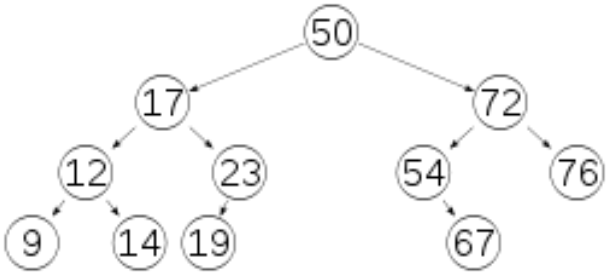
www.agit.com.br

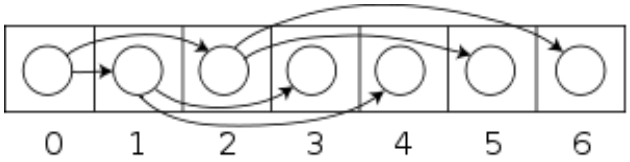
Curso

# Algoritmos com C++

## Fundamentos e Prática para Soluções de Problemas

*por Fabio Galuppo*




# Algoritmos com C++

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com

http://simplycpp.com/

http://github.com/fabiogaluppo

fabiogaluppo@acm.org

@FabioGaluppo

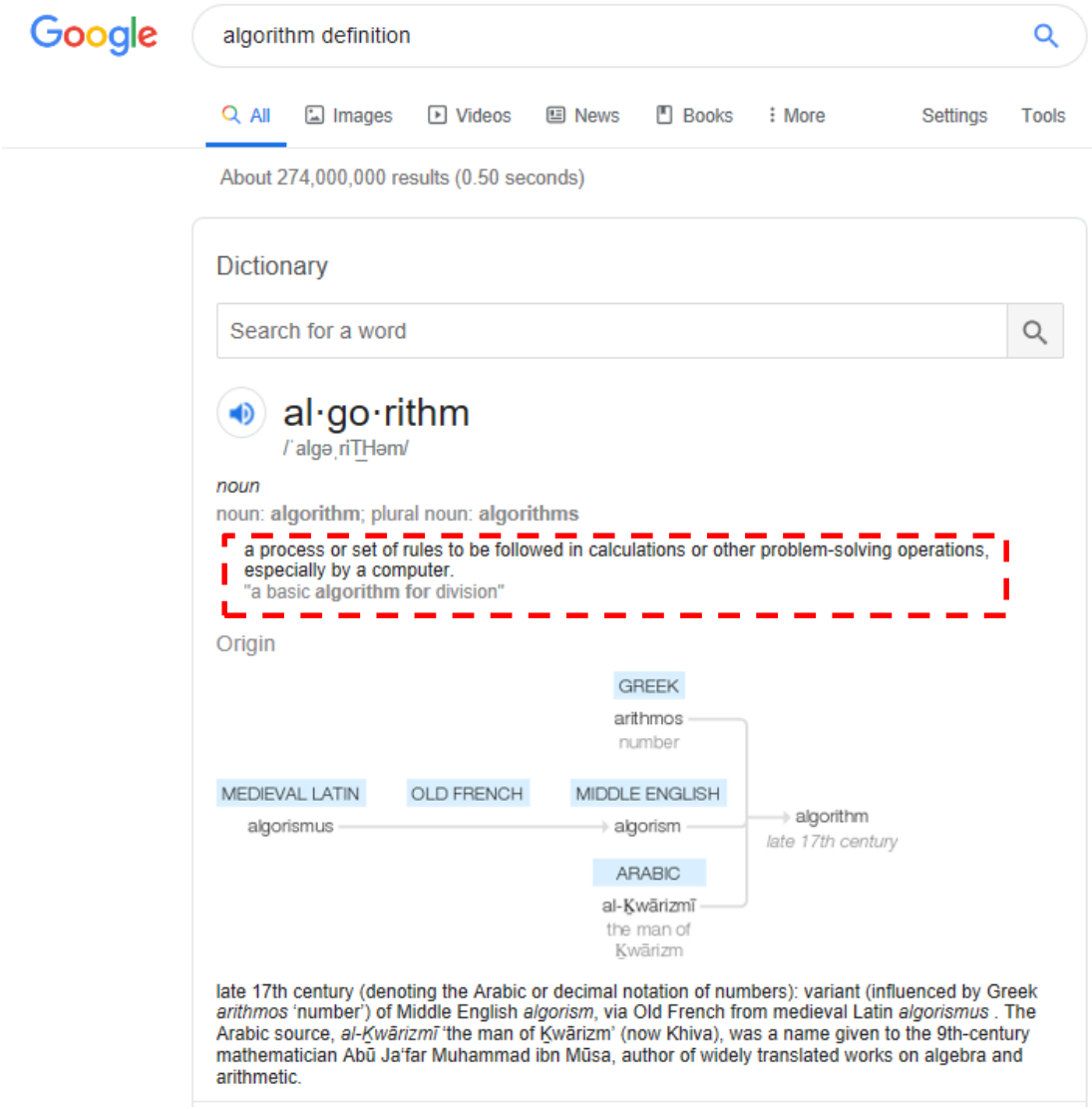
# Algoritmo

## Wikipedia


https://en.wikipedia.org/wiki/Algorithm

# Algoritmo

## Google



# Algoritmo

Jeff Erickson

## 0.1 What is an algorithm?

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose. For example, here is an algorithm for singing that annoying song "99 Bottles of Beer on the Wall", for arbitrary values of 99:

```
BOTTLESOFBEER(n):
  For i ← n down to 1
      Sing "i bottles of beer on the wall, i bottles of beer,"
      Sing "Take one down, pass it around, i − 1 bottles of beer on the wall."
  Sing "No bottles of beer on the wall, no bottles of beer,"
  Sing "Go to the store, buy some more, n bottles of beer on the wall."
```

http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf

# Algoritmo

## CLRS

### 1.1 Algorithms

Informally, an *algorithm* is any well-defined computational procedure that takes some value, or set of values, as *input* and produces some value, or set of values, as *output* in a finite amount of time. An algorithm is thus a sequence of computational steps that transform the input into the output.

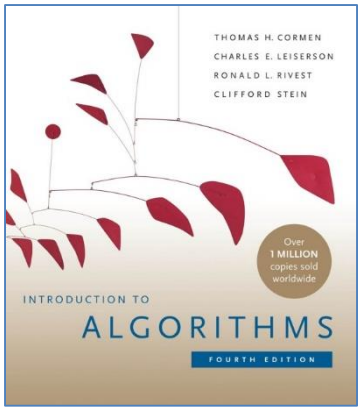
You can also view an algorithm as a tool for solving a well-specified *computational problem*. The statement of the problem specifies in general terms the desired input/output relationship for problem instances, typically of arbitrarily large size. The algorithm describes a specific computational procedure for achieving that input/output relationship for all problem instances.

As an example, suppose that you need to sort a sequence of numbers into monotonically increasing order. This problem arises frequently in practice and provides fertile ground for introducing many standard design techniques and analysis tools. Here is how we formally define the *sorting problem*:

**Input:** A sequence of $n$ numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

**Output:** A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Thus, given the input sequence $\langle 31, 41, 59, 26, 41, 58 \rangle$, a correct sorting algorithm returns as output the sequence $\langle 26, 31, 41, 41, 58, 59 \rangle$. Such an input sequence is




https://www.google.com.br/books/edition/Introduction_to_Algorithms_fourth_editio/RSMuEAAAQBAJ?hl=en&gbpv=1&pg=PA5

# Algoritmo

*wrap-up*

- Conjunto finito de regras explícitas
  - Instruções não ambíguas e descritas com precisão
- Aplicado na resolução de problemas
  - Uma pergunta ao qual se busca uma resposta
- Executado por computadores
  - Preferencialmente
- Independente de linguagem de programação
- Ser eficiente e ter utilidade

# Estrutura de Dados

## Wikipedia


https://en.wikipedia.org/wiki/Data_structure

# Estrutura de Dados

## Britannica, The Free Dictionary, Pat Morin

**Data structure**, way in which data are stored for efficient search and retrieval. Different data structures are suited for different problems. Some data structures are useful for simple general problems, such as retrieving data that has been stored with a specific identifier. For example, an online dictionary can be structured so that it can retrieve the definition of a word. On the other hand, specialized data structures have been devised to solve complex specific search problems. https://www.britannica.com/technology/data-structure

data structure

*n*

(Computer Science) an organized form, such as an array list or string, in which connected data items are held in a computer https://www.thefreedictionary.com/data+structure

Every computer science curriculum in the world includes a course on data structures and algorithms. Data structures are *that* important; they improve our quality of life and even save lives on a regular basis. Many multi-million and several multi-billion dollar companies have been built around data structures.

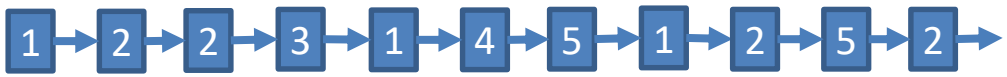
https://opendatastructures.org/ods-cpp.pdf

# Estrutura de Dados

*wrap-up*

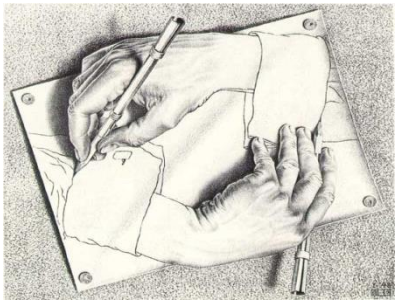
- Organizar dados na memória de um computador
  - *Arrays*, listas, *streams*, tabelas, árvores, grafos, …
- Endereços (índices) e ponteiros
  - Iteradores
- Operações
  - Interface
    - Comportamento e garantias (desempenho)
  - Implementação
- Ser eficiente e ter utilidade

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 3 | 1 | 4 | 5 | 1 | 2 | 5 | 2 |

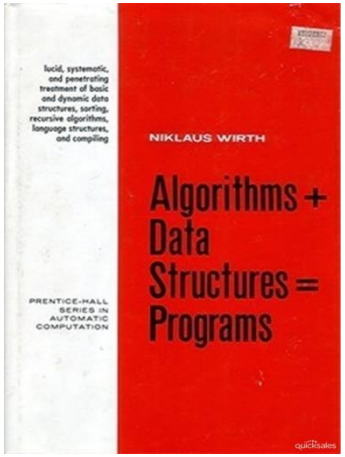


# Algoritmos e Estrutura de Dados

- Uma simbiose artificial
  - Algoritmo em termos de uma estrutura de dados
    - Ex.: *Heapsort* (*heap* usada para ordenação)
  - Estrutura de dados em termos de um algoritmo
    - Ex.: Encontrar a posição para inserir um item na lista



Drawing Hands – M.C. Escher




Niklaus Wirth

- Programas = Algoritmos + Estrutura de Dados
  - São rodados pelo computador para executar tarefas específicas

# Algoritmos e Programação

## Principais diferenças

### Algoritmo

- *Design*
- *Domain Knowledge/Expert*
- Pseudocódigo
  - Texto e Notação Matemática
- Independente
  - *Hardware*
    - Máquina hipotética
  - *Software*
    - SO, Compilador, …
- Análise
  - Em função do tempo ou espaço
  - Teoria

### Programação

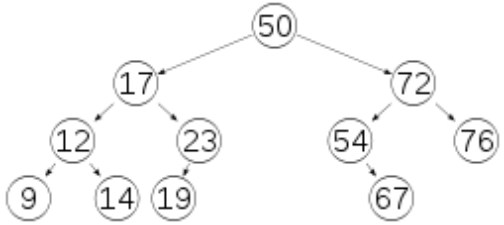
- Implementação
- Programador
- Linguagem de Programação
  - C++, …
- Dependente
  - *Hardware*
    - Máquina real
  - *Software*
    - SO, Compilador, …
- *Benchmarking*
  - Em função do tempo ou espaço
    - custo e energia são outras opções
  - Prática

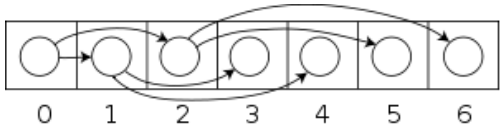
# Algoritmos com C++

## *Design, Analysis and Implementation*

- Este curso, focado em:
  - *Design*
    - *Problem solving*
  - *Analysis*
    - *Classical*
      - *Analysis of Algorithms*
    - *Empirical*
      - *Benchmarking*
  - *Implementation*
    - C++
      - *Raw pointers*
      - Programação Estruturada
      - C com classes (OO)
      - Programação Genérica
        - » *Templates*
        - » STL
      - *Modern* C++
        - » *Smart pointers*

Um curso tradicional de Algoritmos com pseudocódigo é focado nos aspectos em destaque



# Procurar ‘x’ num *array* não-ordenado

```
static std::size_t find_first_char(const char* xs, std::size_t n, char x)
{
    for (std::size_t i = 0; i < n; ++i)
        if (xs[i] == x)
            return i;
    return NOT_FOUND;
}
```

```
const char* xs = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
std::size_t pos_M = find_first_char(xs, 26, 'M'); //12
std::size_t pos_$ = find_first_char(xs, 26, '$'); //NOT_FOUND
```

---

```
static std::size_t find_first_int(const int* xs, std::size_t n, int x)
{
    for (std::size_t i = 0; i < n; ++i)
        if (xs[i] == x)
            return i;
    return NOT_FOUND;
}
```

```
const int ys[]{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
std::size_t pos_x5 = find_first_int(ys, 10, 0x5); //5
std::size_t pos_xA = find_first_int(ys, 10, 0xA); //NOT_FOUND
```

# C++ Templates

- Mecanismo do C++ para tipos parametrizáveis
  - Polimorfismo paramétrico
    - Tipo (classe) ou função
  - Preservar genericamente tipagem estática
  - Substituição do tipo genérico para o tipo específico
    - Instância do *template*
  - Especialização de *template*

```
const char* xs = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
std::size_t pos_M = find_first<char>(xs, 26, 'M'); //12
std::size_t pos_$ = find_first<char>(xs, 26, '$'); //NOT_FOUND

const int ys[]{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
std::size_t pos_x5 = find_first<int>(ys, 10, 0x5); //5
std::size_t pos_xA = find_first<int>(ys, 10, 0xA); //NOT_FOUND
```

# Procurar ‘x’ num *array* não-ordenado

## Genérica

```
template <typename T>
static std::size_t find_first(const T* xs, std::size_t n, const T& x)
{
    for (std::size_t i = 0; i < n; ++i)
        if (xs[i] == x)
            return i;
    return NOT_FOUND;
}
```

```
const char* xs = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
std::size_t pos_M = find_first<char>(xs, 26, 'M'); //12
std::size_t pos_$ = find_first<char>(xs, 26, '$'); //NOT_FOUND
```

---

```
const int ys[]{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
std::size_t pos_x5 = find_first<int>(ys, 10, 0x5); //5
std::size_t pos_xA = find_first<int>(ys, 10, 0xA); //NOT_FOUND
```

---

```
const bool zs[]{ false, false, false, false, true };
std::size_t pos_t1 = find_first<bool>(zs, 5, true); //4
std::size_t pos_t2 = find_first<bool>(zs, 4, true); //NOT_FOUND
```

# Estruturas de Controle e Algoritmos

- Todos os algoritmos (imperativos) executam seus passos através da combinação de três estruturas de controle, são elas:
  - Sequência, Seleção e Iteração

```
template <typename T>
static std::size_t find_first(const T* xs, std::size_t n, const T& x)
{                          1.1    1.2    1.3
  1. for (std::size_t i = 0; i < n; ++i)
  2.       if (xs[i] == x)
               return i;
  3. return NOT_FOUND;
}
```

- Os passos são colocados em sequência;
- O passo 1, é uma iteração. Na primeira execução desse passo, a variável **i** é iniciada com **0** em 1.1. Em 1.2, temos uma seleção, onde é verificado se o valor atual de **i** é menor do que **n**, permitindo o avanço para o passo 2. Caso contrário, pula para o passo 3. Toda vez que retornar ao passo 1, a variável **i** é incrementada e o passo 1.1 é ignorado;
- O passo 2, é uma seleção. É comparado se o valor do vetor na posição **i** é igual ao valor de **x**. Se sim, retorna o indíce **i** e o algoritmo é encerrado. Senão, retorna ao passo 1;
- O passo 3, retorna que o valor de **x** não foi encontrado, encerrado a execução do algoritmo.

# Programação Genérica

- *Generic Programming*
  - 1988 (David R. Musser e Alexander A. Stepanov)
- *"Abstracting from concrete, efficient algorithms to obtain generic algorithms"*

Generic programming centers around the idea of abstracting from concrete, efficient algorithms to obtain generic algorithms that can be combined with different data representations to produce a wide variety of useful software. For example, a class of generic sorting algorithms can be defined which work with finite sequences but which can be instantiated in different ways to produce algorithms working on arrays or linked lists.

http://stepanovpapers.com/genprog.pdf

https://link.springer.com/book/10.1007/3-540-51084-2

# Programação Genérica

- *Generic Programming is about abstracting and classifying algorithms and data structures*

> STL is only a limited success. While it became a widely used library, its central intuition did not get across. People confuse generic programming with using (and abusing) C++ templates. Generic programming is about abstracting and classifying algorithms and data structures. It gets its inspiration from Knuth and not from type theory. Its goal is the incremental construction of systematic catalogs of useful, efficient and abstract algorithms and data structures. Such an undertaking is still a dream.
>
> http://stepanovpapers.com/history%20of%20STL.pdf

- Exemplos de algoritmos de ordenação na STL:
  - std::sort
  - std::partial_sort
  - std::stable_sort
  - std::list::sort

# C++ Padrão e Algoritmos

- Standard Template Library (STL)
  - Algoritmos, *Containers*, *Iterators*
- `#include <algorithm>`


Table 100 — Algorithms library summary

| Subclause | | Header(s) |
|---|---|---|
| 28.5 | Non-modifying sequence operations | |
| 28.6 | Mutating sequence operations | `<algorithm>` |
| 28.7 | Sorting and related operations | |
| 28.8 | C library algorithms | `<cstdlib>` |

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf

http://www.cplusplus.com/reference/algorithm/

# Algoritmo e Memória

(versão imperativa)

- *Index-based*
- *Contiguous, Random Access*
  - `std::vector<T>`
- Contar a incidência de um `int` num `std::vector<int>`

```
size_t count_occurrences(const std::vector<int>& xs, int target)
{
    size_t counter = 0;
    for (size_t i = 0; i < xs.size(); ++i)
        if (xs[i] == target)
            ++counter;
    return counter;
}
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 3 | 1 | 4 | 5 | 1 | 2 | 5 | 2 |

# Algoritmo e Memória

(versão imperativa)

- *Pointer-based*
- *Non-Contiguous, Non-Random Access*
  - `std::forward_list<T>`
- Contar a incidência de um `int` numa lista ligada simples

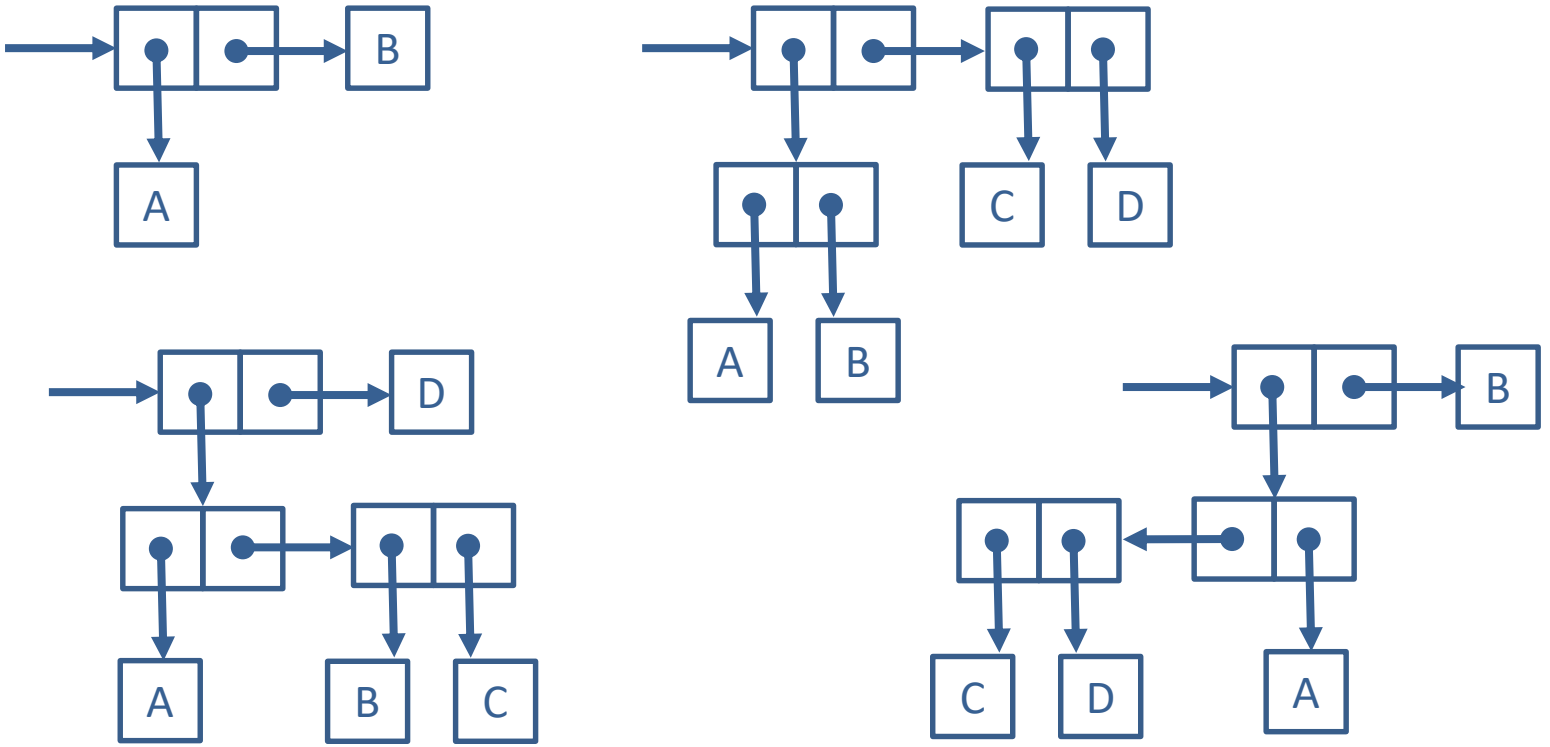
```
size_t count_occurrences(node* ptr, int target)
{
    size_t counter = 0;
    while (ptr != nullptr)
    {
        if (ptr->value == target)
            ++counter;
        ptr = ptr->next;
    }
    return counter;
}
```

```
struct node
{
    node(int value, node* next = nullptr) :
        value(value), next(next) {}
    int value;
    node* next;
};
```

# Algoritmo e Memória

(*pointer-based*)

- *Closure property*
  - *Hierarchical Data Structures*




https://mitpress.mit.edu/sites/default/files/sicp/full-text/book/book-Z-H-15.html#%_sec_2.2

# Algoritmo e Memória

(versão declarativa - STL)

- Iterator: `InputIterator`
- Algoritmos: `std::count` e `std::count_if`
  - Valor (T)

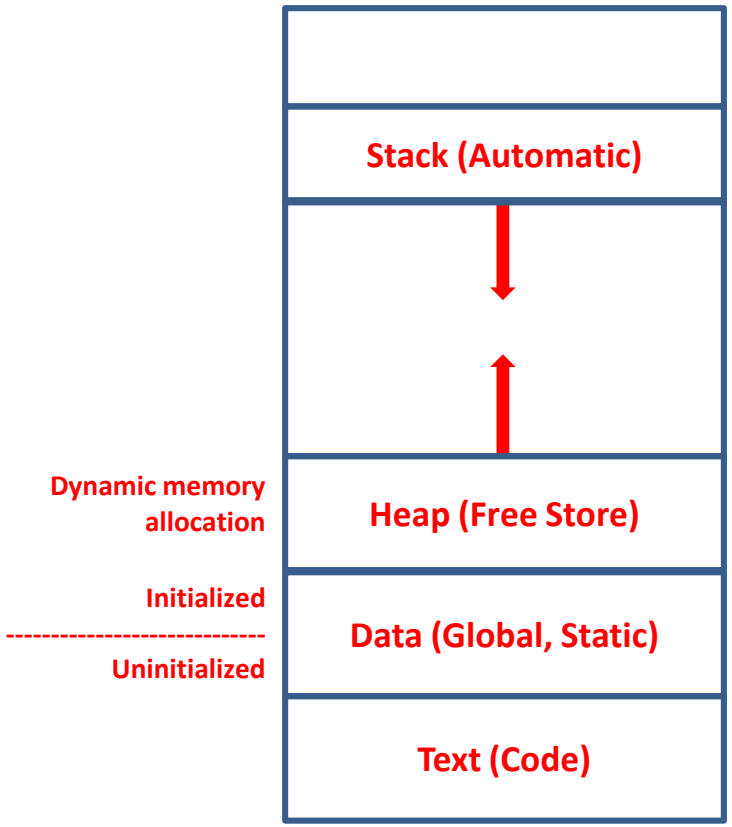
```
std::vector<int> ys{ 1, 2, 2, 3, 1, 4, 5, 1, 2, 5, 2 };
std::cout << std::count(std::begin(ys), std::end(ys), 2) << "\n";

std::forward_list<int> zs{ 1, 2, 2, 3, 1, 4, 5, 1, 2, 5, 2 };
std::cout << std::count(std::begin(zs), std::end(zs), 2) << "\n";
```

  - *Predicate* (`UnaryPredicate`)

