(42) // agrega un elem. en O(1)  
s.front() // lee el elem. del frente en O(1)  
s.back() // lee el elem. de atrás en O(1)  
s.pop() // elimina el elem. del frente en O(1)
```



Set

Es un conjunto ordenado sin elementos duplicados.

```
set<int> s;  
s.insert(42) // agrega un elemento en  $O(\log n)$   
*s.begin() // obtiene el elemento mas pequeño en  $O(1)$   
auto it = s.find(42) // retorna un it a elem. en  $O(\log n)$   
s.erase(42) // elimina un elemento en  $O(\log n)$ 
```

También tiene iteradores, pero son más limitados, no permite hacer operaciones como `it + i`.

Map

Un mapeo entre [llave, valor], que se encuentra ordenado por las llaves.

```
map<int, int> m;  
m[42] = 7; // asigna un valor a una llave en  $O(\log n)$   
m.erase(42); // elimina una llave y valor en  $O(\log n)$ 
```

Priority Queue

Un queue que retorna el elemento más grande (o más pequeño) ingresado hasta ahora.

Priority Queue

Un queue que retorna el elemento más grande (o más pequeño) ingresado hasta ahora.

Se puede hacer lo mismo con un set a la misma complejidad, pero está implementado con estructura subyacente más eficiente.

Priority Queue

Un queue que retorna el elemento más grande (o más pequeño) ingresado hasta ahora.

Se puede hacer lo mismo con un set a la misma complejidad, pero está implementado con estructura subyacente más eficiente.

```
priority_queue<int> q;  
q.push(7) // inserta el elemento en  $O(\log n)$   
q.top() // obtiene el elemento mas grande en  $O(1)$   
q.pop() // elimina el elemento mas grande en  $O(\log n)$ 
```

Priority Queue

Si quieren obtener el elemento más pequeño cuando hacen `top()` pueden hacer el truco prohibido de:

```
q.push(-7)
```

Priority Queue

Si quieren obtener el elemento más pequeño cuando hacen `top()` pueden hacer el truco prohibido de:

~~`q.push(7)`~~

O pueden hacerlo de la forma correcta:

Priority Queue

Si quieren obtener el elemento más pequeño cuando hacen `top()` pueden hacer el truco prohibido de:

~~`q.push(7)`~~

O pueden hacerlo de la forma correcta:

```
priority_queue<int, vector<int>, greater<int>> q;
```

Unordered Set/Map

Tienen la misma interfaz que los set y map, pero sus operaciones toman $O(1)$, ya que están implementados con hashing.

Unordered Set/Map

Tienen la misma interfaz que los set y map, pero sus operaciones toman $O(1)$, ya que están implementados con hashing.

Pero la implementación de la librería estándar es mediocre.

- Tiene una constante alta.
- Un adversario puede crear input que explota la complejidad.

Unordered Set/Map

Tienen la misma interfaz que los set y map, pero sus operaciones toman $O(1)$, ya que están implementados con hashing.

Pero la implementación de la librería estándar es mediocre.

- Tiene una constante alta.
- Un adversario puede crear input que explota la complejidad.

Úsenlo con **cuidado**.

Gracias!