```
std::count_if(std::begin(ys), std::end(ys),
              [](int y) { return !(y & 0x1) /* is even? */; })
```

# *Layout* de Memória do C++

- Os sistemas operacionais que determinam o *layout* de memória de um processo




**Exemplo:**

```
[ 8b]     735644219208 (0x00ab47cff748)       on stack
[ 4b]     735644219200 (0x00ab47cff740):    1 on stack
[ 4b]     735644218916 (0x00ab47cff624):   42 on stack
[ 8b]     735644218992 (0x00ab47cff670)       on stack
[ 8b]     735644218984 (0x00ab47cff668)       on stack
[ 8b]     735644218976 (0x00ab47cff660)       on stack
[ 8b]     735644218968 (0x00ab47cff658)       on stack
[ 4b]     735644218920 (0x00ab47cff628):   84 on stack
[ 4b]     735644218924 (0x00ab47cff62c):  126 on stack
[ 4b]     735644218928 (0x00ab47cff630):  168 on stack
[24b]     735644219032 (0x00ab47cff698):    4 on stack
[ 4b]    2708199769936 (0x02768d490750):   42 on heap
[ 4b]    2708199769968 (0x02768d490770):   84 on heap
[ 4b]    2708199770000 (0x02768d490790):  126 on heap
[ 4b]    2708199770032 (0x02768d4907b0):  168 on heap
[ 4b]    2708199770064 (0x02768d4907d0):   42 on heap
[ 4b]    2708199770068 (0x02768d4907d4):   84 on heap
[ 4b]    2708199770072 (0x02768d4907d8):  126 on heap
[ 4b]    2708199770076 (0x02768d4907dc):  168 on heap
[ 4b]  140697878735728 (0x7ff6c7152b70):    0 on data (uninitialized)
[ 4b]  140697878735744 (0x7ff6c7152b80):    0 on data (uninitialized)
[ 4b]  140697878732800 (0x7ff6c7152000):   42 on data (initialized)
[ 4b]  140697878732804 (0x7ff6c7152004):   42 on data (initialized)
       140697878602592 (0x7ff6c7132360):   printf on text
       140697878597792 (0x7ff6c71310a0):     main on text
       140697878597712 (0x7ff6c7131050):    print on text
```

# Recursos e *RAII*

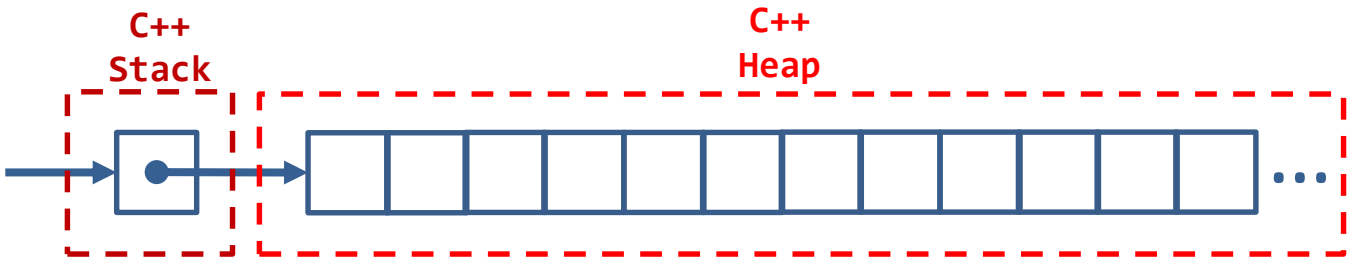
- *Resource Acquisition Is Initialization* (*RAII*)
- Idioma para prevenção de *leaks*
  - Memória, Arquivos, *Sockets*, *Threads*, entre outros recursos
- O recurso é:
  - Adquirido pelo objeto na sua inicialização (*constructor*)
  - Mantido durante o tempo de vida do objeto
  - Liberado na destruição do objeto (*destructor*)
- *"Modern C++ avoids using heap memory as much as possible by declaring objects on the stack. When a resource is too large for the stack, then it should be owned by an object… The owning object itself is declared on the stack."*
  - *https://docs.microsoft.com/en-us/cpp/cpp/object-lifetime-and-resource-management-modern-cpp*

```cpp
//Using RAII idiom
static void write_strings_to_file(const char* filename, std::vector<std::string> ss)
{
    //stack
    file f(filename, "w+");
    f.write_all_lines(ss);
}
```

https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Re-raii

# dynamic_array<T>

(*interface*, add e remove)




```
//dynamic array interface
/*
void add(const T&);
void remove();
T& operator[](std::size_t) const;
T& at(std::size_t) const;
bool empty() const;
std::size_t size() const;
std::size_t capacity() const;
void clear();
T* data();
T* begin();
T* end();
const T* cbegin() const;
const T* cend() const;
*/
```

```
void add(const T& item)
{
    if (size() == capacity())
    {
        const std::size_t N = 2 * capacity();
        if (resize(N) < N)
            throw std::bad_alloc();
    }
    items[current_size++] = item;
}

void remove()
{
    if (empty())
        throw std::runtime_error("is empty");

    if (!try_shrink())
        --current_size;
}
```

# dynamic_array<T>

## (resize e try_shrink)

```
std::size_t resize(std::size_t capacity, bool preserve = true)
{
    capacity = clamp(capacity, DYNAMIC_ARRAY_MIN_CAPACITY, DYNAMIC_ARRAY_MAX_CAPACITY);
    T* new_items = new T[capacity];
    if (preserve)
    {
        for (std::size_t i = 0, l = min(current_size, capacity); i < l; ++i)
            new_items[i] = items[i];
    }
    if (items)
        delete[] items;
    items = new_items;
    max_size = capacity;
    if (max_size < current_size)
        current_size = max_size;
    return capacity;
}
```

---

```
bool try_shrink(bool preserve = true)
{
    bool can_shrink = (2 * DYNAMIC_ARRAY_MIN_CAPACITY) <= size() - 1 && size() - 1 == capacity() / 4;
    if (can_shrink)
        resize(capacity() / 4, preserve);
    return can_shrink;
}
```

# dynamic_array<T>

(*benchmark*)

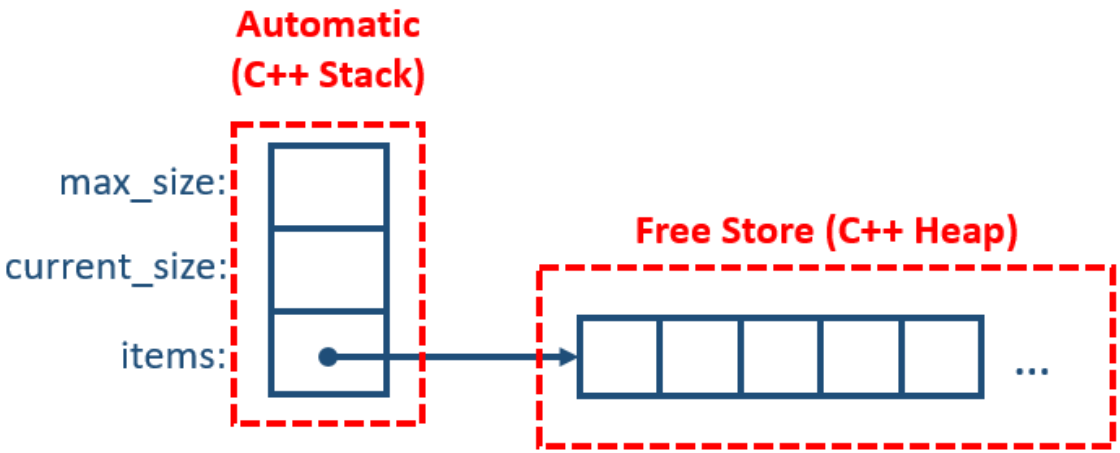
MinGW g++ v7.3.0 x64–O3

```
dynamic_array:
================================================================================
   Name (baseline is *)   |   Dim    | Total ms | ns/op  |Baseline| Ops/second
================================================================================
       bench_dynarr_add * |    1000  |    0.006 |      5 |      - |174550532.4
  bench_std_vec_push_back |    1000  |    0.011 |     10 |  1.842 | 94741828.5
       bench_dynarr_add * |   10000  |    0.109 |     10 |      - | 91603612.8
  bench_std_vec_push_back |   10000  |    0.087 |      8 |  0.801 |114345827.5
       bench_dynarr_add * |  100000  |    0.653 |      6 |      - |153165627.2
  bench_std_vec_push_back |  100000  |    0.467 |      4 |  0.715 |214214411.5
       bench_dynarr_add * | 1000000  |    4.578 |      4 |      - |218418774.8
  bench_std_vec_push_back | 1000000  |    4.840 |      4 |  1.057 |206619382.5
================================================================================

C:\Users\Fabio Galuppo\AlgsCpp\code>
```

Visual C++ v15.0.26430.12 x64 Release

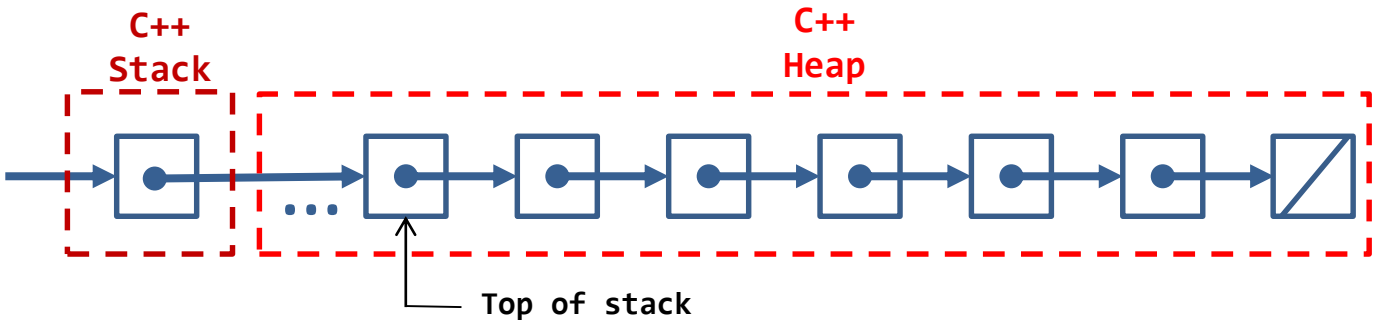
```
dynamic_array:
================================================================================
   Name (baseline is *)   |   Dim    | Total ms | ns/op  |Baseline| Ops/second
================================================================================
       bench_dynarr_add * |    1000  |    0.009 |      9 |      - |106963311.6
  bench_std_vec_push_back |    1000  |    0.168 |    167 | 17.935 |  5964096.1
       bench_dynarr_add * |   10000  |    0.065 |      6 |      - |154954675.8
  bench_std_vec_push_back |   10000  |    0.068 |      6 |  1.061 |146081367.3
       bench_dynarr_add * |  100000  |    0.508 |      5 |      - |197031134.9
  bench_std_vec_push_back |  100000  |    0.776 |      7 |  1.529 |128827961.9
       bench_dynarr_add * | 1000000  |    4.872 |      4 |      - |205250934.7
  bench_std_vec_push_back | 1000000  |    9.049 |      9 |  1.857 |110504991.2
================================================================================
```

# `dynamic_array<T>` e *layout* de memória




# node_based_stack<T>

(*interface*, *node*, push e pop)




```
//stack interface
/*
bool empty() const;
size_t size() const;
T& top();
void push(const T&);
void pop();
void swap(stack&);
*/
```

```
struct singly_linked_node final
{
    T item;
    singly_linked_node* next = nullptr;
};
```

```
void push(const T& val)
{
    singly_linked_node* next = root;
    root = new singly_linked_node();
    root->item = val;
    root->next = next;
    ++n;
}
```

```
void pop()
{
    singly_linked_node* previous = root;
    root = root->next;
    delete previous;
    --n;
}
```

# linear_search

(algoritmo de busca linear)

- *Container* (N *algorithms* **x** M *containers*)

```
template <typename T>
static std::size_t linear_search(const dynamic_array::dynamic_array<T>& xs, const T& x)
{
    for (std::size_t i = 0; i < xs.size(); ++i)
        if (xs[i] == x)
            return i;
    return NOT_FOUND;
}

template <typename T>
static std::size_t linear_search(const std::vector<T>& xs, const T& x)
{
    for (std::size_t i = 0; i < xs.size(); ++i)
        if (xs[i] == x)
            return i;
    return NOT_FOUND;
}
```

- *Iterator* (N *algorithms* **+** M *containers*)
  - Desacopla o algoritmo do *container*

```
template<typename InputIterator, typename T>
static InputIterator linear_search(InputIterator first, InputIterator last, const T &x)
{
    for (; first != last; ++first)
        if (*first == x)
            return first;
    return last;
}
```

# std::find

```
910  // find
911
912  template <class _InputIterator, class _Tp>
913  _LIBCPP_NODISCARD_EXT inline
914  _LIBCPP_INLINE_VISIBILITY _LIBCPP_CONSTEXPR_AFTER_CXX17
915  _InputIterator
916  find(_InputIterator __first, _InputIterator __last, const _Tp& __value_)
917  {
918      for (; __first != __last; ++__first)
919          if (*__first == __value_)
920              break;
921      return __first;
922  }
```

```
924  // find_if
925
926  template <class _InputIterator, class _Predicate>
927  _LIBCPP_NODISCARD_EXT inline
928  _LIBCPP_INLINE_VISIBILITY _LIBCPP_CONSTEXPR_AFTER_CXX17
929  _InputIterator
930  find_if(_InputIterator __first, _InputIterator __last, _Predicate __pred)
931  {
932      for (; __first != __last; ++__first)
933          if (__pred(*__first))
934              break;
935      return __first;
936  }
937
938  // find_if_not
939
940  template<class _InputIterator, class _Predicate>
941  _LIBCPP_NODISCARD_EXT inline
942  _LIBCPP_INLINE_VISIBILITY _LIBCPP_CONSTEXPR_AFTER_CXX17
943  _InputIterator
944  find_if_not(_InputIterator __first, _InputIterator __last, _Predicate __pred)
945  {
946      for (; __first != __last; ++__first)
947          if (!__pred(*__first))
948              break;
949      return __first;
950  }
```

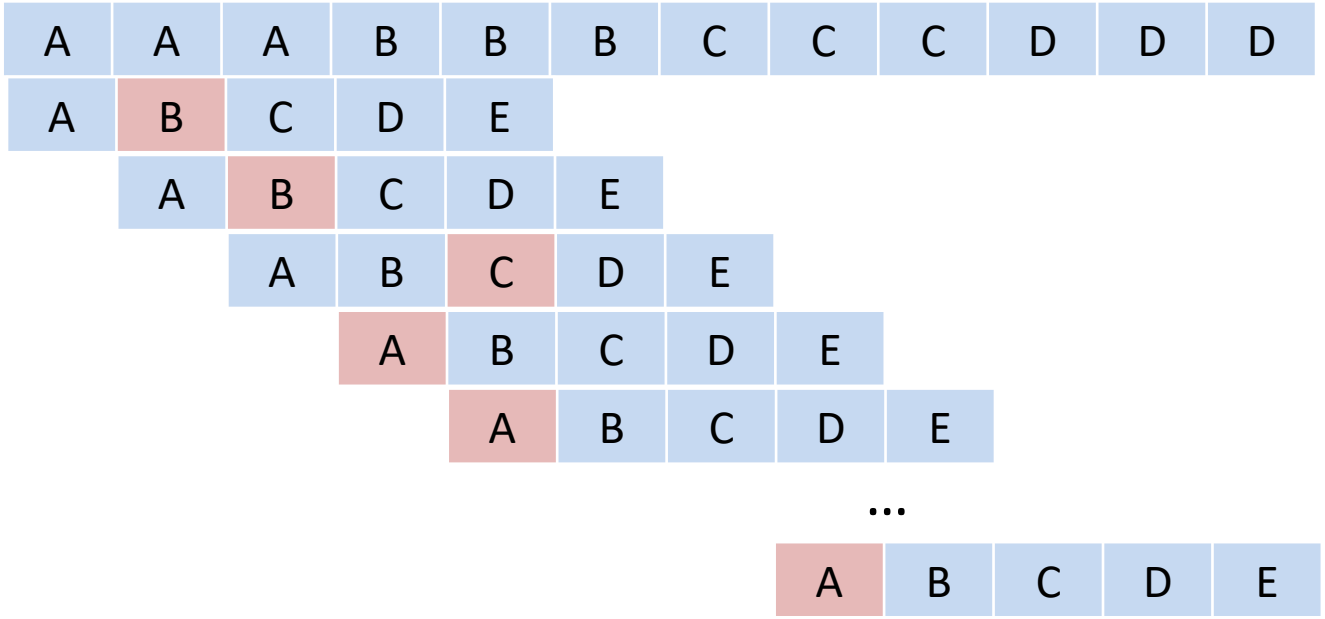
https://github.com/llvm-mirror/libcxx/blob/master/include/algorithm#L910

# *Find substring*

(algoritmo de busca não-linear)

- Brute-force

$$TEXT = AAABBBCCCDDD \qquad M = |TEXT| = 12$$
$$PATTERN = ABCDE \qquad N = |PATTERN| = 5$$




$$\big(N * (M - N + 1)\big) = MN - N^2 + N \quad \text{where } M > N = O(MN)$$

# *Find substring*

(busca por força bruta)

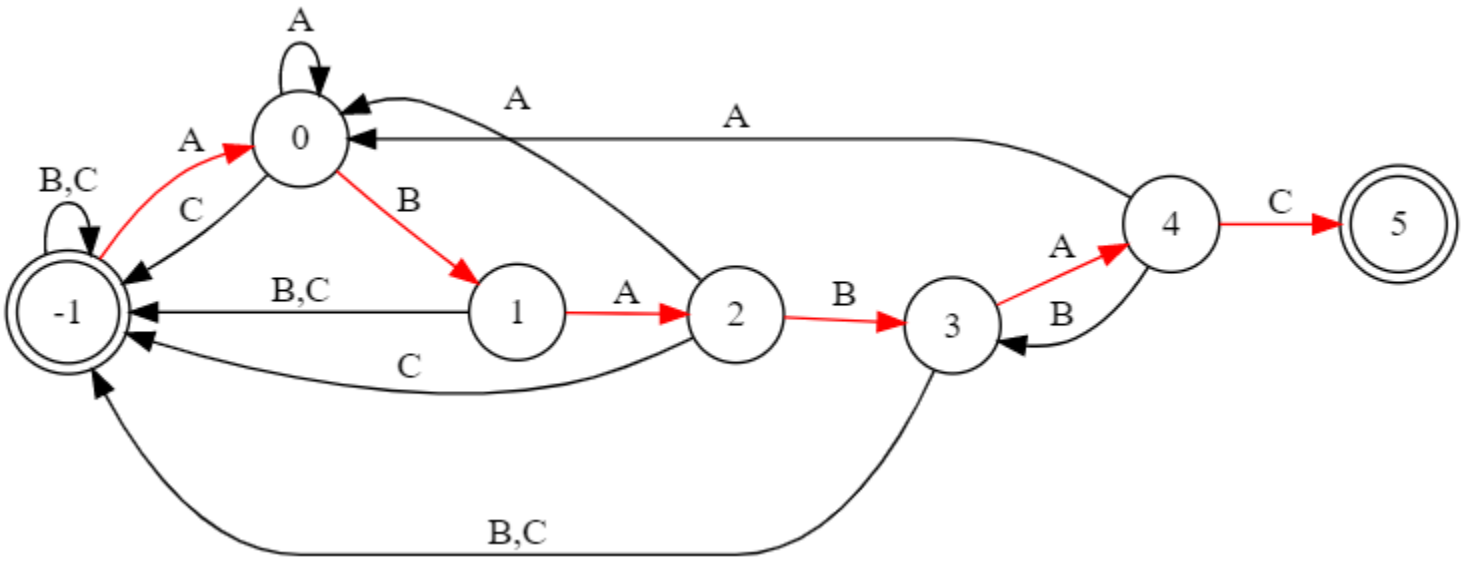
```
std::size_t search_algo(const std::string& text, std::size_t init)
{
    //brute-force search algorithm

    const std::size_t T = text.size();
    const std::size_t P = pattern.size();
    if (P <= T + init)
    {
        for (std::size_t i = init; i <= T - P; ++i)
        {
            std::size_t j = 0;
            while (j < P)
            {
                if (text[i + j] != pattern[j])
                    break;
                ++j;
            }
            if (j == P)
                return i;
        }
    }
    return std::string::npos;
}
```

# *Find substring*

(autômato finito determinístico)

- Knuth-Morris-Pratt
  - Primeiro algoritmo de busca de *substring* em tempo linear
  - Pré-processamento do *pattern*, baseado em autômato (DFA)
    - Usado para deslocamento

$$PATTERN = ABABAC$$



# *Find substring*

(tabela de deslocamento)

- Knuth-Morris-Pratt
  - Pré-processamento do *pattern*, baseado em autômato (DFA)
    - Usado para deslocamento
      - Cada fase do padrão, considerar a largura máxima do prefixo adequado e do sufixo adequado
        - » *proper prefix == proper suffix*

$PATTERN = ABABAC$

| P1 | P2 | P3 | P4 | P5 | P6 |
|---|---|---|---|---|---|
| A | B | A | B | A | C |

$N = |PATTERN| = 6$

| | Proper prefix | Proper suffix | Len | | Proper prefix | Proper suffix | Len |
|---|---|---|---|---|---|---|---|
| P1 | - | - | 0 | P4 | ABA, AB, A | BAB, AB, A | 2 |
| P2 | A | B | 0 | P5 | ABAB, ABA, AB, A | BABA, ABA, BA, A | 3 |
| P3 | AB, A | BA, A | 1 | P6 | ABABA, ABAB, ABA, AB, A | BABAC, ABAC, BAC, AC, C | 0 |

$Length = \{0,0,1,2,3,0\}$

$Shift = \{-1,-1,0,1,2,-1\}$

# *Find substring*

(deslocamento e busca)

- Knuth-Morris-Pratt
  - Tabela de deslocamento (exemplos)

$$PATTERN = ABABAC \qquad Shift = \{-1,-1,0,1,2,-1\}$$

$$PATTERN = AAAAAA \qquad Shift = \{-1,0,1,2,3,4\}$$

$$PATTERN = ABCDEF \qquad Shift = \{-1,-1,-1,-1,-1,-1\}$$

$$PATTERN = ABBACA \qquad Shift = \{-1,-1,-1,0,-1,0\}$$

$$PATTERN = ABBAAA \qquad Shift = \{-1,-1,-1,0,0,0\}$$

  - Busca

$$PATTERN = \underset{j+1}{ABABAC} \qquad TEXT = \underset{i}{AABABCAAABABAC}$$

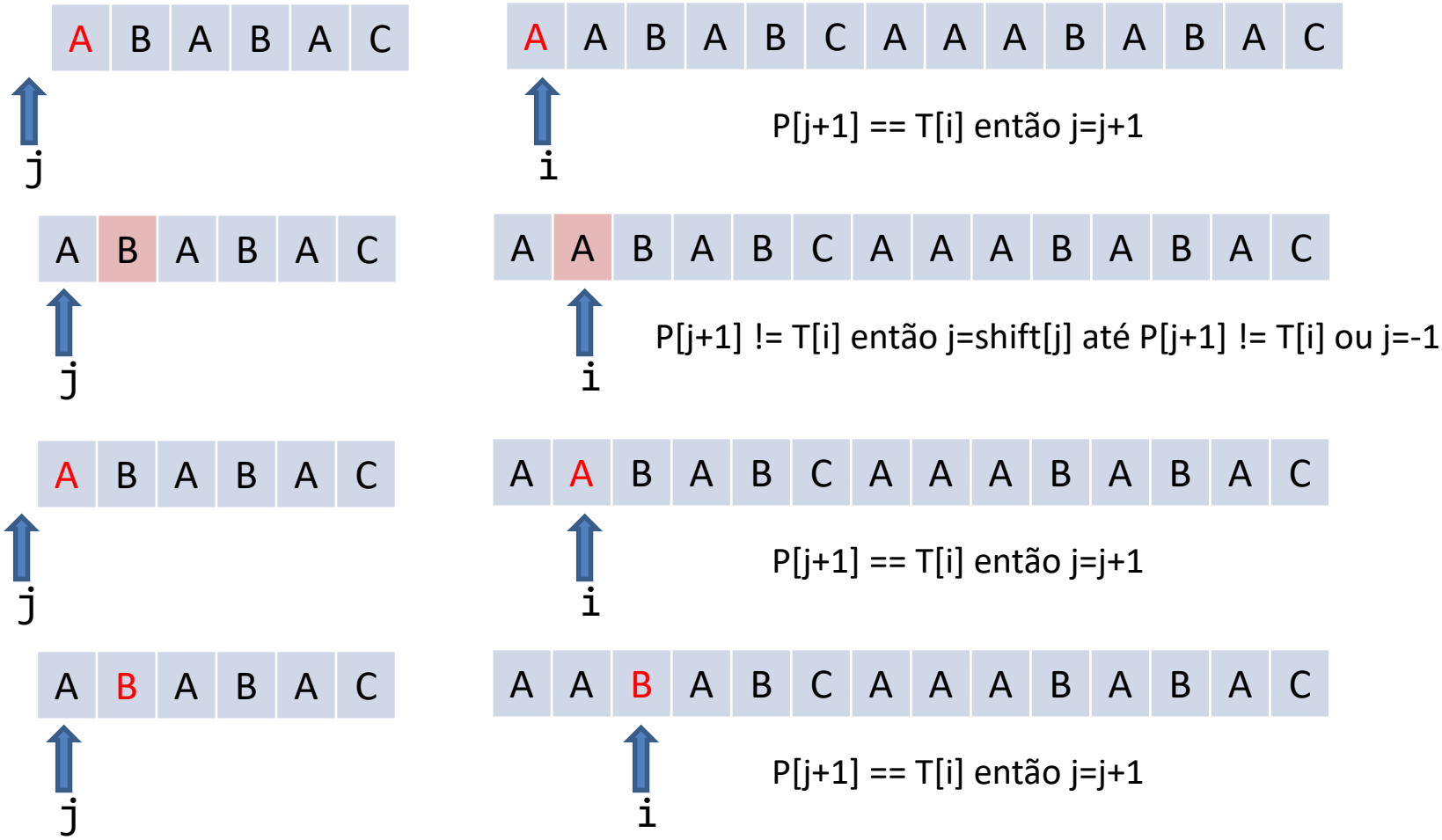
| | | | |
|---|---|---|---|
| 1 | j: -1 i: 0 A == A? | j: -1 i: 7 A == A? | 8 |
| 2 | j: -1 i: 1 A == A? | j: -1 i: 8 A == A? | 9 |
| 3 | j: 0 i: 2 B == B? | j: 0 i: 9 B == B? | 10 |
| 4 | j: 1 i: 3 A == A? | j: 1 i: 10 A == A? | 11 |
| 5 | j: 2 i: 4 B == B? | j: 2 i: 11 B == B? | 12 |
| 6 | j: -1 i: 5 A == C? | j: 3 i: 12 A == A? | 13 |
| 7 | j: -1 i: 6 A == A? | j: 4 i: 13 C == C? | 14 |

# *Find substring*

(passo a passo - 1 de 5)

$$PATTERN = ABABAC \qquad TEXT = AABABCAAABABAC$$

$$Shift = \{-1, -1, 0, 1, 2, -1\}$$

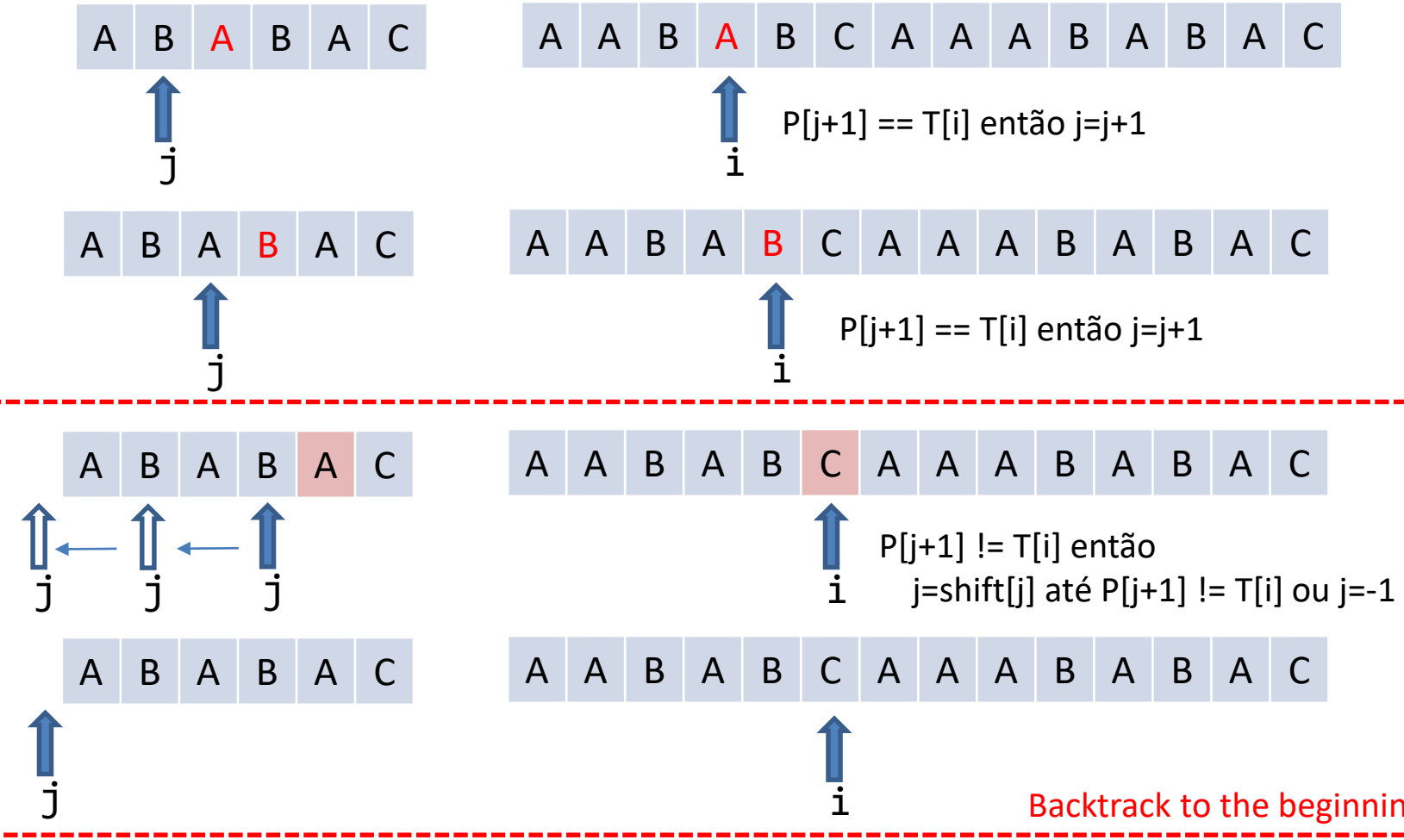


# *Find substring*

(passo a passo - 2 de 5)

$$PATTERN = ABABAC \qquad TEXT = AABABCAAABABAC$$

$$Shift = \{-1, -1, 0, 1, 2, -1\}$$

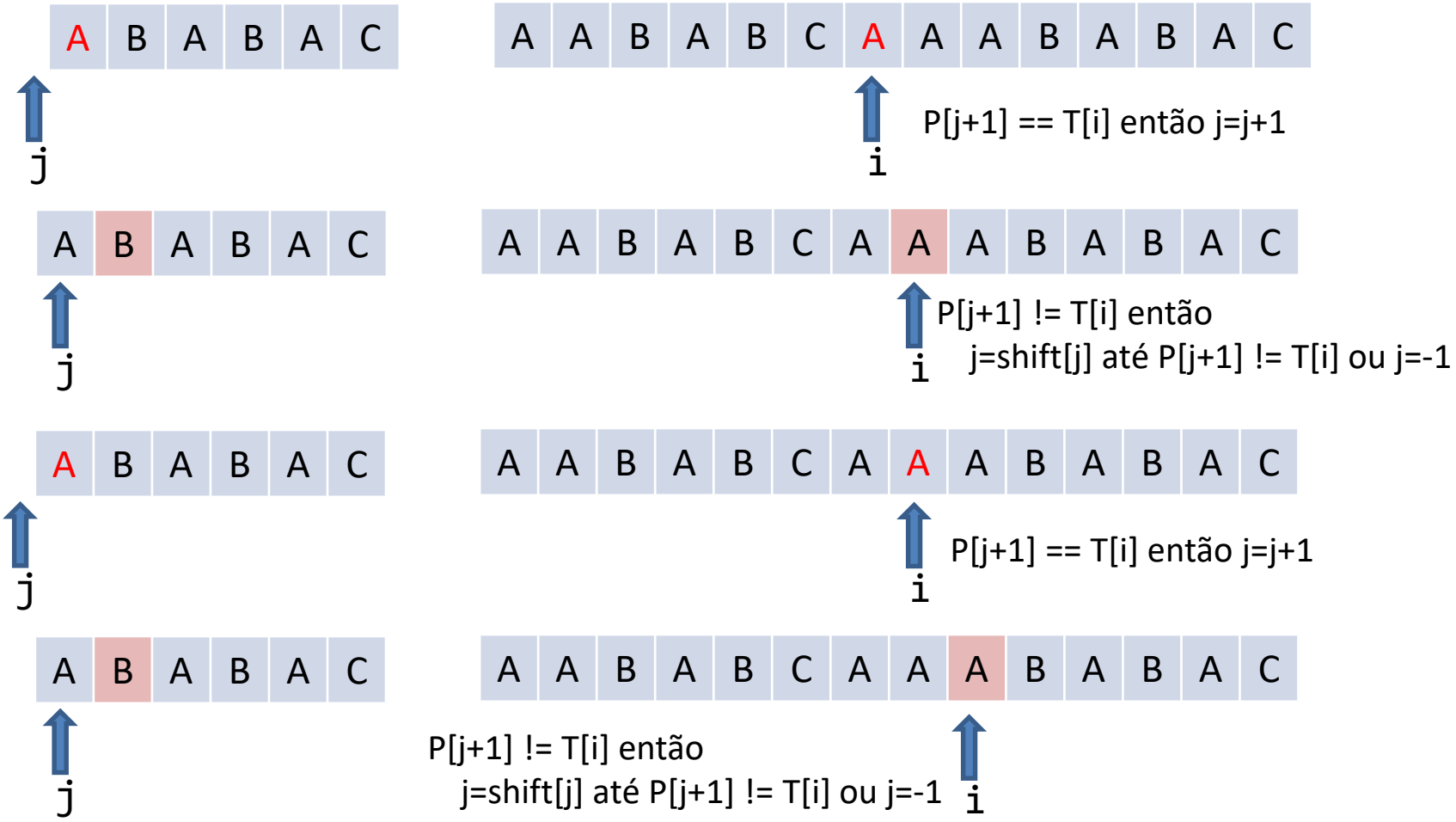


# *Find substring*

(passo a passo - 3 de 5)

$$PATTERN = ABABAC \qquad TEXT = AABABCAAABABAC$$

$$Shift = \{-1, -1, 0, 1, 2, -1\}$$

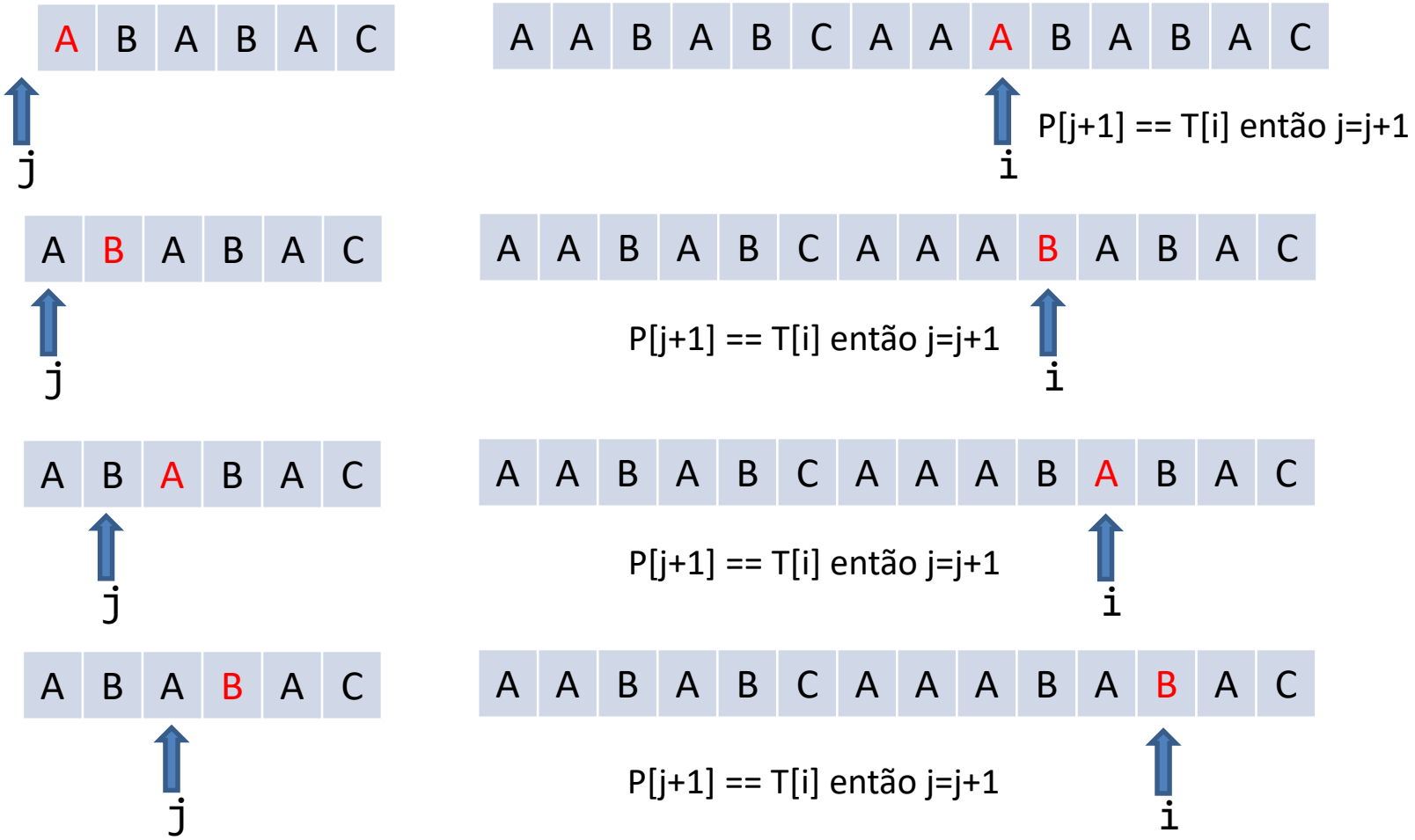


# *Find substring*

(passo a passo - 4 de 5)

$$PATTERN = ABABAC \qquad TEXT = AABABCAAABABAC$$

$$Shift = \{-1, -1, 0, 1, 2, -1\}$$

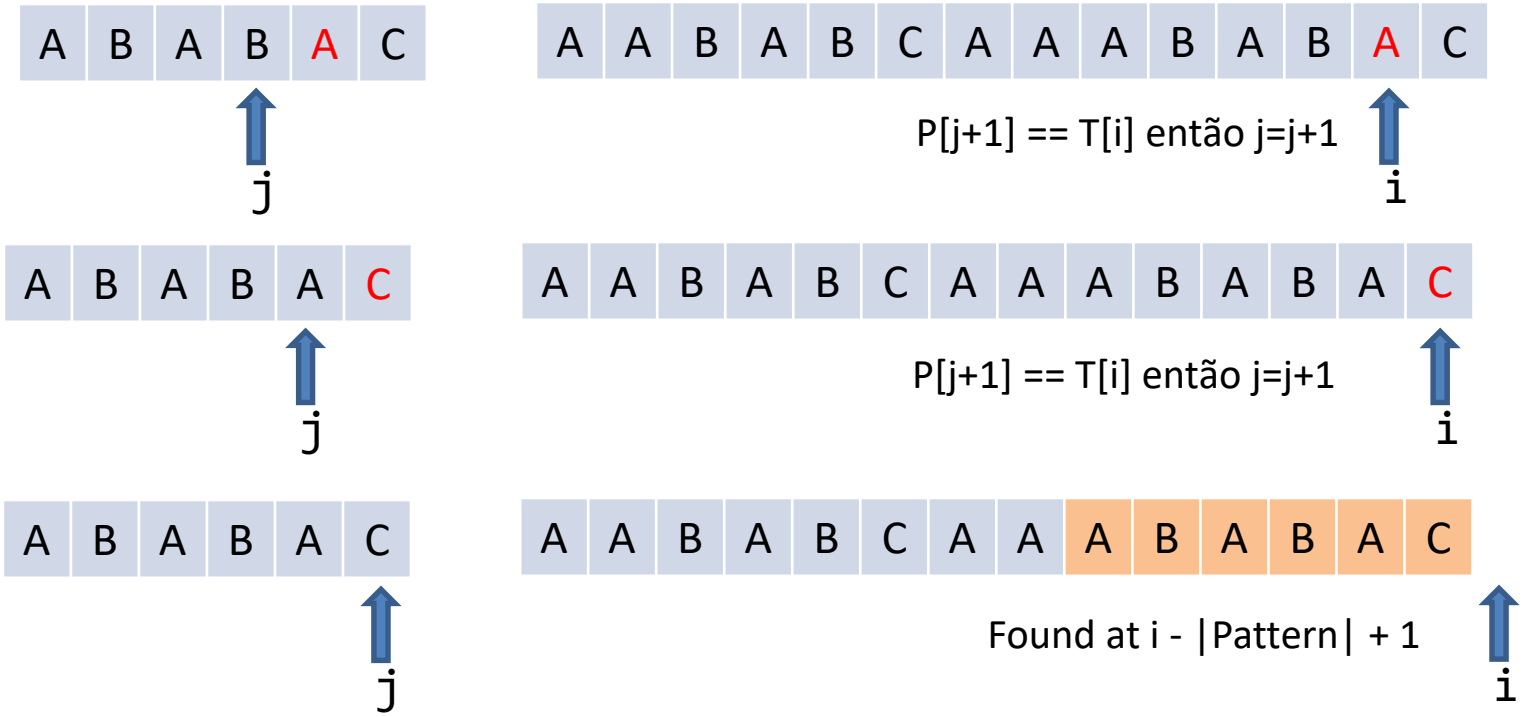


# *Find substring*

(passo a passo - 5 de 5)

$$PATTERN = ABABAC \qquad TEXT = AABABCAAABABAC$$

$$Shift = \{-1, -1, 0, 1, 2, -1\}$$




| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | B | A | B | C | A | A | A | B | A | B | A | C |

$$M = |TEXT| = 14$$

$$N = |PATTERN| = 6$$

$$\theta(M) + \theta(N) = \theta(M + N)$$

# *Find substring*

(algoritmo KMP de busca linear)

```
explicit knuth_morris_pratt(const std::string& pattern)
    : base(pattern), shift(pattern.size(), -1)
{
    //build shift table

    const int P = static_cast<int>(pattern.size());
    int j = -1;
    for (int i = 1; i < P; ++i)
    {
        while (j >= 0 && pattern[j + 1] != pattern[i])
            j = shift[j];
        if (pattern[j + 1] == pattern[i]) ++j;
        shift[i] = j;
    }
}
```

```
std::size_t search_algo(const std::string& text, std::size_t init)
{
    //search algorithm

    const std::size_t T = text.size();
    const std::size_t P = pattern.size() - 1;
    if ((P + 1) <= T + init)
    {
        std::size_t j = -1;
        for (std::size_t i = init; i < T; ++i)
        {
            while (j != -1 && pattern[j + 1] != text[i])
                j = shift[j];
            if (pattern[j + 1] == text[i]) ++j;
            if (j == P)
                return i - P;
        }
    }
    return std::string::npos;
}
```

# *CPF*

## (algoritmo descrito na Wikipédia)

$$\vec{a} = [9, 8, 7, 6, 5, 4, 3, 2, 1]$$ 9 primeiros dígitos do CPF em ordem reversa

$$d'_1 = \left[ \left( \sum_{i=1}^{9} a_i \times \left[ 9 - (i \mod 10) \right] \right) \right]$$

$$d'_2 = \left[ \left( \sum_{i=1}^{9} a_i \times \left\{ 9 - \left[ (i+1) \mod 10 \right] \right\} \right) \right]$$

$$d_1 = (d'_1 \mod 11) \mod 10$$

$$d_2 = \left\{ \left[ d'_2 + (d_1 \times 9) \right] \mod 11 \right\} \mod 10$$

https://pt.wikipedia.org/wiki/Cadastro_de_Pessoas_F%C3%ADsicas#C%C3%A1lculo_do_d%C3%ADgito_verificador

# *CPF*

## (abordagem Álgebra Linear)

$\vec{a} = [1, 2, 3, 4, 5, 6, 7, 8, 9]$ 9 primeiros dígitos do CPF

$\vec{b} = [10, 9, 8, 7, 6, 5, 4, 3, 2]$ sequência numérica em ordem reversa

$x = \vec{a} \cdot \vec{b}$ produto escalar (ou produto interno)

$$d'_1 = x \mod 11$$

$$d_1 = \begin{cases} 0, & \text{if } d'_1 < 2. \\ 11 - d'_1, & \text{otherwise.} \end{cases}$$

$$d'_2 = (2 \times d_1 + x + \sum_{i=1}^{9} a_i) \mod 11$$

$$d_2 = \begin{cases} 0, & \text{if } d'_2 < 2. \\ 11 - d'_2, & \text{otherwise.} \end{cases}$$

# *CPF*

## (algoritmo descrito em GeradorCPF)

- $d_1 =$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 4 | 4 | 4 | 7 | 7 | 7 | 9 primeiros dígitos do CPF |
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | sequência numérica em ordem reversa |

1) Multiplicar os valores das colunas, aplicar o somatório (Σ) e o módulo de 11
2) Se o resultado for maior ou igual a 2, substrair 11 senão é 0

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 8 | 28 | 24 | 20 | 28 | 21 | 14 | $= 162 \bmod 11 = 8 = 11 - 8 = 3$ |

- $d_2 =$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 4 | 4 | 4 | 7 | 7 | 7 | 3 | 9 primeiros dígitos do CPF + $d_1$ |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | sequência numérica em ordem reversa |

1) Multiplicar os valores das colunas, aplicar o somatório (Σ) e o módulo de 11
2) Se o resultado for maior ou igual a 2, substrair 11 senão é 0

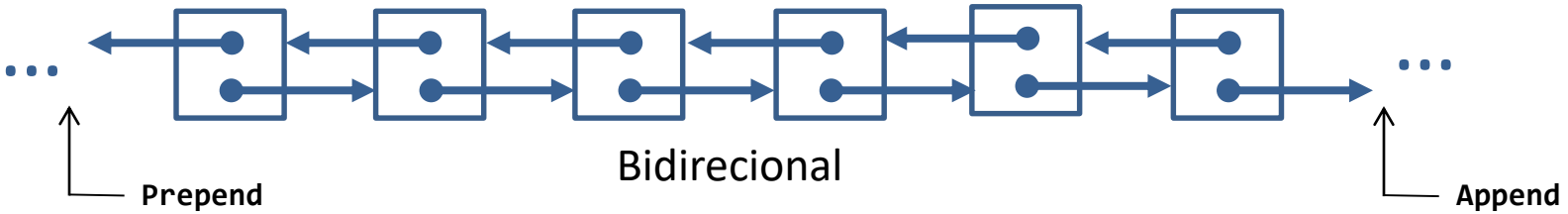
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 11 | 10 | 9 | 32 | 28 | 24 | 35 | 28 | 21 | 6 | $= 204 \bmod 11 = 6 = 11 - 6 = 5$ |

$$d_1\ d_2 = 35$$

https://www.geradorcpf.com/algoritmo_do_cpf.htm

# doubly_linked_list<T>

(*interface*)




raw pointer

```
//doubly_linked_list interface
/*
void traverse_forward(doubly_linked_list<T>&, Visit);
void traverse_backward(doubly_linked_list<T>&, Visit);
doubly_linked_node<T>* find_forward(doubly_linked_list<T>&, const T&);
doubly_linked_node<T>* find_backward(doubly_linked_list<T>&, const T&);
void prepend(doubly_linked_list<T>&, const T&);
void append(doubly_linked_list<T>& list, const T&);
void insert_before(doubly_linked_list<T>&, const T&, const T&);
void insert_after(doubly_linked_list<T>&, const T&, const T&);
void remove_forward(doubly_linked_list<T>&, const T&);
void remove_backward(doubly_linked_list<T>&, const T&);
void remove_front(doubly_linked_list<T>&);
void remove_back(doubly_linked_list<T>&);
void clear(doubly_linked_list<T>&);
std::size_t count(doubly_linked_list<T>&);
std::ptrdiff_t distance_forward(doubly_linked_list<T>&, const T&)
std::ptrdiff_t distance_backward(doubly_linked_list<T>&, const T&)
*/
```

```
template <typename T>
struct doubly_linked_node final
{
    doubly_linked_node<T>* prev = nullptr;
    doubly_linked_node<T>* next = nullptr;
    T data;
};

template <typename T>
struct doubly_linked_list final
{
    doubly_linked_node<T>* first = nullptr;
    doubly_linked_node<T>* last = nullptr;
};
```

# Ponteiros em C++

*raw (dumb) pointer* e *smart pointers*

- *Raw pointer* (ou *dumb pointer*)
  - Armazena o endereço de um objeto em memória
  - Se tiver *ownership* sobre o recurso, o controle de tempo de vida é manual
  - Pode se atribuir o endereço de outra variável não-ponteiro ou `nullptr`
  - Suporta aritmética de ponteiros
- *Smart pointer*
  - Encapsula um *raw pointer*
  - Simula a sintaxe e a parte semântica de um ponteiro
    - Usualmente não suporta aritmética
- Tem *ownership* sobre o recurso, o controle de tempo de vida é automático
  - Este é o objetivo principal de um *smart pointer*
- Idioma RAII
  - *Resource Acquisition Is Initialization*
- Alternativa ao *Garbage Collection*

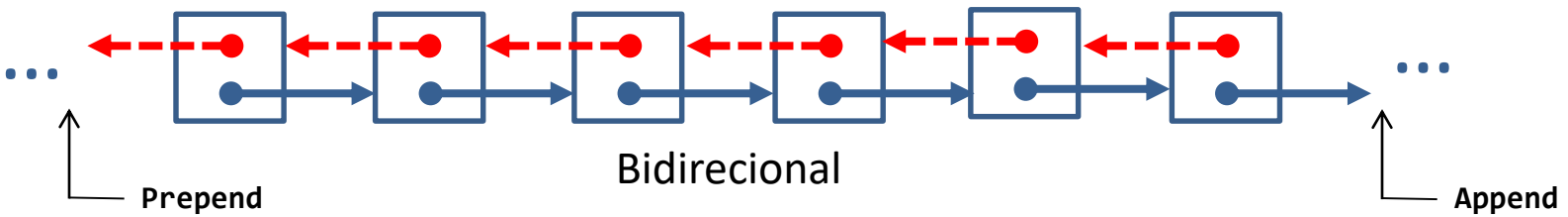
# Ponteiros em C++

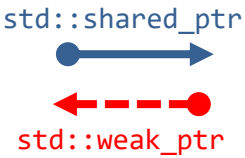
## *smart pointers C++* (*ownership* e gerenciamento de recurso)

- Três tipos de smart pointers a partir do C++ 11:
  - `std::unique_ptr`
  - `std::shared_ptr`
  - `std::weak_ptr`
- **`unique_ptr`**
  - *Ownership* exclusivo (não compartilhado) para gerenciamento de recurso
  - Suporta apenas *move semantics* para transferir o *ownership*
    - Não suporta cópia
  - Pequeno e eficiente, é equivalente ao tamanho de um ponteiro (ex.: `sizeof(int*)`)
- **`shared_ptr`**
  - *Ownership* compartilhado para gerenciamento de recurso
  - Suporta contagem de referência (*refcount*) para gerenciamento de recurso compartilhado
    - Quando a contagem chega a zero, o recurso é liberado (destruído)
    - Suporta ser copiado ou movido
  - Possui um pequeno *overhead* devido sua estrutura de controle para *refcount* e requerer operações atômicas
- **`weak_ptr`**
  - Pega emprestado um recurso, se estiver disponível
  - Permite quebrar referências cíclicas

# doubly_linked_list_alt<T>

(*interface*)



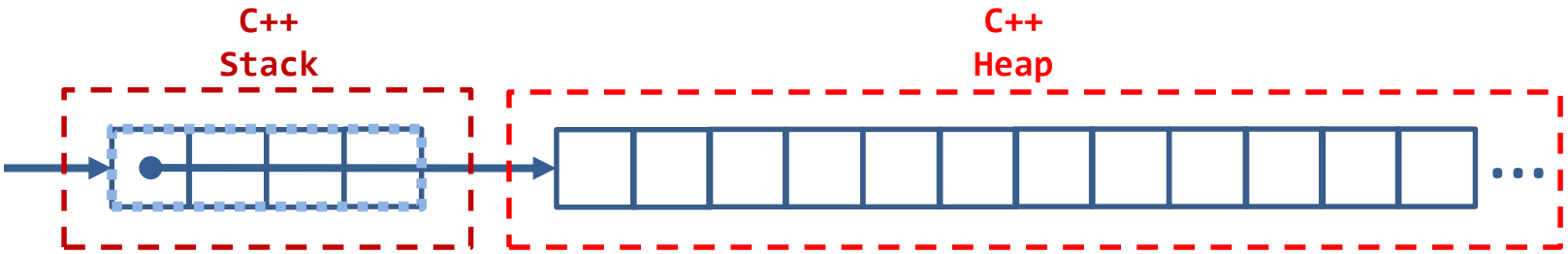




```
//doubly_linked_list_alt interface
/*
void traverse_forward(doubly_linked_list_alt<T>&, Visit);
void traverse_backward(doubly_linked_list_alt<T>& list, Visit visit);
std::shared_ptr<doubly_linked_node_alt<T>> find_forward(doubly_linked_list_alt<T>&, const T&);
std::shared_ptr<doubly_linked_node_alt<T>> find_backward(doubly_linked_list_alt<T>&, const T&);
void prepend(doubly_linked_list_alt<T>&, const T&);
void append(doubly_linked_list_alt<T>&, const T&);
void insert_before(doubly_linked_list_alt<T>&, const T&, const T&);
void insert_after(doubly_linked_list_alt<T>&, const T&, const T&);
void remove_forward(doubly_linked_list_alt<T>&, const T&);
void remove_backward(doubly_linked_list_alt<T>&, const T&);
void remove_front(doubly_linked_list_alt<T>&);
void remove_back(doubly_linked_list_alt<T>&);
void clear(doubly_linked_list_alt<T>&);
std::size_t count(doubly_linked_list_alt<T>&);
*/
```

```
template <typename T>
struct doubly_linked_node_alt final
{
    std::weak_ptr<doubly_linked_node_alt<T>> prev;
    std::shared_ptr<doubly_linked_node_alt<T>> next;
    T data;
};

template <typename T>
struct doubly_linked_list_alt final
{
    std::shared_ptr<doubly_linked_node_alt<T>> first;
    std::weak_ptr<doubly_linked_node_alt<T>> last;
};
```

# dynamic_array_lbo::dynamic_array<T>

(*interface* e estrutura interna – *local buffer optimization*)




```
//dynamic array interface
/*
void add(const T&);
void remove();
T& operator[](std::size_t) const;
T& at(std::size_t) const;
bool empty() const;
std::size_t size() const;
std::size_t capacity() const;
void clear();
T* data();
T* begin();
T* end();
const T* cbegin() const;
const T* cend() const;
bool is_data_on_stack() const;
std::size_t data_on_stack_size() const;
*/
```

```
private:
    std::size_t max_size, current_size;
    T* items = nullptr;

    static const std::size_t CACHE_LINE_SIZE = 64;

    static constexpr std::size_t storage_stack_size() { ... }

    static constexpr bool is_trivially_copyable() noexcept
    {
        return std::is_trivially_copyable<T>::value;
    }

    union
    {
        T* storage_heap;
        T  storage_stack[storage_stack_size()];
    };
};
```

# *Regular Types*

- É um tipo *semiregular* com comparação por igualdade
  - `operator==` e `operator!=`
- *Semiregular*, deve suportar:
  - Construção padrão (*default constructor*)
  - Construção por cópia (*copy constructor*)
  - Atribuição por cópia (*copy assignment*)
  - Destruição (*destructor*)
- Suporte a *total ordering*
  - Relação binária, onde qualquer dois elementos do conjunto possam ser comparáveis (https://en.wikipedia.org/wiki/Total_order)
  - `operator<`
    - E os operadores <=, >, >= implementados em termos de operator<
- Suporta os *Containers* da STL
  - Um tipo regular é garantido ser hospedado de forma apropriada
- "*A type is regular if and only if its basis includes equality, assignment, destructor, default constructor, copy constructor, total ordering, and underlying type.*" (http://elementsofprogramming.com/eop.pdf on page 7)

# *Containers* da STL

- Armazena e organiza uma coleção de objetos
  - Estruturas de dados
    - *Sequence containers*
    - *Container adaptors*
    - *Associative containers*
    - *Unordered Associative containers*
- Acesso aos elementos:
  - Direto
  - Através de *iterators*

| Container class templates | |
|---|---|
| **Sequence containers:** | |
| **array** C++11 | Array class (class template ) |
| **vector** | Vector (class template ) |
| **deque** | Double ended queue (class template ) |
| **forward_list** C++11 | Forward list (class template ) |
| **list** | List (class template ) |
| **Container adaptors:** | |
| **stack** | LIFO stack (class template ) |
| **queue** | FIFO queue (class template ) |
| **priority_queue** | Priority queue (class template ) |
| **Associative containers:** | |
| **set** | Set (class template ) |
| **multiset** | Multiple-key set (class template ) |
| **map** | Map (class template ) |
| **multimap** | Multiple-key map (class template ) |
| **Unordered associative containers:** | |
| **unordered_set** C++11 | Unordered Set (class template ) |
| **unordered_multiset** C++11 | Unordered Multiset (class template ) |
| **unordered_map** C++11 | Unordered Map (class template ) |
| **unordered_multimap** C++11 | Unordered Multimap (class template ) |

https://cplusplus.com/reference/stl/

# Recursão

- Quando uma função é implementada em termos de si mesma
  - Necessita de um caso base (para sair da recursividade)
- Método de resolução de problemas onde o resultado final dependerá das soluções das instâncias menores do mesmo problema
  - *Top-down approach*

$$L(n) = \begin{cases} 2, & n = 0 \\ 1, & n = 1 \\ L(n-1) + L(n-2), & n \geq 2 \end{cases}$$

https://oeis.org/A000032

# Recursão

- Recursion

```
//recursive
static long long lucas(short int n)
{
    if (n == 0) return 2LL;
    if (n == 1) return 1LL;
    return lucas(n - 1) + lucas(n - 2);
}
```

- *Tail recursion*

```
//tail recursive
static long long lucas_tailrec(short int n, long long current, long long next)
{
    if (n == 0) return current;
    return lucas_tailrec(n - 1, next, current + next);
}

static long long lucas_tailrec(short int n)
{
    return lucas_tailrec(n, 2LL, 1LL);
}
```

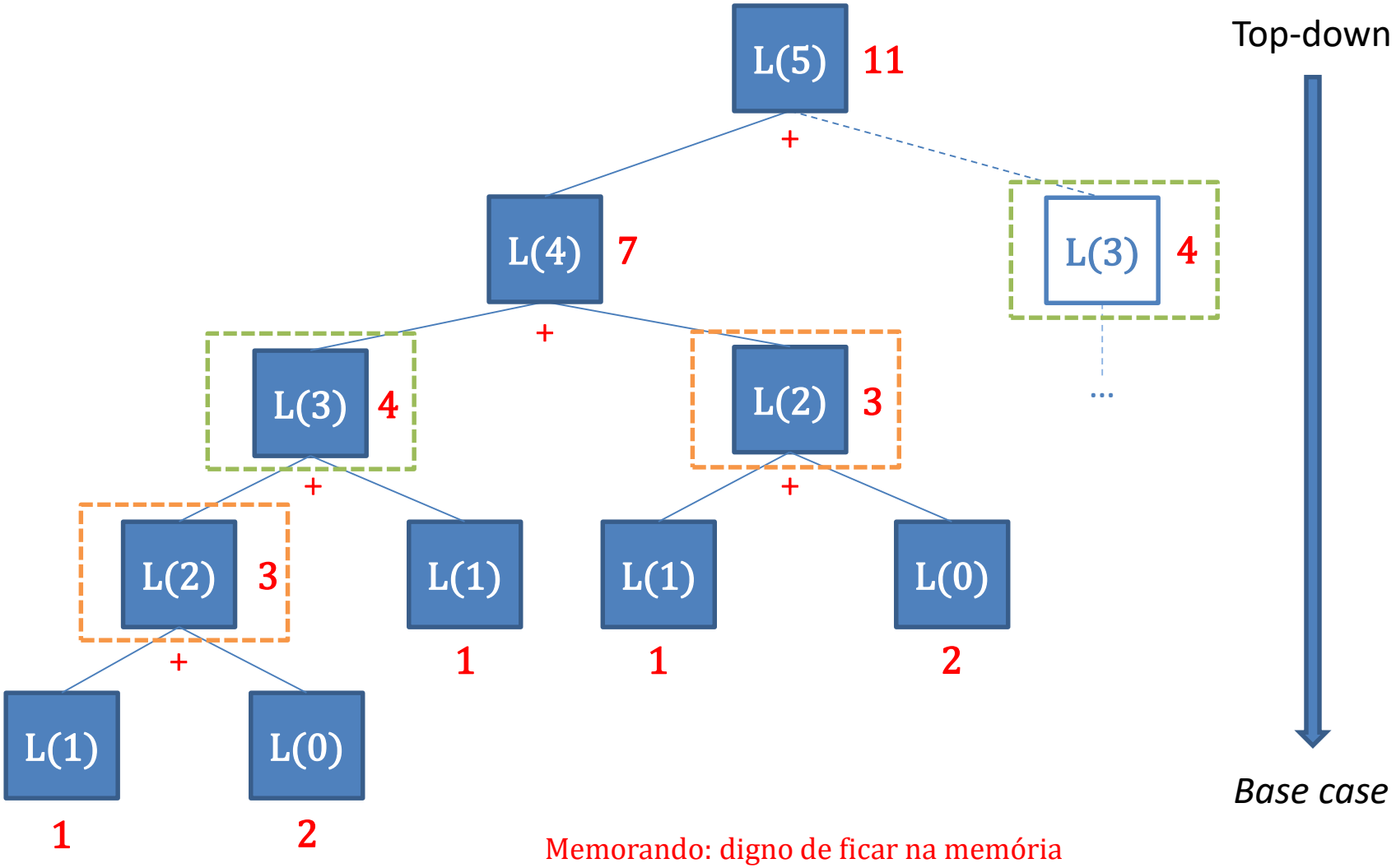
# *Memoization*

- Programação Dinâmica
  - *Memoization* armazena valores previamente computados para recuperar posteriormente (*caching*)

```
//dynamic programming: memoization
class lucas_with_memoization
{
    using map_type = std::map<short int, long long>;
    map_type memo;
public:
    long long operator()(short int n)
    {
        if (n == 0) return 2LL;
        if (n == 1) return 1LL;
        map_type::iterator it = memo.find(n);
        if (it != memo.end())
            return it->second;
        long long result = (operator()(n - 1) + operator()(n - 2));
        memo.emplace(n, result);
        return result;
    }
};
```

# *Memoization*

(*Overlapping Subproblems*)




Memorando: digno de ficar na memória

# *Merging*

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 11 | 13 | 15 | 18 | 19 |

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 10 | 12 | 14 | 16 | 17 |

- Juntar duas estruturas numa estrutura auxiliar
  - As estruturas, incluindo a auxiliar, devem possuir uma organização, por exemplo: ordernação ascendente
- A estrutura auxiliar pode ser:
  - Uma terceira estrutura, preservando as estruturas originais (*merge copy*)

    ou
  - Uma das estruturas originais (*inplace merge*)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

# merge

(requisito)

xs =

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 11 | 18 | 13 | 15 | 19 |

ys =

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 12 | 16 | 14 | 17 | 10 |

```
std::is_sorted(xs.begin(), xs.end()) ? false
```

```
std::is_sorted(ys.begin(), ys.end()) ? false
```

xs =

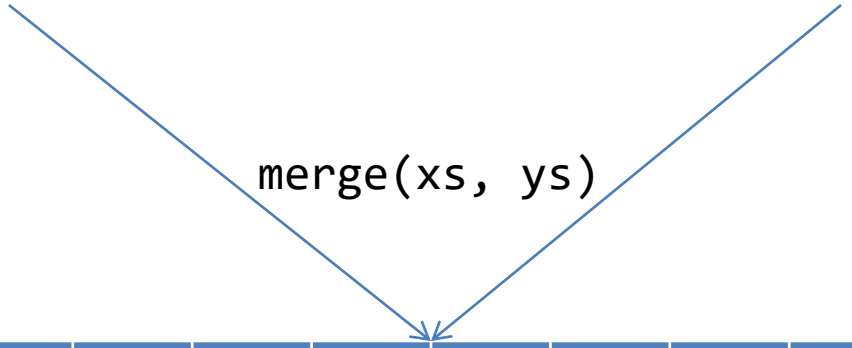
| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 11 | 13 | 15 | 18 | 19 |

ys =

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 10 | 12 | 14 | 16 | 17 |

```
std::is_sorted(xs.begin(), xs.end()) ? true
```

```
std::is_sorted(ys.begin(), ys.end()) ? true
```

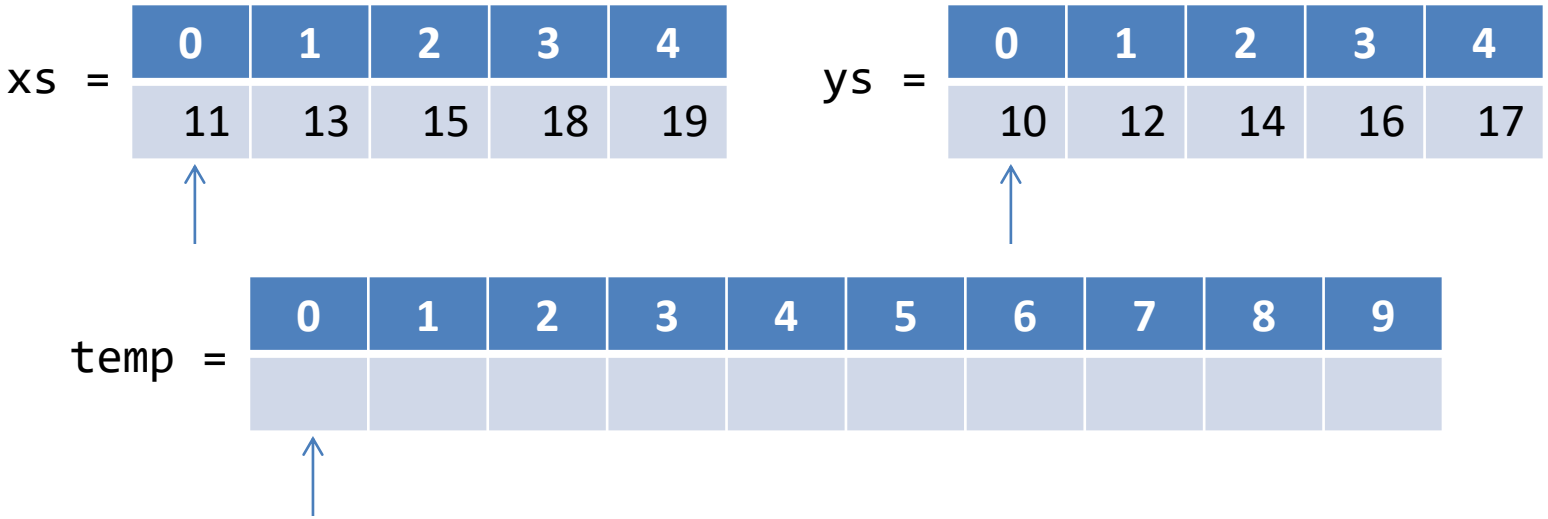


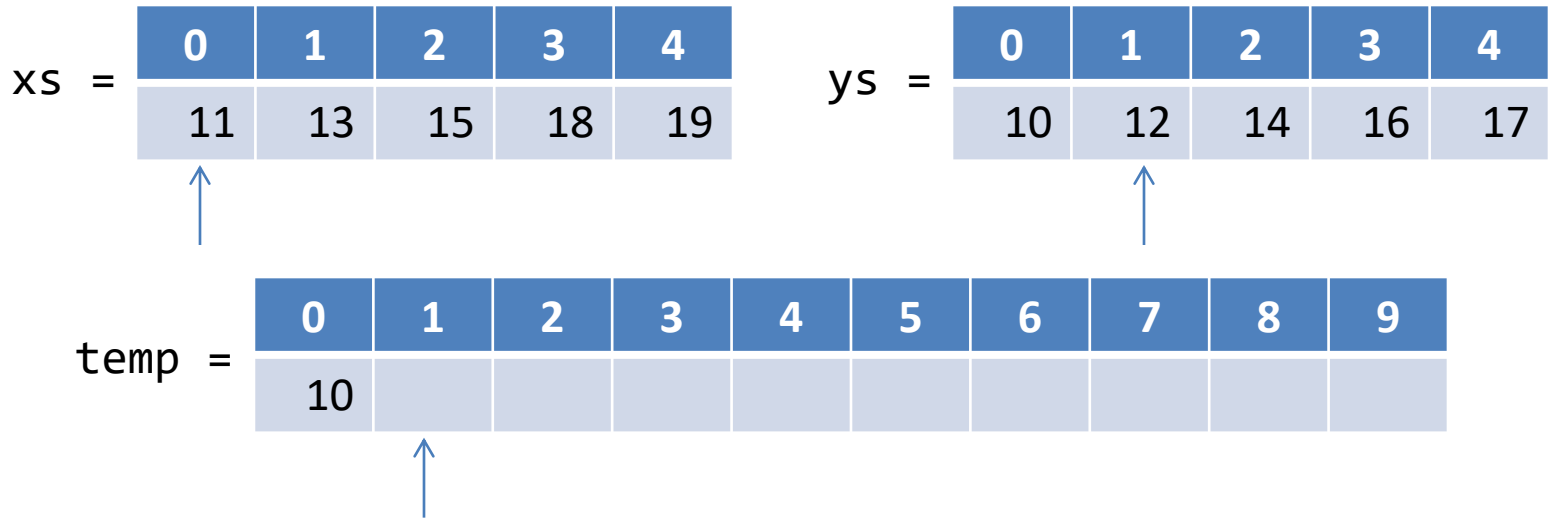

zs =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

# merge

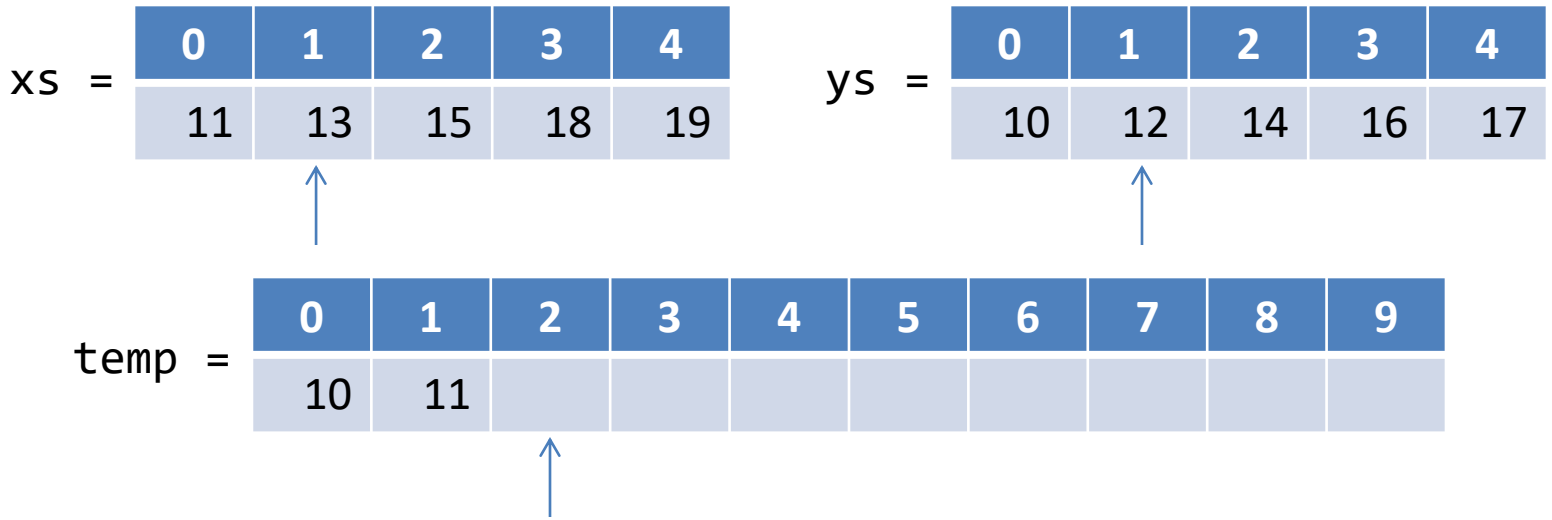
(passo a passo)

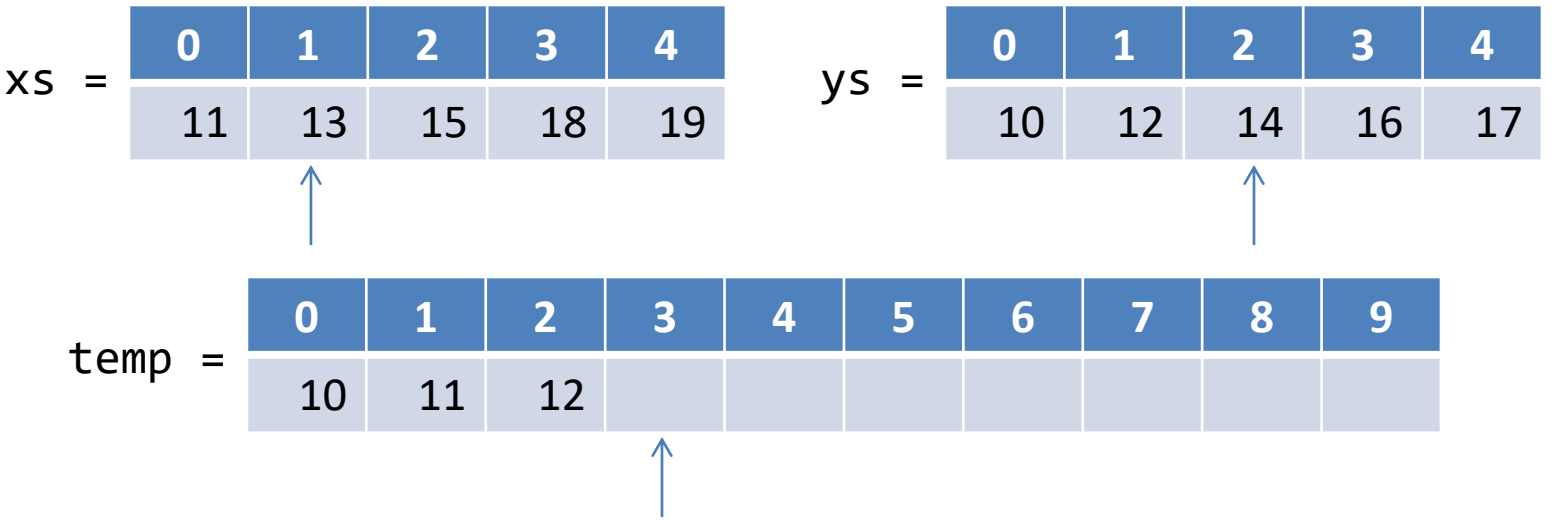


# merge

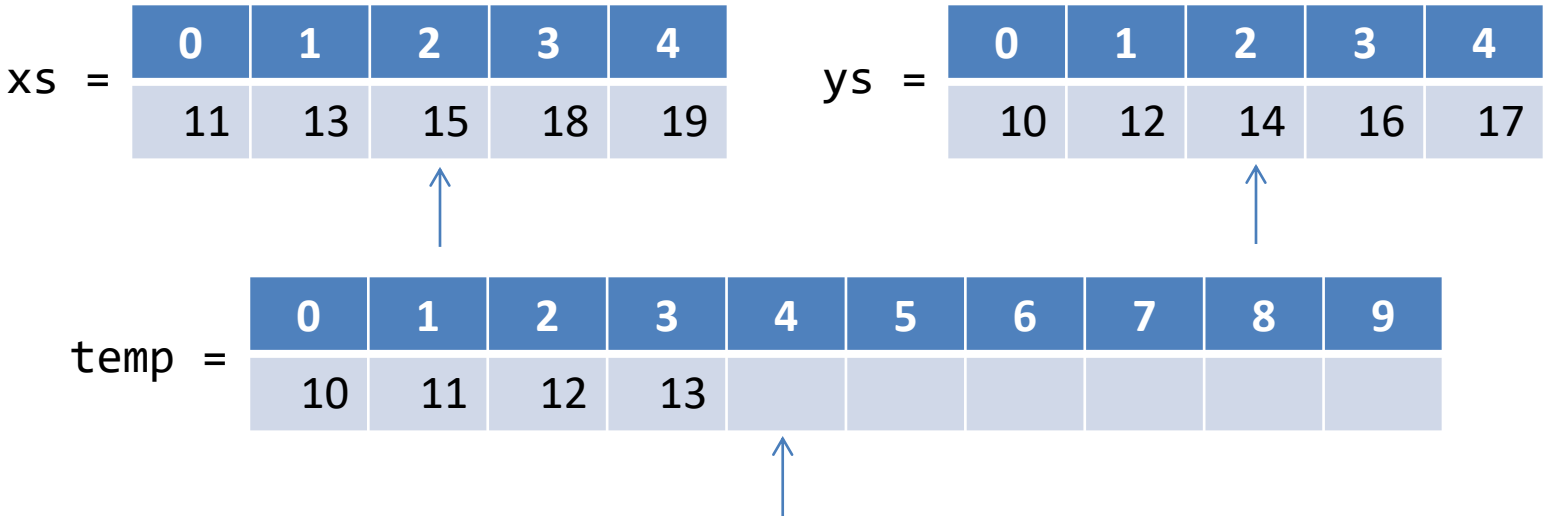
(passo a passo)

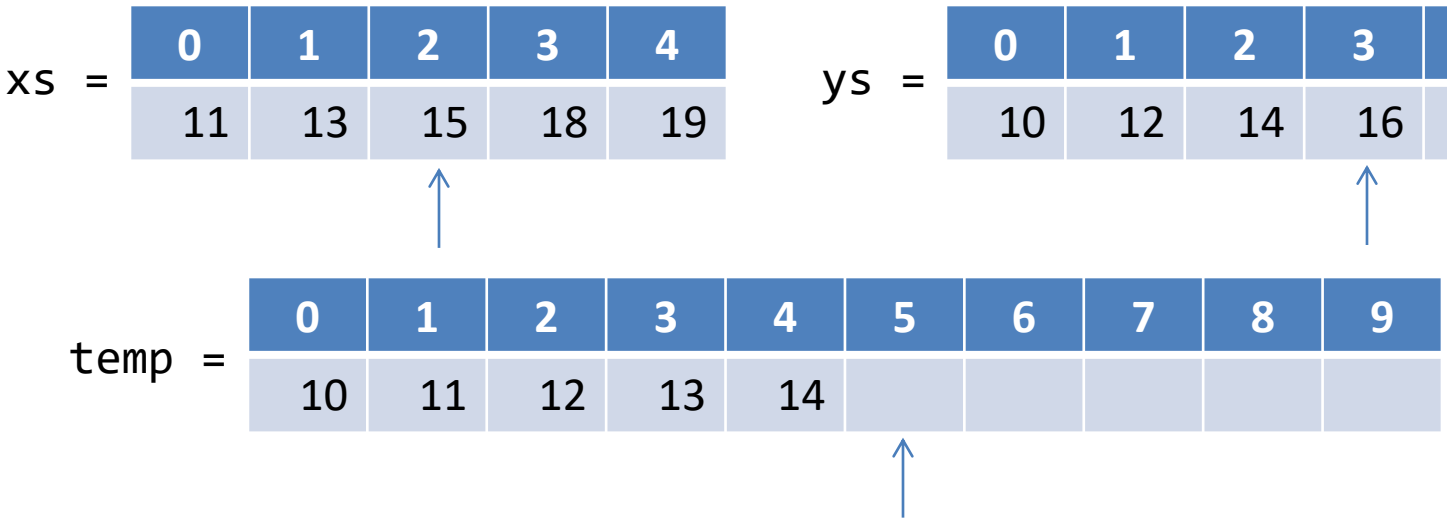


# merge

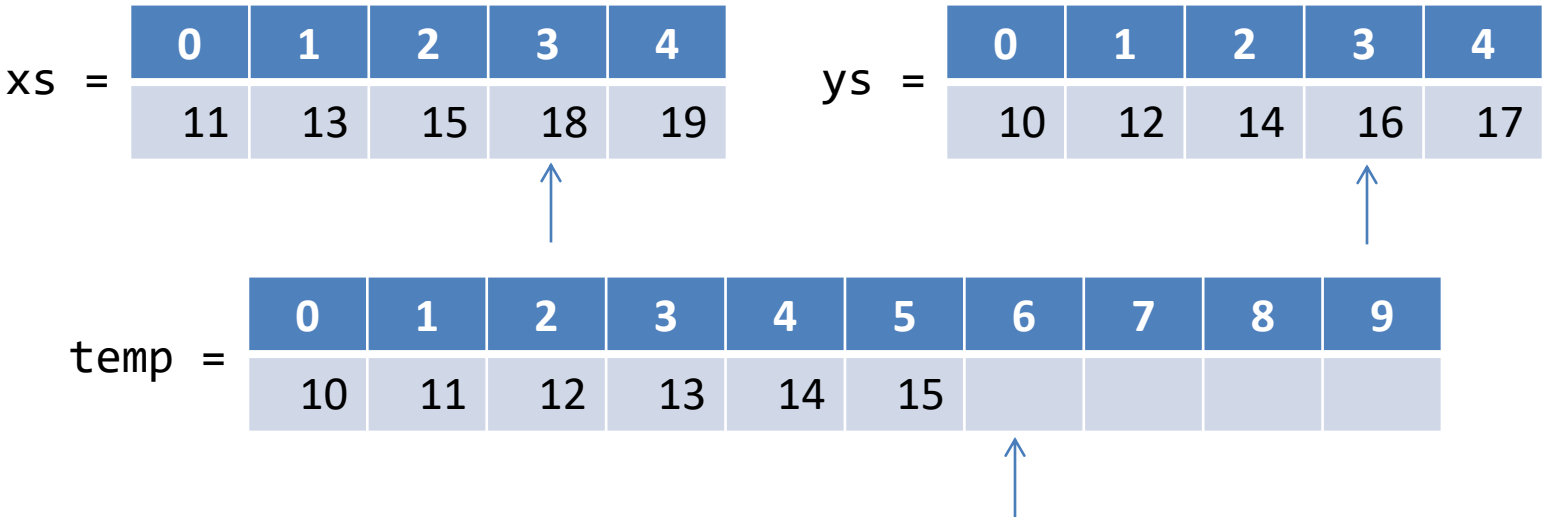
(passo a passo)



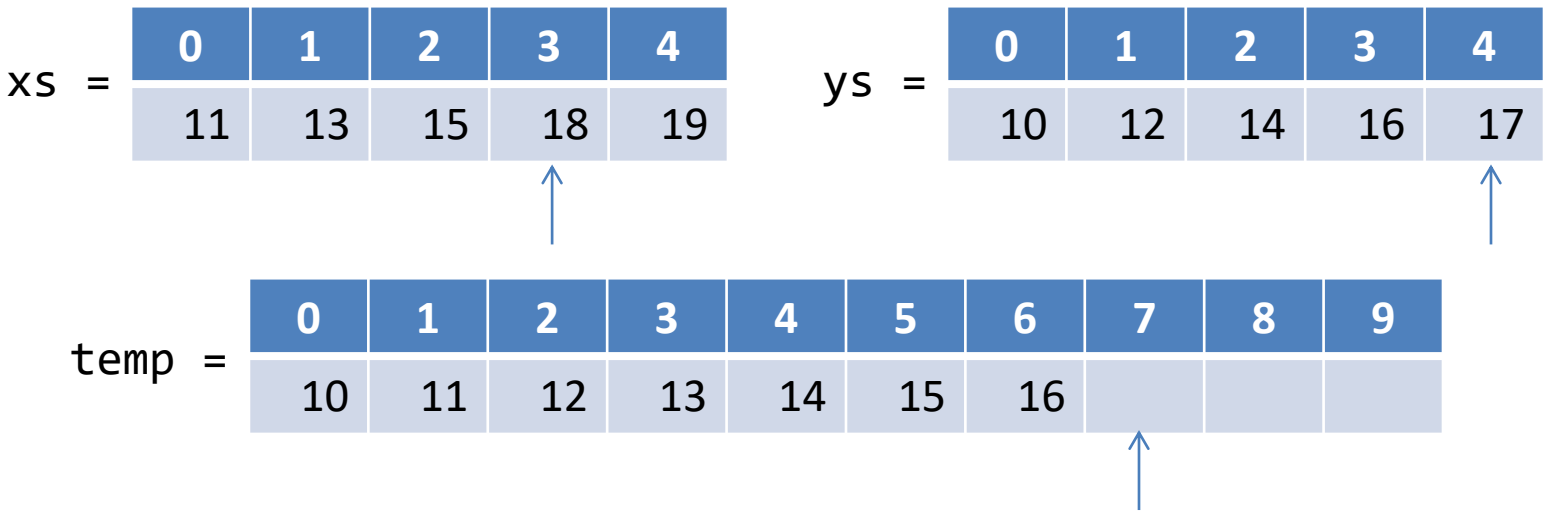



# merge

(passo a passo)

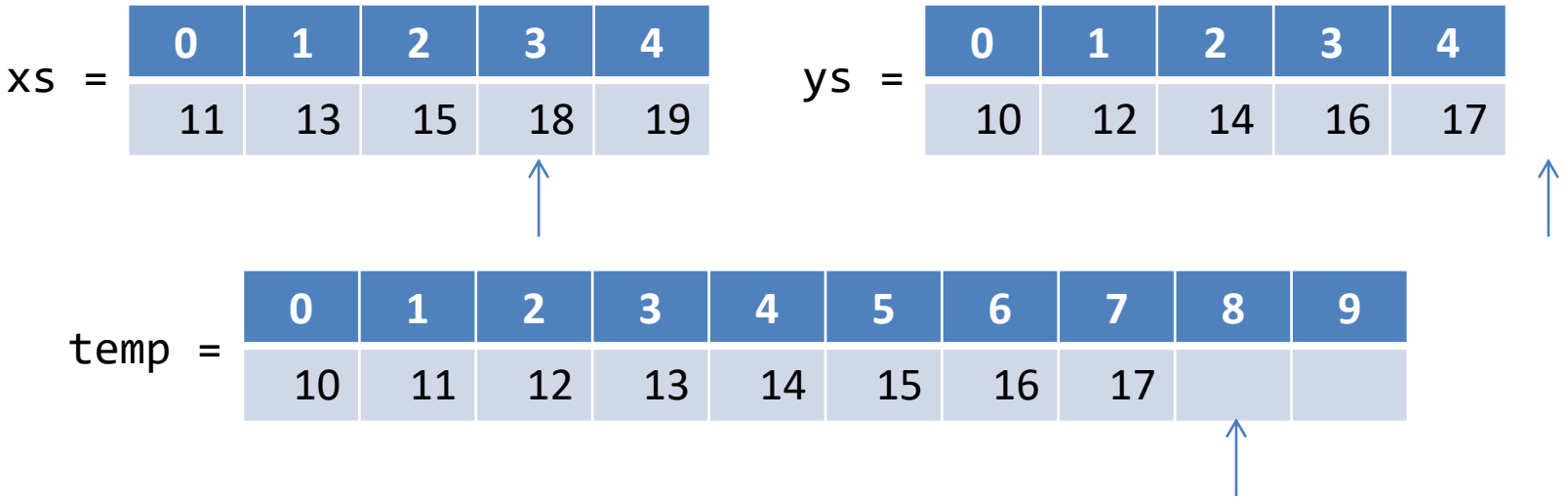


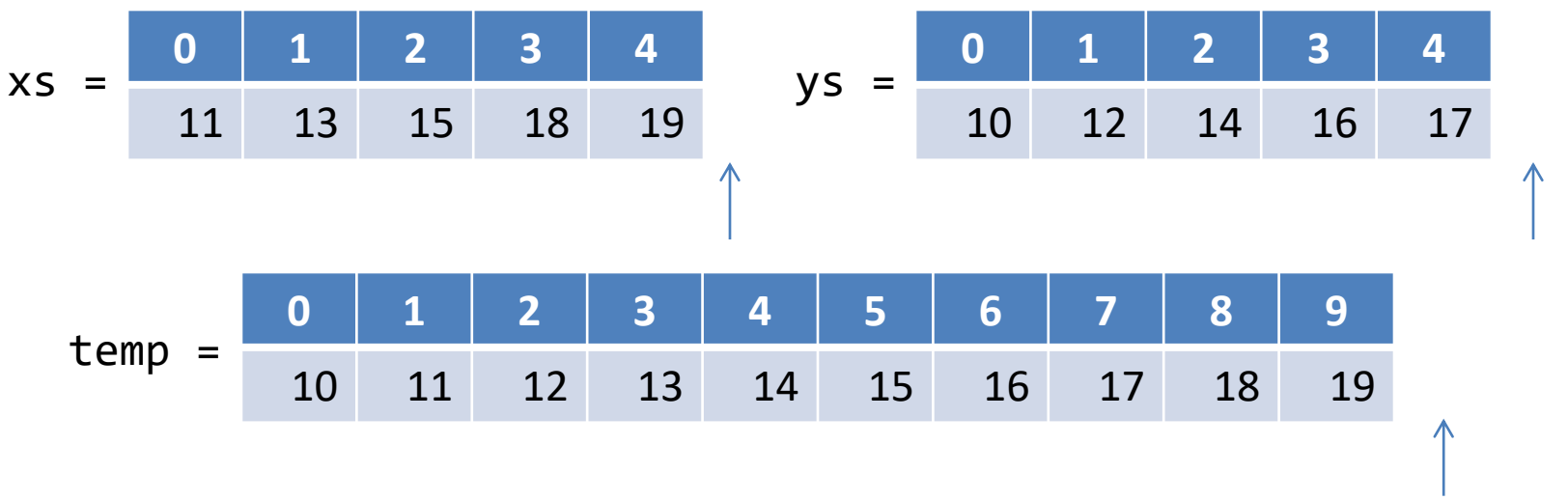

---



# merge

(passo a passo)



# merge_copy

```cpp
template<typename T>
static inline std::vector<T> merge_copy(const std::vector<T>& lhs, const std::vector<T>& rhs)
{
    assert(std::is_sorted(lhs.begin(), lhs.end()));
    assert(std::is_sorted(rhs.begin(), rhs.end()));

    const std::size_t N = lhs.size() + rhs.size();
    std::vector<T> temp;
    temp.reserve(N);
    std::size_t i = 0, j = 0;
    while (i + j < N)
    {
        if (i >= lhs.size())
        {
            std::copy(rhs.begin() + j, rhs.end(), std::back_inserter(temp));
            break;
        }
        else if (j >= rhs.size())
        {
            std::copy(lhs.begin() + i, lhs.end(), std::back_inserter(temp));
            break;
        }
        else if (rhs[j] < lhs[i])
        {
            temp.push_back(rhs[j++]);
        }
        else //if (rhs[j] >= lhs[i])
        {
            temp.push_back(lhs[i++]);
        }
    }
    return std::move(temp);
}
```

# std::merge

- InputIterator
  - std::vector<T>, std::list<T>
- OutputIterator
- Algoritmo com complexidade de tempo linear

```
4342  // merge
4343
4344  template <class _Compare, class _InputIterator1, class _InputIterator2, class _OutputIterator>
4345  _OutputIterator
4346  __merge(_InputIterator1 __first1, _InputIterator1 __last1,
4347          _InputIterator2 __first2, _InputIterator2 __last2, _OutputIterator __result, _Compare __comp)
4348  {
4349      for (; __first1 != __last1; ++__result)
4350      {
4351          if (__first2 == __last2)
4352              return _VSTD::copy(__first1, __last1, __result);
4353          if (__comp(*__first2, *__first1))
4354          {
4355              *__result = *__first2;
4356              ++__first2;
4357          }
4358          else
4359          {
4360              *__result = *__first1;
4361              ++__first1;
4362          }
4363      }
4364      return _VSTD::copy(__first2, __last2, __result);
4365  }
```

$O(N)$

```
4377  template <class _InputIterator1, class _InputIterator2, class _OutputIterator>
4378  inline _LIBCPP_INLINE_VISIBILITY
4379  _OutputIterator
4380  merge(_InputIterator1 __first1, _InputIterator1 __last1,
4381        _InputIterator2 __first2, _InputIterator2 __last2, _OutputIterator __result)
4382  {
4383      typedef typename iterator_traits<_InputIterator1>::value_type __v1;
4384      typedef typename iterator_traits<_InputIterator2>::value_type __v2;
4385      return _VSTD::merge(__first1, __last1, __first2, __last2, __result, __less<__v1, __v2>());
4386  }
```

https://github.com/llvm-mirror/libcxx/blob/master/include/algorithm#L4342

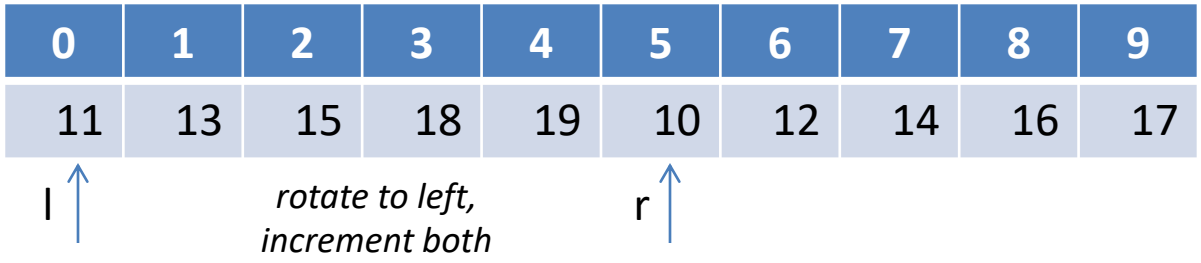
# inplace_merge

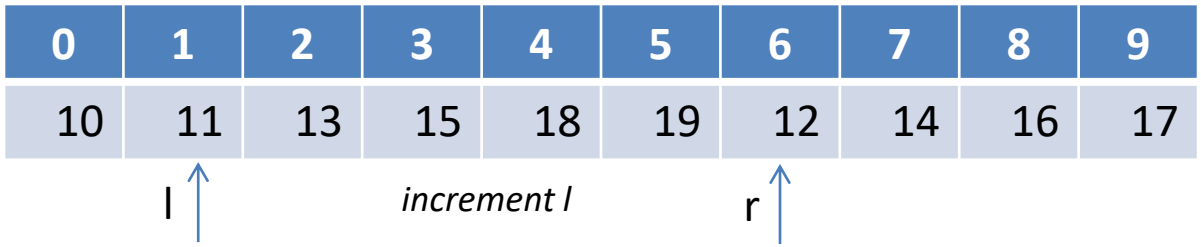
- Evitar cópia (*buffer* extra)

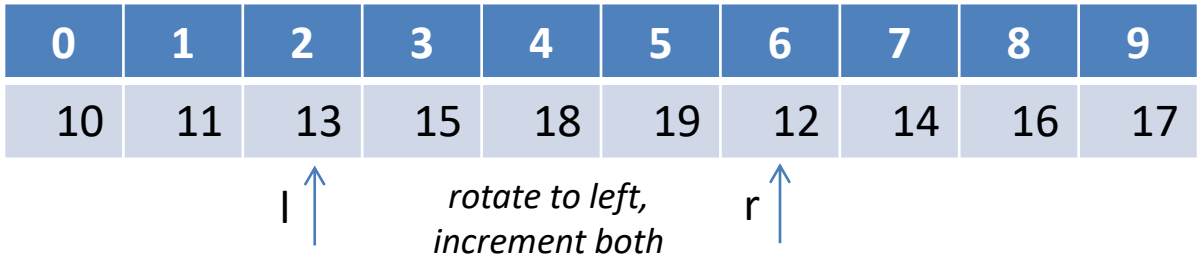
```
template<typename T>
static inline void inplace_merge(std::vector<T>& xs, std::size_t first, std::size_t last)
{
    assert(first < last);
    const std::size_t mid = first + (last - first) / 2;
    std::size_t l = first, r = mid;
    assert(std::is_sorted(xs.begin(), xs.begin() + mid));
    assert(std::is_sorted(xs.begin() + mid, xs.end()));

    if (mid > 0)
    {
        while (l < r && r < last)
        {
            if (xs[r] < xs[l])
            {
                T* l_ptr = &xs[l];
                T* r_ptr = &xs[r];
                //rotate [l_ptr, r_ptr] to left distance(l_ptr, r_ptr) times
                std::rotate(l_ptr, r_ptr, r_ptr + 1);
                ++r;
            }
            ++l;
        }
    }

    assert(std::is_sorted(xs.begin(), xs.end()));
}
```

# inplace_merge

(passo a passo)

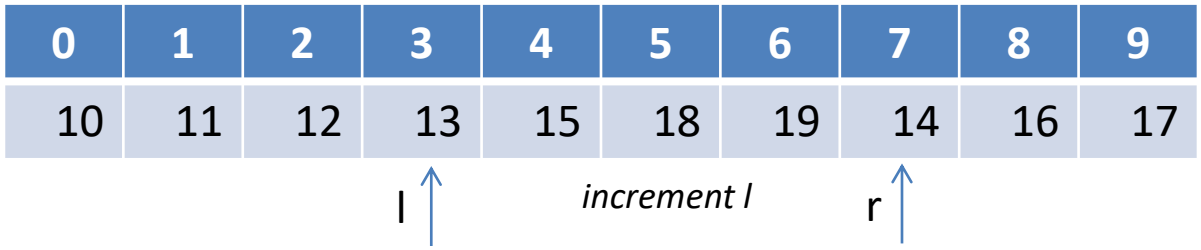


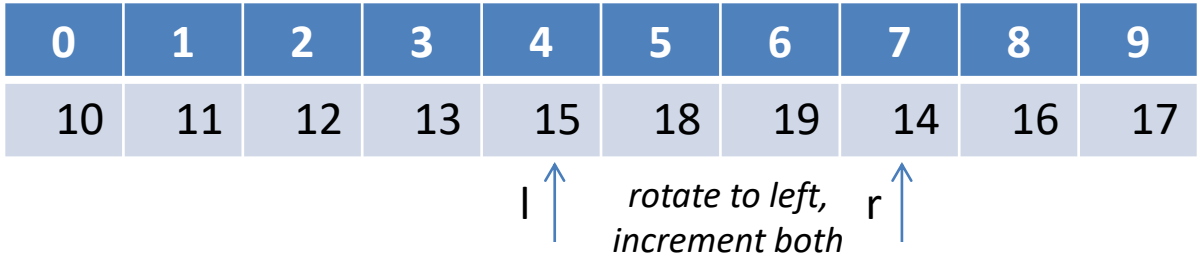



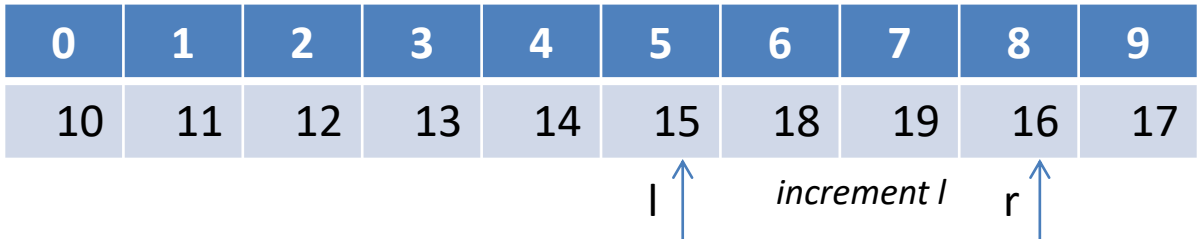



# inplace_merge

(passo a passo)

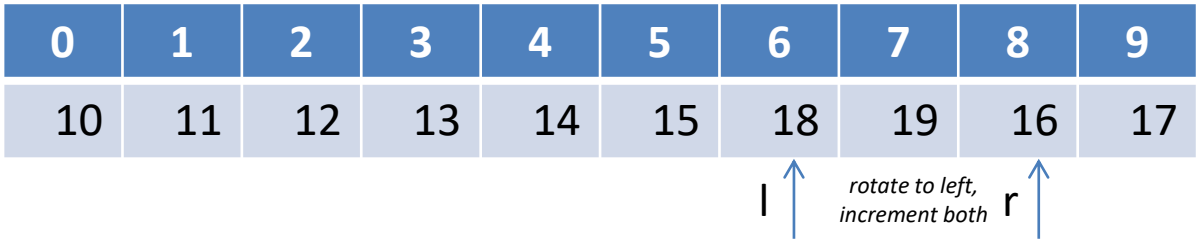


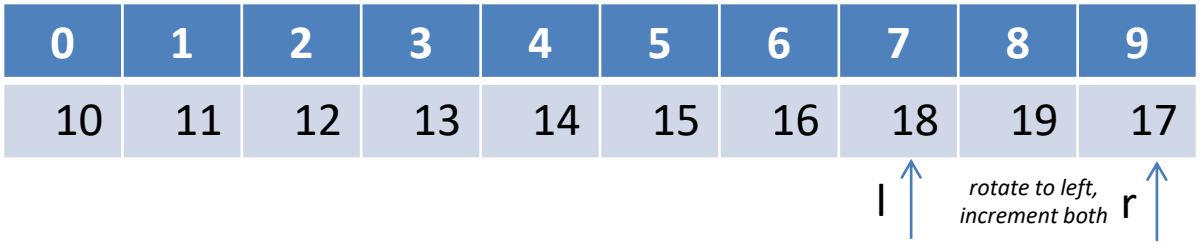



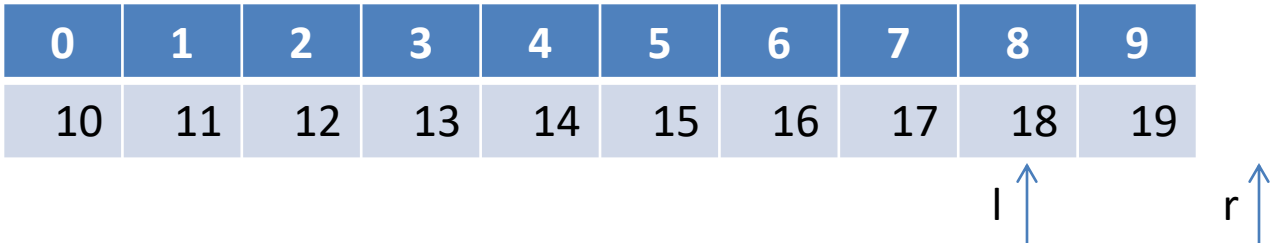



# inplace_merge

(passo a passo)

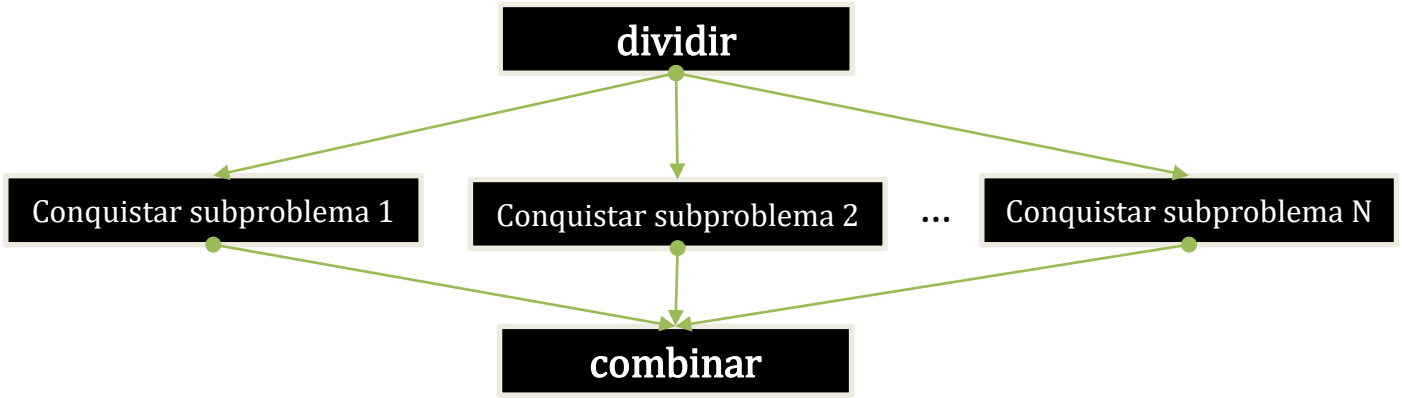


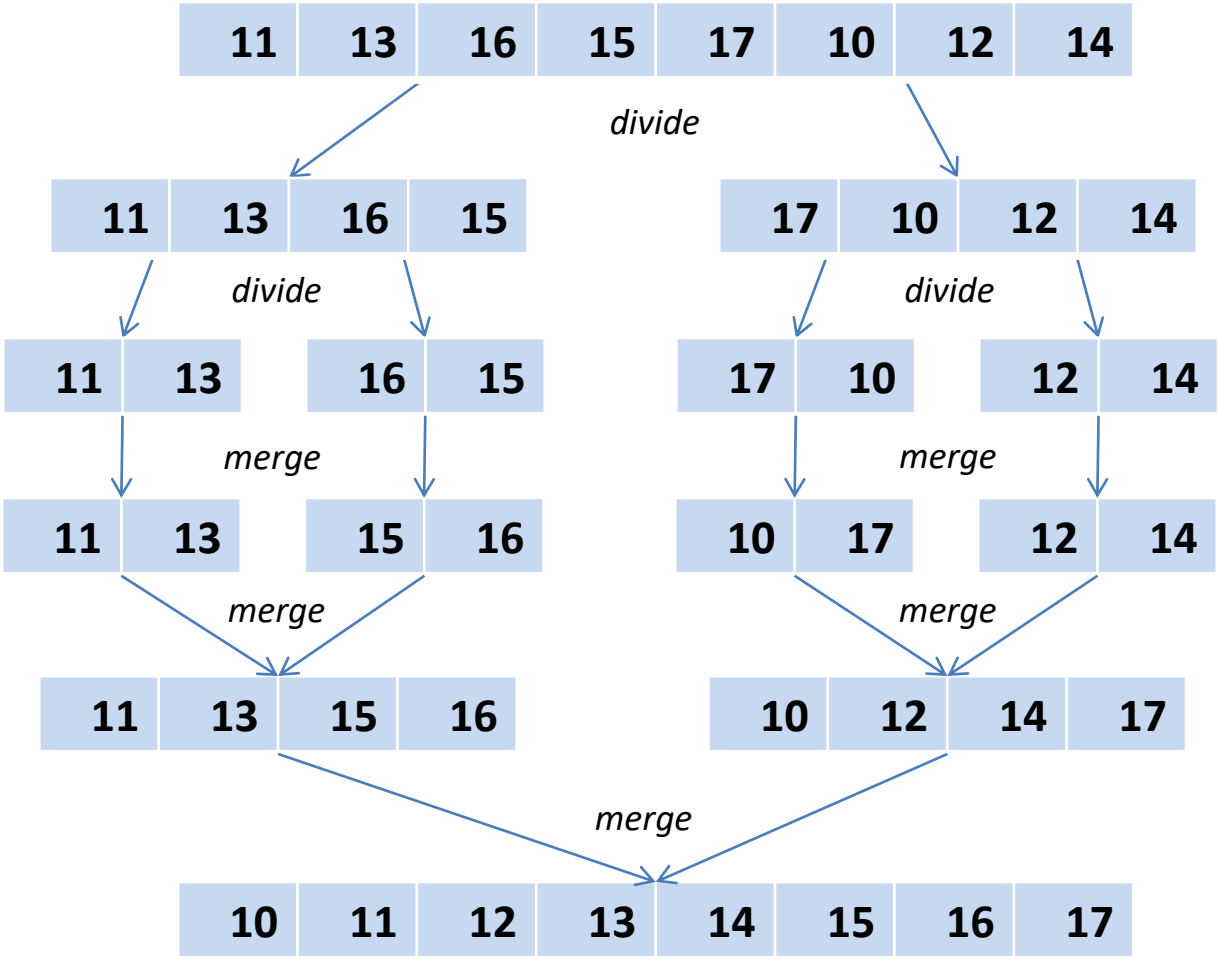
# Dividir e Conquistar

- Estratégia para resolução de problemas, onde um problema é divido em subproblemas que são solucionandos recursivamente
  - **Dividir** o problema em um ou mais subproblemas (menores que o problema original)
    - Se o problema for pequeno suficiente, resolvê-lo diretamente (caso base)
  - **Conquistar** os subproblemas, resolvendo-os recursivamente
    - (caso recursivo)
  - **Combinar** as soluções dos subproblemas para formar a solução do problema original
- *Fork-Join*, *Map-Reduce*



# mergesort
(ordenação)

- Dividir e conquistar



# mergesort

```
template <typename Container>
static inline void mergesort(Container& xs, std::size_t first, std::size_t last)
{
    assert(first <= last);

    const std::size_t N = last - first;
    if (N < 2)
    {
        return;
    }
    if (N == 2)
    {
        if (xs[first + 1] < xs[first])
            std::swap(xs[first + 1], xs[first]);
        return;
    }

    const std::size_t mid = N / 2;
    mergesort(xs, first, first + mid);
    mergesort(xs, first + mid, last);

    using ptr_type = typename Container::pointer;
    ptr_type beg_ptr = &xs[first];
    ptr_type mid_ptr = beg_ptr + mid;
    ptr_type end_ptr = beg_ptr + N;
    std::inplace_merge(beg_ptr, mid_ptr, end_ptr);
}
```

# std::stable_sort

```
4692  template <class _Compare, class _RandomAccessIterator>
4693  void
4694  __stable_sort(_RandomAccessIterator __first, _RandomAccessIterator __last, _Compare __comp,
4695                typename iterator_traits<_RandomAccessIterator>::difference_type __len,
4696                typename iterator_traits<_RandomAccessIterator>::value_type* __buff, ptrdiff_t __buff_size)
4697  {
4698      typedef typename iterator_traits<_RandomAccessIterator>::value_type value_type;
4699      typedef typename iterator_traits<_RandomAccessIterator>::difference_type difference_type;
4700      switch (__len)
4701      {
4702      case 0:
4703      case 1:
4704          return;
4705      case 2:
4706          if (__comp(*--__last, *__first))
4707              swap(*__first, *__last);
4708          return;
4709      }
4710      if (__len <= static_cast<difference_type>(__stable_sort_switch<value_type>::value))
4711      {
4712          __insertion_sort<_Compare>(__first, __last, __comp);
4713          return;
4714      }
4715      typename iterator_traits<_RandomAccessIterator>::difference_type __l2 = __len / 2;
4716      _RandomAccessIterator __m = __first + __l2;
4717      if (__len <= __buff_size)
4718      {
4719          __destruct_n __d(0);
4720          unique_ptr<value_type, __destruct_n&> __h2(__buff, __d);
4721          __stable_sort_move<_Compare>(__first, __m, __comp, __l2, __buff);
4722          __d.__set(__l2, (value_type*)0);
4723          __stable_sort_move<_Compare>(__m, __last, __comp, __len - __l2, __buff + __l2);
4724          __d.__set(__len, (value_type*)0);
4725          __merge_move_assign<_Compare>(__buff, __buff + __l2, __buff + __l2, __buff + __len, __first, __comp);
4726  //          __merge<_Compare>(move_iterator<value_type*>(__buff),
4727  //                            move_iterator<value_type*>(__buff + __l2),
4728  //                            move_iterator<_RandomAccessIterator>(__buff + __l2),
4729  //                            move_iterator<_RandomAccessIterator>(__buff + __len),
4730  //                            __first, __comp);
4731          return;
4732      }
4733      __stable_sort<_Compare>(__first, __m, __comp, __l2, __buff, __buff_size);
4734      __stable_sort<_Compare>(__m, __last, __comp, __len - __l2, __buff, __buff_size);
4735      __inplace_merge<_Compare>(__first, __m, __last, __comp, __l2, __len - __l2, __buff, __buff_size);
4736  }
```

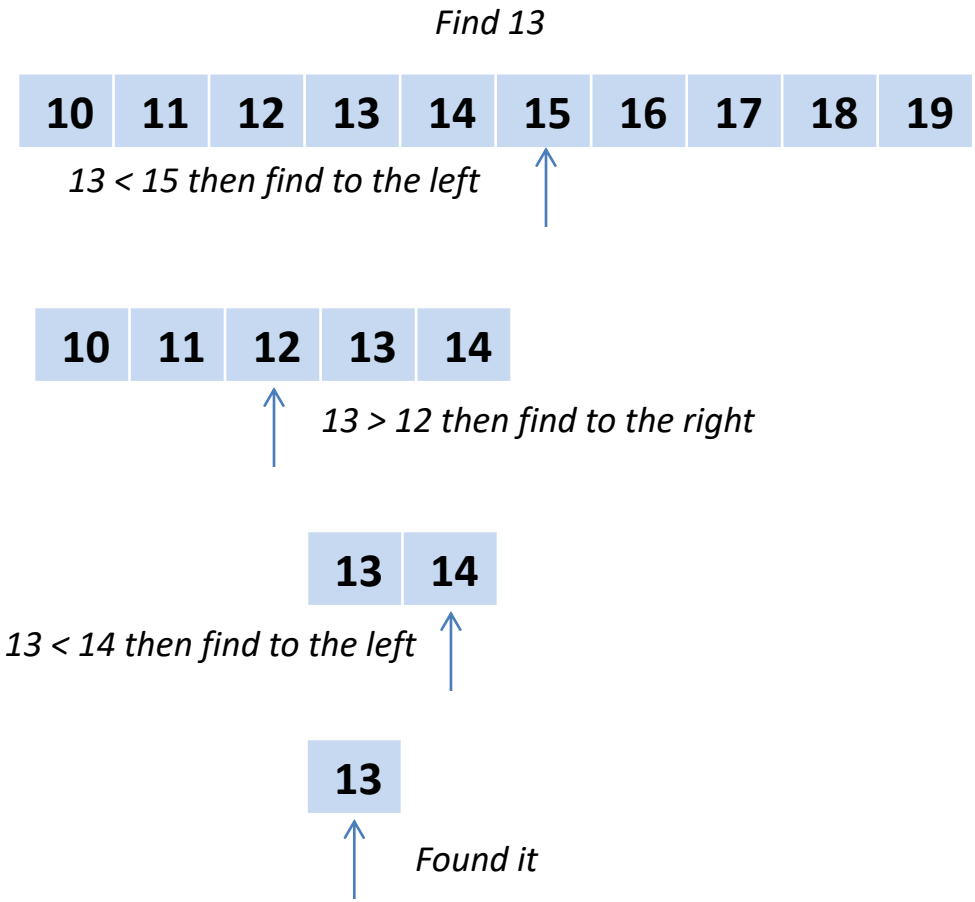
$$O(N \times lgN)$$

https://github.com/llvm-mirror/libcxx/blob/master/include/algorithm#L4694

# `binary_search`

(algoritmo de busca binária)

- *Container* deve estar ordenado
- Inicia as buscas a partir do ponto médio



# binary_search

```
template <typename Container, typename Comparer>
std::size_t binary_search(const Container& xs, const typename Container::value_type& x, Comparer compare)
{
    std::size_t l = 0, r = xs.size(), mid;
    while (l < r)
    {
        //mid = (r + l) / 2; //overflow case
        mid = l + (r - l) / 2; //no overflow case

        //x == xs[mid] -> 0
        //x <  xs[mid] -> < 0 (usually -1)
        //x >  xs[mid] -> > 0 (usually +1)
        int comp = compare(x, xs[mid]);
        if (comp == 0)
            return mid;
        else if (comp < 0)
            r = mid;
        else /* if (comp > 0) */
            l = mid + 1;
    }
    return NOT_FOUND;
}
```

# std::lower_bound

- binary_search_leftmost

```
4167  // lower_bound
4168
4169  template <class _Compare, class _ForwardIterator, class _Tp>
4170  _LIBCPP_CONSTEXPR_AFTER_CXX17 _ForwardIterator
4171  __lower_bound(_ForwardIterator __first, _ForwardIterator __last, const _Tp& __value_, _Compare __comp)
4172  {
4173      typedef typename iterator_traits<_ForwardIterator>::difference_type difference_type;
4174      difference_type __len = _VSTD::distance(__first, __last);
4175      while (__len != 0)
4176      {
4177          difference_type __l2 = _VSTD::__half_positive(__len);
4178          _ForwardIterator __m = __first;
4179          _VSTD::advance(__m, __l2);
4180          if (__comp(*__m, __value_))
4181          {
4182              __first = ++__m;
4183              __len -= __l2 + 1;
4184          }
4185          else
4186              __len = __l2;
4187      }
4188      return __first;
4189  }
```

$O(lgN)$

https://github.com/llvm-mirror/libcxx/blob/master/include/algorithm#L4167

# std::binary_search

- Implementado em termos de `std::lower_bound`

```
// binary_search

template <class _Compare, class _ForwardIterator, class _Tp>
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_CONSTEXPR_AFTER_CXX17
bool
__binary_search(_ForwardIterator __first, _ForwardIterator __last, const _Tp& __value_, _Compare __comp)
{
    __first = __lower_bound<_Compare>(__first, __last, __value_, __comp);
    return __first != __last && !__comp(__value_, *__first);
}

template <class _ForwardIterator, class _Tp, class _Compare>
_LIBCPP_NODISCARD_EXT inline
_LIBCPP_INLINE_VISIBILITY _LIBCPP_CONSTEXPR_AFTER_CXX17
bool
binary_search(_ForwardIterator __first, _ForwardIterator __last, const _Tp& __value_, _Compare __comp)
{
    typedef typename __comp_ref_type<_Compare>::type _Comp_ref;
    return __binary_search<_Comp_ref>(__first, __last, __value_, __comp);
}
```

$O(lgN)$

https://github.com/llvm-mirror/libcxx/blob/master/include/algorithm#L4311

# between

- Implementado em termos de `std::lower_bound` e de `std::upper_bound`

```
template <typename Container>
std::tuple<typename Container::const_iterator, typename Container::const_iterator>
    between(const Container& xs, const typename Container::value_type& lhs, const typename Container::value_type& rhs)
{
    //assert(lhs <= rhs);
    //assert(std::is_sorted(xs.cbegin(), xs.cend()));
    auto first = std::lower_bound(xs.cbegin(), xs.cend(), lhs);
    auto last = std::upper_bound(xs.cbegin(), xs.cend(), rhs);
    return std::make_tuple(first, last);
}
```

$O(lgN)$

```
collection size: 12
int: find equals, int,int: find between, ?: display collection
?
1 1 1 2 2 3 5 5 5 6 6 6
2,5
found at [3, 9)
2 2 3 5 5 5
```

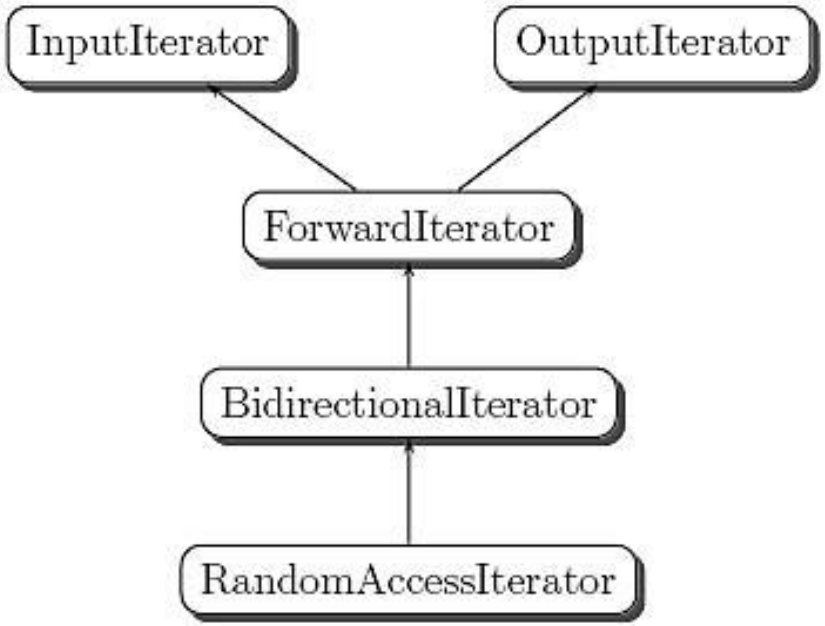
# *Iterators*

- Generalização de ponteiro
- Um objeto que aponta para algum elemento num *container*, *array* ou sequência
- Um par de iterators representa um intervalo fechado-aberto [*begin*, *end*)
- 5 categorias
  - `InputIterator`
  - `OutputIterator`
  - `ForwardIterator`
  - `BidirectionalIterator`
  - `RandomAccessIterator`
    - Um ponteiro tem todos os requisitos de um `RandomAccessIterator`

# *Iterators*

(hierarquia)

- Hierarquia dos *iterators* na STL




- Se uma função espera receber um `InputIterator` ou um `OutputIterator`, ela também aceitará um:
  - `ForwardIterator`,
  - `BidirectionalIterator` ou
  - `RandomAccessIterator`

# *Iterators*

(propriedades)

| category | | | | properties | valid expressions |
|---|---|---|---|---|---|
| all categories | | | | *copy-constructible*, *copy-assignable* and *destructible* | `X b(a);`<br>`b = a;` |
| | | | | Can be incremented | `++a`<br>`a++` |
| Random Access | Bidirectional | Forward | Input | Supports equality/inequality comparisons | `a == b`<br>`a != b` |
| | | | | Can be dereferenced as an *rvalue* | `*a`<br>`a->m` |
| | | | Output | Can be dereferenced as an *lvalue* (only for *mutable iterator types*) | `*a = t`<br>`*a++ = t` |
| | | | | *default-constructible* | `X a;`<br>`X()` |
| | | | | Multi-pass: neither dereferencing nor incrementing affects dereferenceability | `{ b=a; *a++; *b; }` |
| | | | | Can be decremented | `--a`<br>`a--`<br>`*a--` |
| | | | | Supports arithmetic operators + and - | `a + n`<br>`n + a`<br>`a - n`<br>`a - b` |
| | | | | Supports inequality comparisons (<, >, <= and >=) between iterators | `a < b`<br>`a > b`<br>`a <= b`<br>`a >= b` |
| | | | | Supports compound assignment operations += and -= | `a += n`<br>`a -= n` |
| | | | | Supports offset dereference operator ([]) | `a[n]` |

http://www.cplusplus.com/reference/iterator/

# *String Utilities*

(std::string)

```
template<typename UnaryPredicate>
static inline std::string& strip_left_if(std::string& source, UnaryPredicate is_space)
{
    auto first = source.begin();
    auto is_not_space = std::not1(std::function<bool(char)>(is_space));
    auto last = std::find_if(source.begin(), source.end(), is_not_space);
    source.erase(first, last);
    return source;
}

static inline std::string& strip_left(std::string& source)
{
    return strip_left_if(source, [](int ch) { return std::isspace(ch); });
}
```

$O(N)$

# *String Utilities*

(std::vector)

```
template<typename UnaryPredicate>
static inline std::vector<char>& strip_left_if(std::vector<char>& source, UnaryPredicate is_space)
{
    auto first = source.begin();
    auto is_not_space = std::not1(std::function<bool(char)>(is_space));
    auto last = std::find_if(source.begin(), source.end(), is_not_space);
    auto n = std::distance(first, last);
    if (n)
    {
        std::move(last, source.end(), first);
        source.resize(source.size() - n);
    }
    return source;
}

static inline std::vector<char>& strip_left(std::vector<char>& source)
{
    return strip_left_if(source, [](int ch) { return std::isspace(ch); });
}
```

$O(N)$

# *String Utilities*

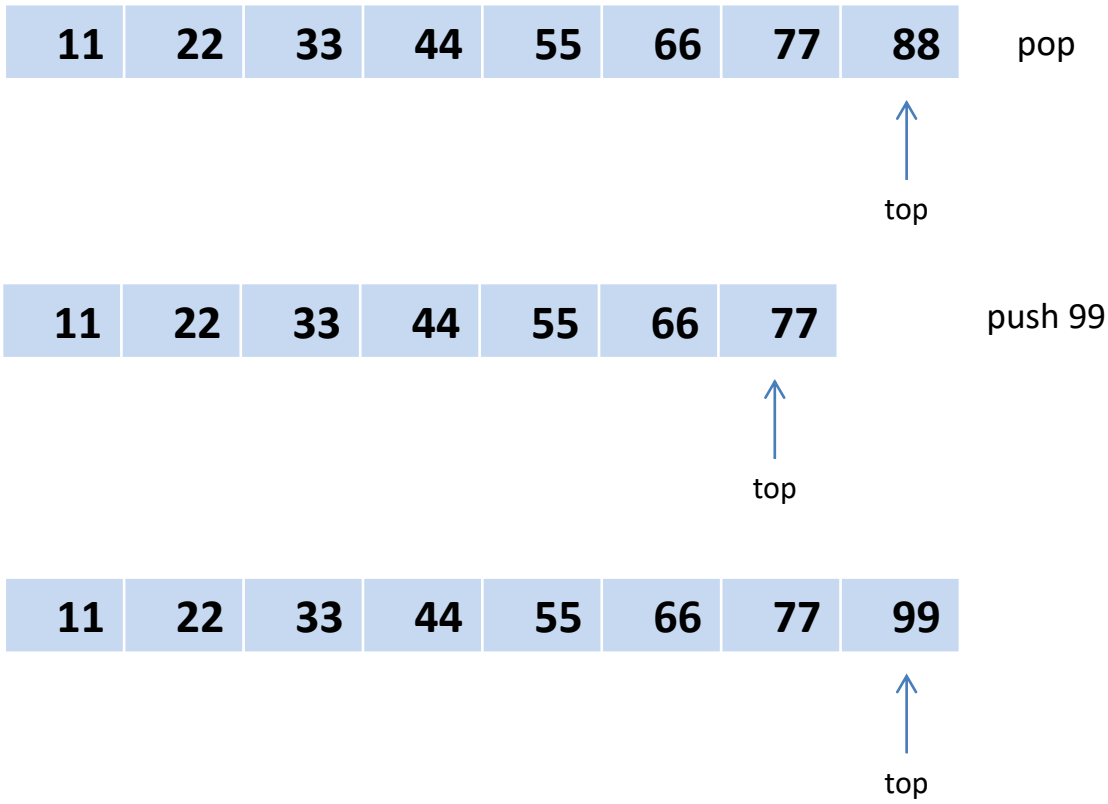
## (Iterators)

```
template<typename InputIterator, typename OutputIterator, typename UnaryPredicate>
static inline OutputIterator strip_left_if(InputIterator first, InputIterator last, OutputIterator result, UnaryPredicate predicate)
{
    using T = typename std::iterator_traits<InputIterator>::value_type;
    auto not_predicate = std::not1(std::function<bool(T)>(predicate));
    return std::copy(std::find_if(first, last, not_predicate), last, result);
}

template<typename BidirectionalIterator, typename OutputIterator, typename UnaryPredicate>
static inline OutputIterator strip_right_if(BidirectionalIterator first, BidirectionalIterator last, OutputIterator result, UnaryPredicate predicate)
{
    while (first != last)
    {
        last = std::prev(last);
        if (!predicate(*last))
            break;
    }
    if (std::distance(first, last) && !predicate(*last))
        last = std::next(last);
    return std::copy(first, last, result);
}

template<typename BidirectionalIterator, typename UnaryPredicate>
static inline BidirectionalIterator strip_if(BidirectionalIterator first, BidirectionalIterator last, UnaryPredicate predicate)
{
    return string_utilities::iterator_support::strip_right_if
           (
                first,
                string_utilities::iterator_support::strip_left_if(first, last, first, predicate),
                first,
                predicate
           );
}
```

$O(N)$

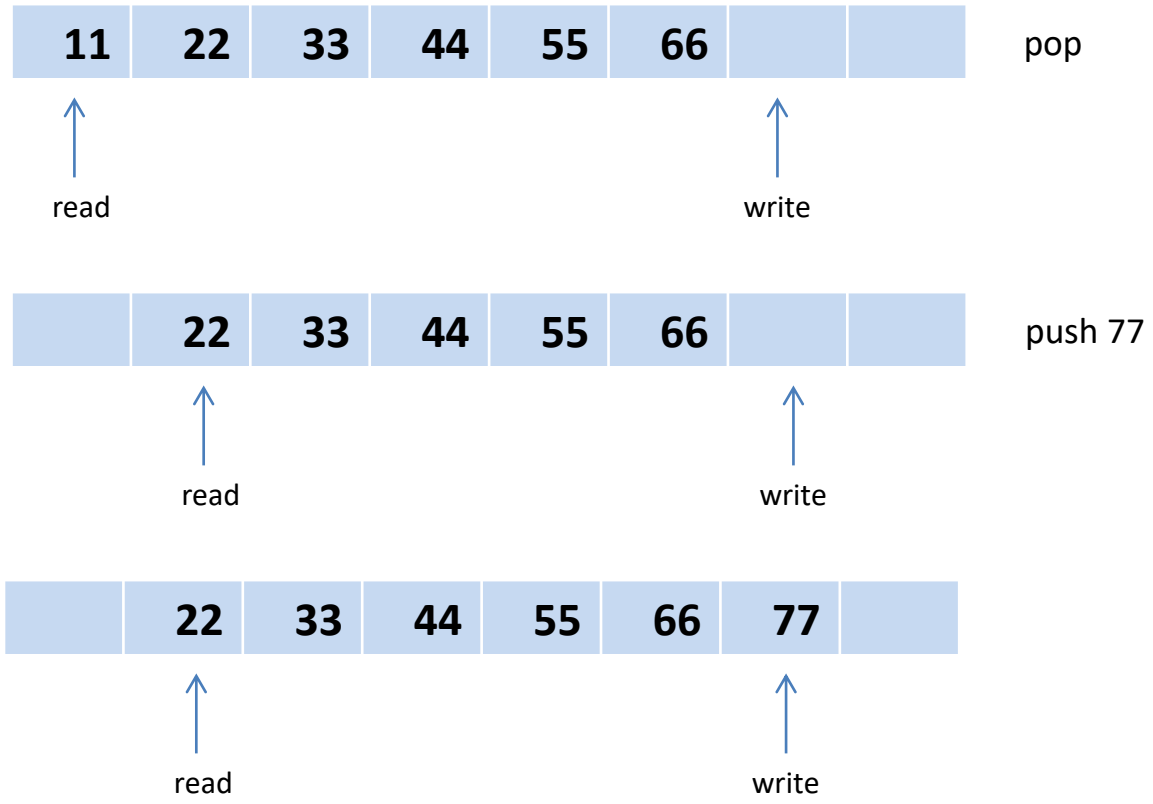
# Pilha

- Estrutura de Dados baseada em *LIFO*
  - *Last-In First-Out*



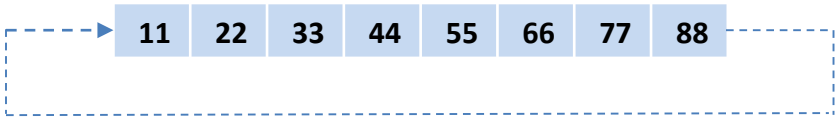
# Fila

- Estrutura de Dados baseada em *FIFO*
  - *First-In First-Out*

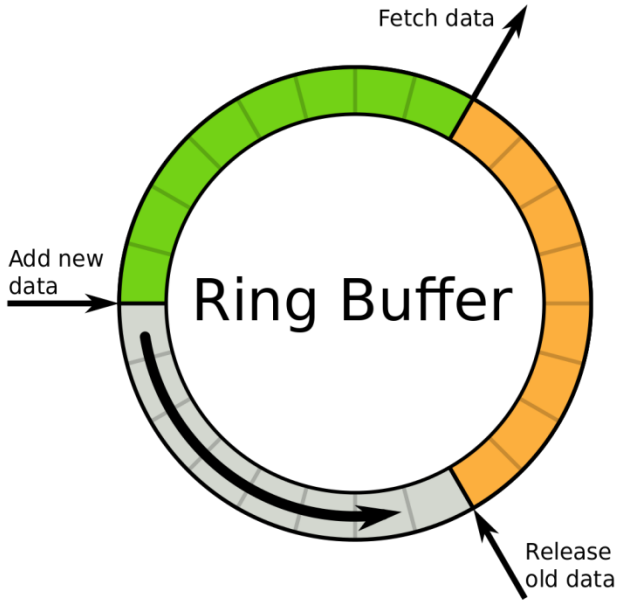


# *Ring Buffer*

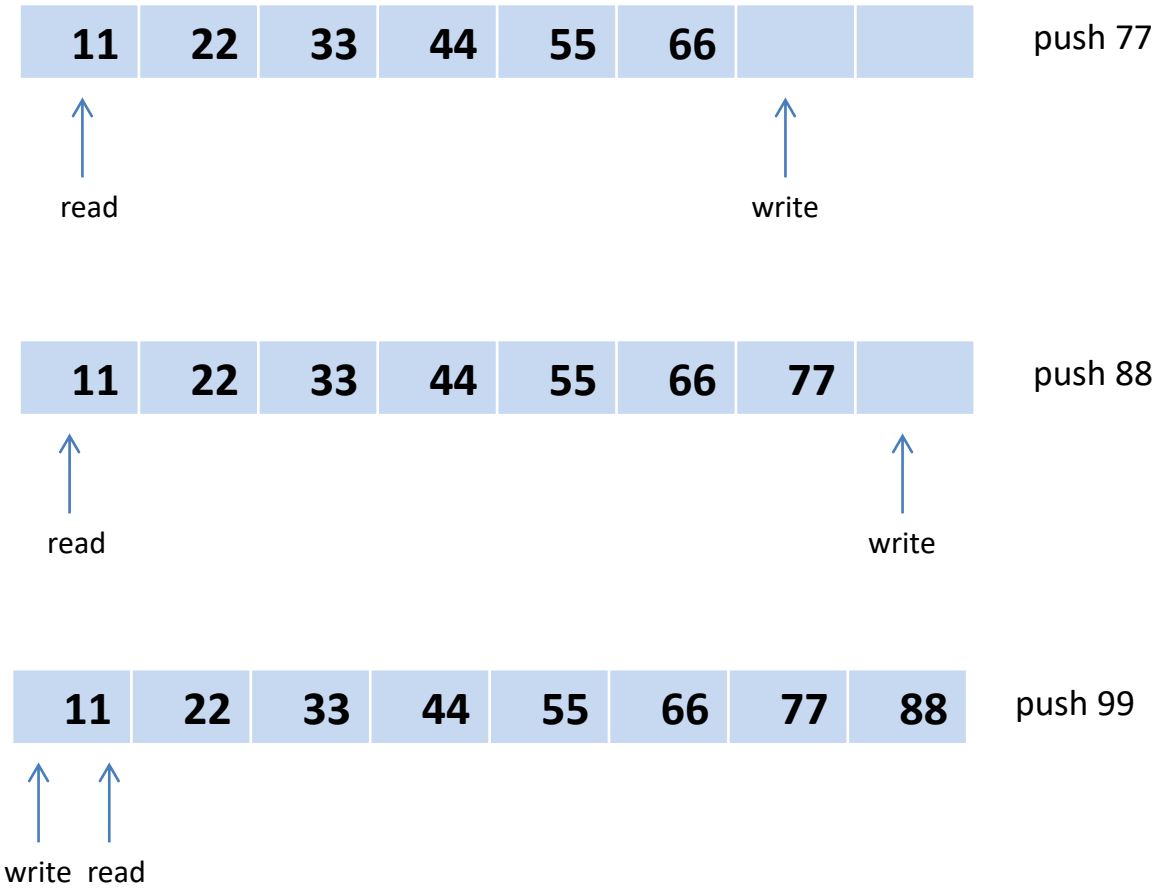
- Estrutura de Dados baseada em *FIFO*
  - Conectada nas extremidades
  - Produtor poderá sobrescrever elementos mais antigos



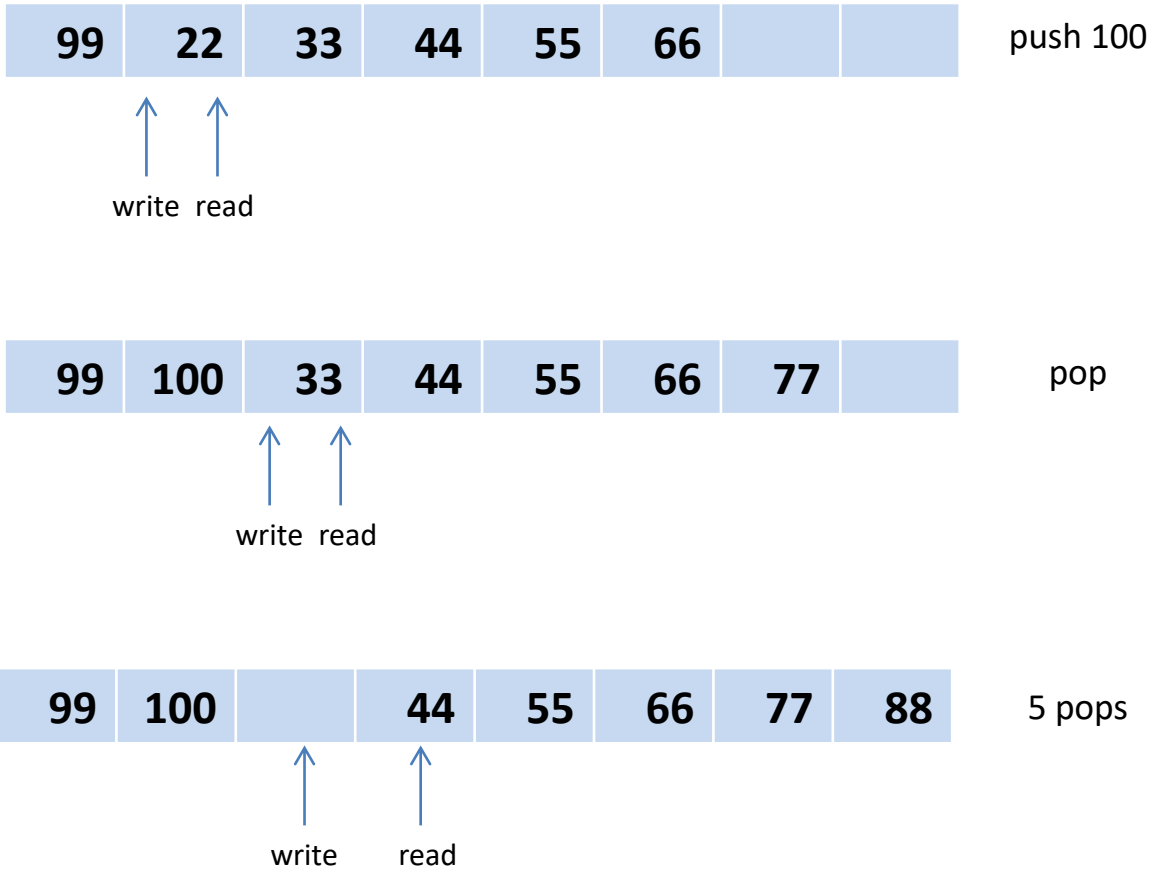

- *Ring buffer*
  - *Circular Buffer*
  - *Cyclic Buffer*
  - *Circular Queue*



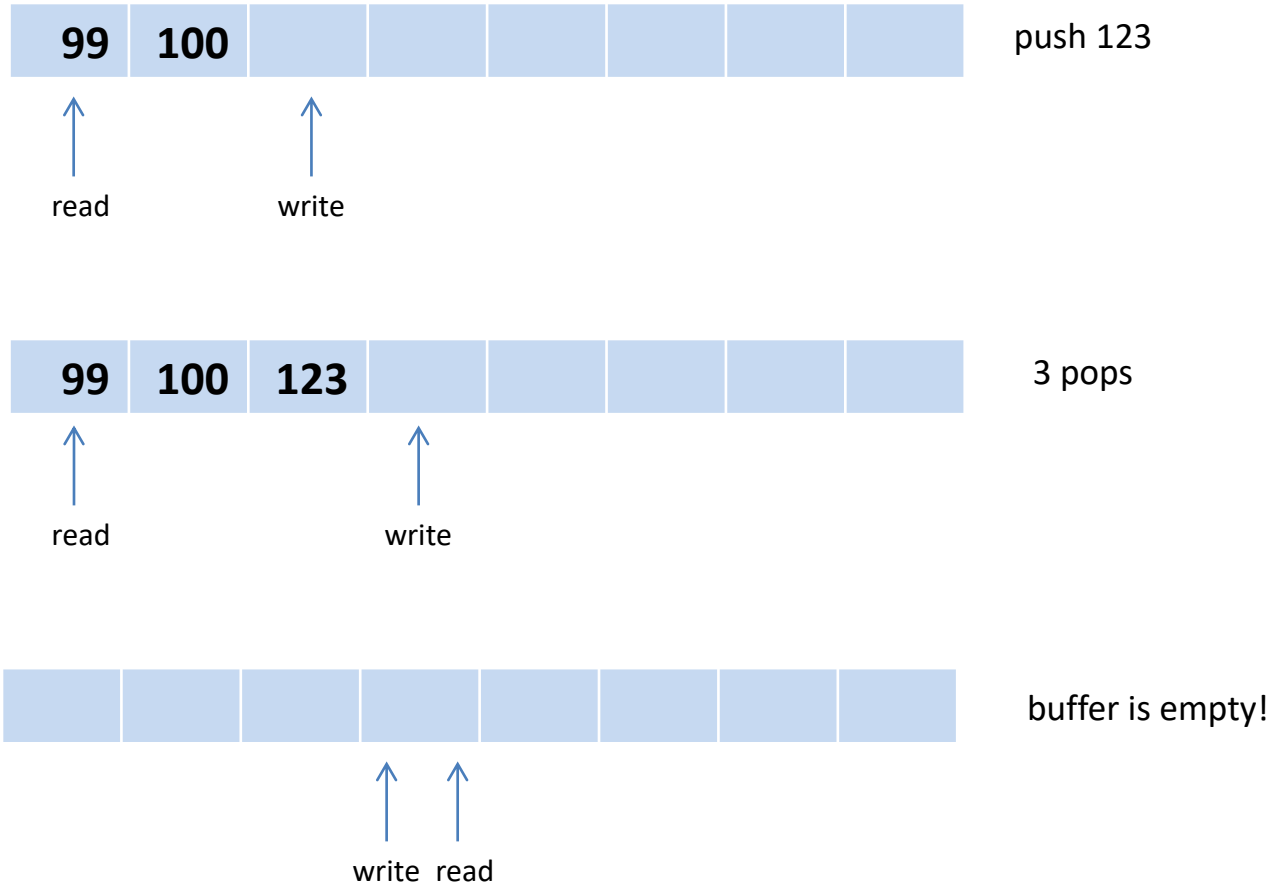
# *Ring Buffer*



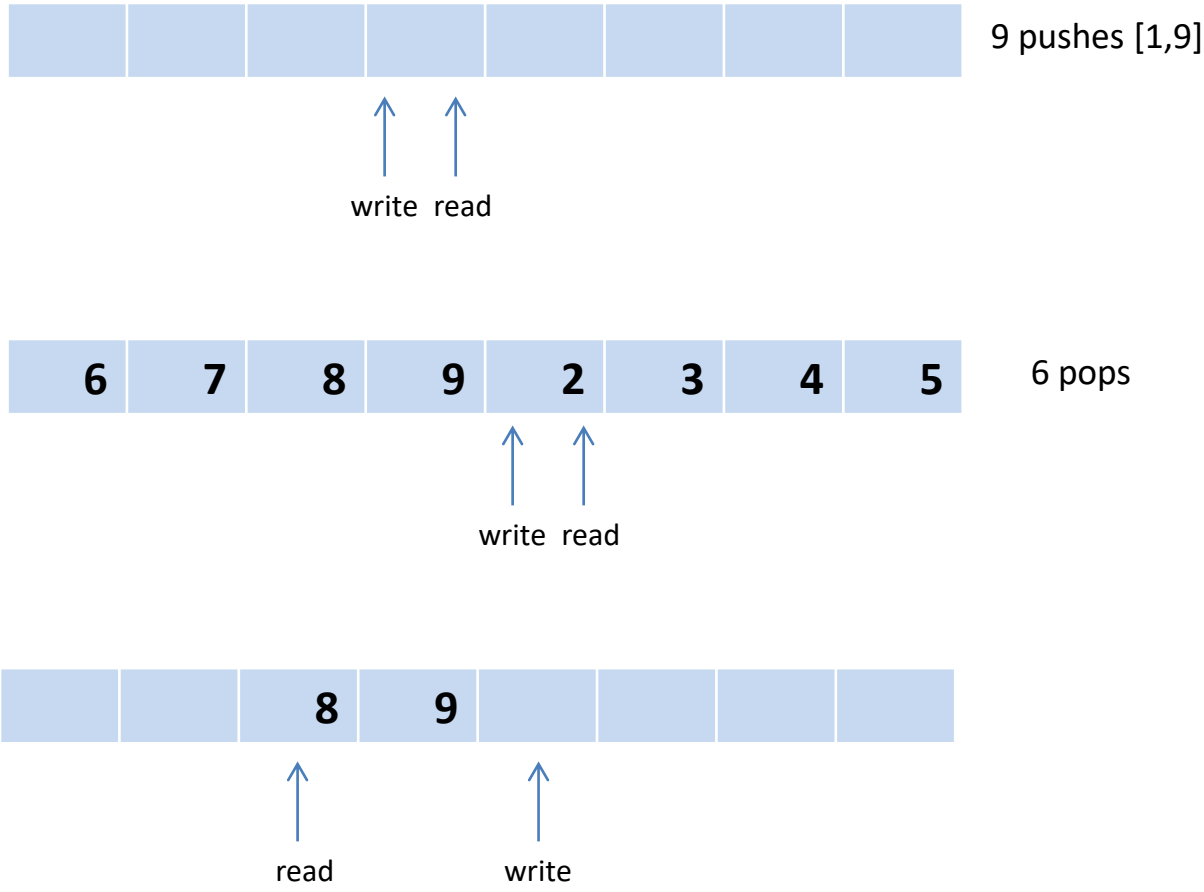
# *Ring Buffer*



# *Ring Buffer*



# *Ring Buffer*



# *Bounded Stack, Queue and Ring Buffer*

- No exemplo, a implementação compartilha da mesma interface pública
  - *Bounded* usada aqui para representar tamanho fixo
    - Se estiver cheio, ignorará os *pushes* subsequentes (retornará *false*)

```
//bounded_stack interface
/*
bool push(const T&);
bool pop(T&);
bool top(T&);
bool at(std::size_t, T&);
std::size_t size();
std::size_t capacity();
bool empty();
*/
```

```
//bounded_queue interface
/*
bool push(const T&);
bool pop(T&);
bool top(T&);
bool at(std::size_t, T&);
std::size_t size();
std::size_t capacity();
bool empty();
*/
```

```
//bounded_ring_buffer interface
/*
bool push(const T&);
bool pop(T&);
bool top(T&);
bool at(std::size_t, T&);
std::size_t size();
std::size_t capacity();
bool empty();
*/
```

# STL *Container Adaptors*

- *LIFO* (*Last-In First-Out*)

  `std::stack` `template <class T, class Container = deque<T>> class stack;`

- *FIFO* (*First-In First-Out*)

  `std::queue` `template <class T, class Container = deque<T>> class queue;`

```
//stack's interface
/*
bool empty() const;
size_t size() const;
T& top();
const T& top() const;
void push (const T&);
void push (T&&);
template <class... Args> void emplace (Args&&...);
void pop();
void swap (stack&) noexcept;
*/
```

```
//queue's interface
/*
bool empty() const;
size_t size() const;
T& front();
const T& front() const;
T& back();
const T& back() const;
void push (const T&);
void push (T&&);
template <class... Args> void emplace (Args&&...);
void pop();
void swap (queue&) noexcept;
*/
```

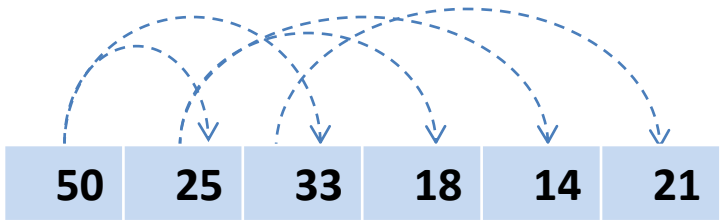
# stack_adapter

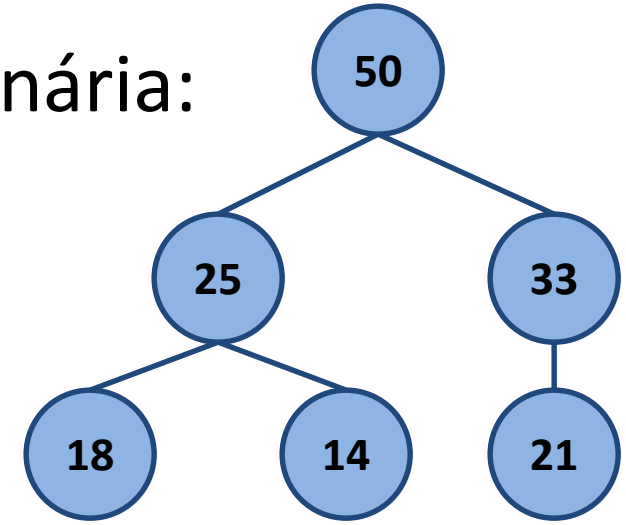
```
//stack's underlying container interface
/*
bool empty() const;
size_t size() const;
T& back();
void push_back(const T&);
void pop_back();
*/
```

```
std::stack<int> s;
std::stack<int, std::vector<int>> s;
std::stack<int, dynamic_array_adapter<int>> s;
stack_adapter<int> s;
stack_adapter<int, std::vector<int>> s;
stack_adapter<int, dynamic_array_adapter<int>> s;
```

# *Heap*

- Estrutura de Dados baseada em árvore organizada por máximos ou por mínimos em relação ao nó pai e seus nós filhos
  - No entanto, ela é implementada como um *array*
    - Não há nós e ponteiros de ligação
- Exemplo de uma *Max-heap* binária:



# *Unbounded Heap*

- No exemplo, a interface pública é similar a *Bounded Stack* (*Queue* ou *Ring Buffer*)
  - *Unbounded* usada aqui para representar tamanho não fixo ou "sem limites"
- Implementação de Max-heap e Min-heap
  - Em termos dos algoritmos da STL
    - `std::make_heap` $O(N)$
    - `std::push_heap` e `std::pop_heap` $O(lgN)$

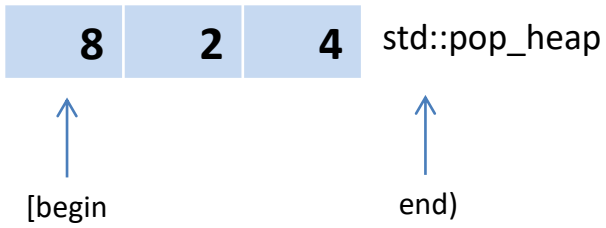
```
//unbounded_heap interface
/*
bool push(const T&);
bool pop(T&);
bool top(T&);
bool at(std::size_t, T&);
std::size_t size();
bool empty();
*/
```

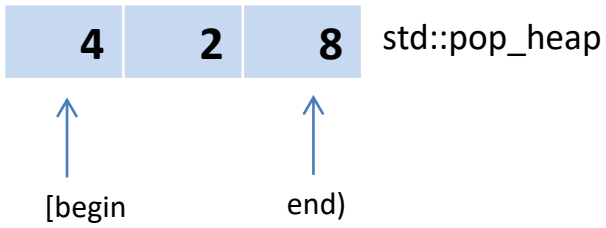
```
template <typename T>
using unbounded_max_heap = unbounded_heap<T, std::less<T>>;

template <typename T>
using unbounded_min_heap = unbounded_heap<T, std::greater<T>>;
```

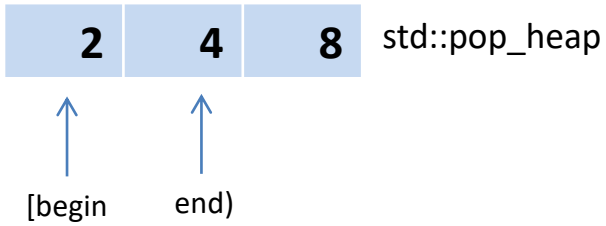
# heapsort

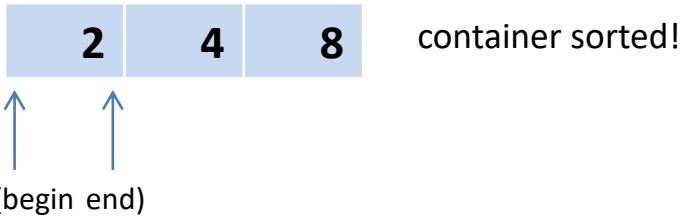
(ordenação)

- Ordem crescente
  - Montar uma *Heap-max*
    - *Heapify* o *container* (`std::make_heap`)
  - "Remover" todos os elementos do topo
    - Reposiciona para o final do *container* (`std::pop_heap`)



# heapsort

- Compare
  - `std::less<T>` => Ordem crescente
  - `Std::greater<T>` => Ordem decrescente

```
template <typename Container, typename Compare>
static inline void heapsort(Container& xs, Compare comp)
{
    if (xs.size() > 1)
    {
        using ptr_type = typename Container::pointer;
        ptr_type beg_ptr = &xs[0];
        ptr_type end_ptr = beg_ptr + xs.size();;

        std::make_heap(beg_ptr, end_ptr, comp);
        while (beg_ptr != end_ptr)
        {
            std::pop_heap(beg_ptr, end_ptr, comp);
            --end_ptr;
        }
    }
}
```

$$O(N \times lgN)$$

# Fila de Prioridade

- Estrutura de Dados baseada em *FIFO* com prioridade associada
  - Elemento com maior (ou menor) prioridade será o próximo a ser consumido
  - Usualmente, implementado em termos de uma *heap*
- Implementação de Max-PQ e Min-PQ
  - Em termos dos algoritmos da STL
    - `std::priority_queue`

```
template <typename T>
using MinPQBase = std::priority_queue<T, std::vector<T>, std::greater<T>>;

template <typename T>
using MaxPQBase = std::priority_queue<T, std::vector<T>, std::less<T>>;
```

```
//max_pq interface
/*
bool insert(const T&);
T& max();
bool max(T&);
bool remove_max();
bool remove_max(T&);
std::size_t size();
bool empty();
*/
```

```
//min_pq interface
/*
bool insert(const T&);
T& min();
bool min(T&);
bool remove_min();
bool remove_min(T&);
std::size_t size();
bool empty();
*/
```

# *Greedy*

- Estratégia ou heurística de resolução de problema de otimização onde se escolhe o ótimo local
  - Máximo local ou mínimo local em cada fase de execução, dependendo do problema
- Produz um resultado ótimo local, onde a solução é aproximada do ótima global, com a vantagem do tempo de processamento e consumo de recursos
  - Usualmente melhores do que uma estratégia que visa buscar o ótimo global
  - As vezes o ótimo local pode ser o ótimo global
    - Exemplo: Montar uma *Minimum Spanning Tree* (MST)

# *Bin Packing Problem*

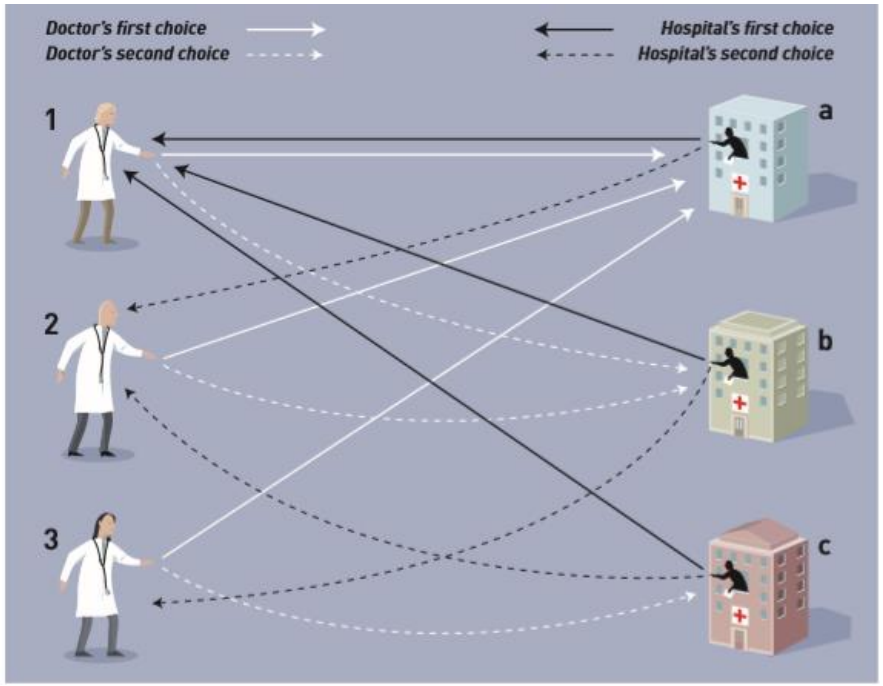
- Dada a capacidade uniforme das cestas e uma sequência de itens com seus respectivos pesos. Encontre o número mínimo de cestas necessárias para atribuir cada item da sequência.
  - *Greedy*
  - *Max Priority Queue*

```
unbounded_priority_queue::max_pq<Bin> pq;
pq.insert(Bin(bin_capacity));
for (auto w : weights)
{
    if (w <= bin_capacity)
    {
        if (pq.max().size() < w)
            pq.insert(Bin(bin_capacity));
        Bin b = pq.max();
        pq.remove_max();
        b.try_decrease(w);
        pq.insert(b);
    }
}
```

# *Stable Matching Problem*

- Encontrar uma correlação estável entre dois conjuntos de elementos com a mesma cardinalidade, onde cada elemento possui uma lista preferências
- Algoritmo de **Gale-Shapley**
  - *Greedy*
  - Ganhador do Prêmio Nobel

$$O(N^2)$$





**Matching doctors and hospitals.** When the doctors make offers, they all first choose hospital a, which accepts doctor 1 (the hospital's first choice). In a second stage, doctor 2 makes an offer to hospital b, and doctor 3 to hospital c, which gives a stable matching. When the hospitals have the right to make offers, the result is instead that doctor 2 is matched with hospital c and 3 with b.

https://www.nobelprize.org/uploads/2018/06/popular-economicsciences2012-1.pdf

# *Stable Matching Problem*

(exemplo – passo 1)

$$ms = \{AMAZON \quad MICROSOFT \quad NVIDIA\}$$

| | | |
|---|---|---|
| ↓ | ↓ | ↓ |
| $ADAM$ | $CARL$ | $ADAM$ |
| $ANNA$ | $ANNA$ | $CARL$ |
| $CARL$ | $ADAM$ | $ANNA$ |

$$ws = \{ADAM \quad ANNA \quad CARL\}$$

| | | |
|---|---|---|
| ↓ | ↓ | ↓ |
| $MICROSOFT$ | $NVIDIA$ | $NVIDIA$ |
| $AMAZON$ | $MICROSOFT$ | $AMAZON$ |
| $NVIDIA$ | $AMAZON$ | $MICROSOFT$ |

$$AMAZON \rightarrow ADAM$$

Preferência

| AMAZON | MICROSOFT | NVIDIA |
|---|---|---|
| 0 | 0 | 0 |

Fila de escolha

| |
|---|
| **AMAZON** |
| MICROSOFT |
| NVIDIA |

# *Stable Matching Problem*

(exemplo – passo 2)

$ms = \{AMAZON \quad MICROSOFT \quad NVIDIA\}$

| | | |
|---|---|---|
| $* ADAM$ | $CARL$ | $ADAM$ |
| $ANNA$ | $ANNA$ | $CARL$ |
| $CARL$ | $ADAM$ | $ANNA$ |

$ws = \{ADAM \quad ANNA \quad CARL\}$

| | | |
|---|---|---|
| $MICROSOFT$ | $NVIDIA$ | $NVIDIA$ |
| $* AMAZON$ | $MICROSOFT$ | $AMAZON$ |
| $NVIDIA$ | $AMAZON$ | $MICROSOFT$ |

Preferência

| AMAZON | MICROSOFT | NVIDIA |
|---|---|---|
| 1 | 0 | 0 |

Fila de escolha

| |
|---|
| AMAZON |
| **MICROSOFT** |
| NVIDIA |

$AMAZON \rightarrow ADAM$

$MICROSOFT \rightarrow CARL$

# *Stable Matching Problem*

(exemplo – passo 3.1)

$ms = \{AMAZON \quad MICROSOFT \quad NVIDIA\}$

| | | |
|---|---|---|
| $ADAM$ | $CARL$ | $ADAM$ |
| $ANNA$ | $ANNA$ | $CARL$ |
| $CARL$ | $ADAM$ | $ANNA$ |

$ws = \{ADAM \quad ANNA \quad CARL\}$

| | | |
|---|---|---|
| $MICROSOFT$ | $NVIDIA$ | $NVIDIA$ |
| $* AMAZON$ | $MICROSOFT$ | $AMAZON$ |
| $NVIDIA$ | $AMAZON$ | $* MICROSOFT$ |

Preferência

| AMAZON | MICROSOFT | NVIDIA |
|---|---|---|
| 1 | 1 | 0 |

Fila de escolha

| |
|---|
| AMAZON |
| MICROSOFT |
| **NVIDIA** |

$AMAZON \rightarrow ADAM$

$MICROSOFT \rightarrow CARL$

$NVIDIA \rightarrow ADAM?$

# *Stable Matching Problem*

(exemplo – passo 3.2)

$ms = \{AMAZON \quad MICROSOFT \quad NVIDIA\}$

| | | |
|---|---|---|
| $ADAM$ | $CARL$ | $ADAM$ |
| $ANNA$ | $ANNA$ | $CARL$ |
| $CARL$ | $ADAM$ | $ANNA$ |

$ws = \{ADAM \quad ANNA \quad CARL\}$

| | | |
|---|---|---|
| $MICROSOFT$ | $NVIDIA$ | $NVIDIA$ |
| $* AMAZON$ | $MICROSOFT$ | $AMAZON$ |
| $NVIDIA$ | $AMAZON$ | $* MICROSOFT$ |

Preferência

| AMAZON | MICROSOFT | NVIDIA |
|---|---|---|
| 1 | 1 | 1 |

Fila de escolha

| |
|---|
| AMAZON |
| MICROSOFT |
| **NVIDIA** |

$AMAZON \rightarrow ADAM$

$MICROSOFT \rightarrow CARL$

$NVIDIA \rightarrow$ ~~$ADAM?$~~ $CARL?$

# *Stable Matching Problem*

(exemplo – passo 3.3)

$ms = \{AMAZON \quad MICROSOFT \quad NVIDIA\}$

| | | |
|---|---|---|
| $ADAM$ | $CARL$ | $ADAM$ |
| $ANNA$ | $ANNA$ | $CARL$ |
| $CARL$ | $ADAM$ | $ANNA$ |

$ws = \{ADAM \quad ANNA \quad CARL\}$

| | | |
|---|---|---|
| $MICROSOFT$ | $NVIDIA$ | $*\ NVIDIA$ |
| $*\ AMAZON$ | $MICROSOFT$ | $AMAZON$ |
| $NVIDIA$ | $AMAZON$ | $MICROSOFT$ |

Preferência

| AMAZON | MICROSOFT | NVIDIA |
|---|---|---|
| 1 | 1 | 2 |

Fila de escolha

| |
|---|
| AMAZON |
| MICROSOFT |
| NVIDIA |
| **MICROSOFT** |

$AMAZON \rightarrow ADAM$

$MICROSOFT \rightarrow \sout{CARL}$

$NVIDIA \rightarrow CARL$

# *Stable Matching Problem*

(exemplo – passo 4)

$ms = \{AMAZON \quad MICROSOFT \quad NVIDIA\}$

| | | |
|---|---|---|
| $ADAM$ | $CARL$ | $ADAM$ |
| $ANNA$ | $ANNA$ | $CARL$ |
| $CARL$ | $ADAM$ | $ANNA$ |

$ws = \{ADAM \quad ANNA \quad CARL\}$

| | | |
|---|---|---|
| $MICROSOFT$ | $NVIDIA$ | $* NVIDIA$ |
| $* AMAZON$ | $MICROSOFT$ | $AMAZON$ |
| $NVIDIA$ | $AMAZON$ | $MICROSOFT$ |

Preferência

| AMAZON | MICROSOFT | NVIDIA |
|---|---|---|
| 1 | 1 | 2 |

Fila de escolha

| |
|---|
| AMAZON |
| MICROSOFT |
| NVIDIA |
| **MICROSOFT** |

$AMAZON \rightarrow ADAM$

$MICROSOFT \rightarrow \cancel{CARL}\ ANNA$

$NVIDIA \rightarrow CARL$

# *Stable Matching Problem*

(exemplo – resultado)

$$ms = \{AMAZON \quad MICROSOFT \quad NVIDIA\}$$

| | | |
|---|---|---|
| $\rightarrow ADAM$ | $CARL$ | $ADAM$ |
| $ANNA$ | $\rightarrow ANNA$ | $\rightarrow CARL$ |
| $CARL$ | $ADAM$ | $ANNA$ |

$$ws = \{ADAM \quad ANNA \quad CARL\}$$

| | | |
|---|---|---|
| $MICROSOFT$ | $NVIDIA$ | $* NVIDIA$ |
| $* AMAZON$ | $* MICROSOFT$ | $AMAZON$ |
| $NVIDIA$ | $AMAZON$ | $MICROSOFT$ |

Preferência

| AMAZON | MICROSOFT | NVIDIA |
|---|---|---|
| 1 | 2 | 2 |

Fila de escolha

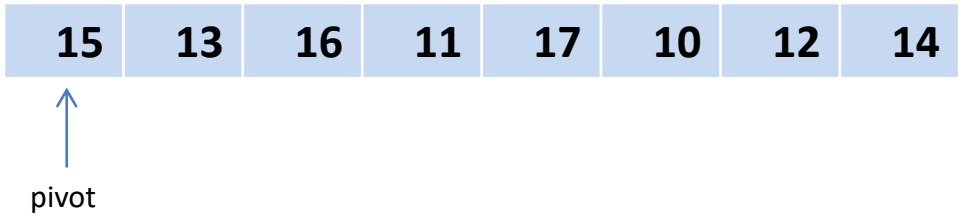
| |
|---|
| AMAZON |
| MICROSOFT |
| NVIDIA |
| MICROSOFT |

$AMAZON \rightarrow ADAM$

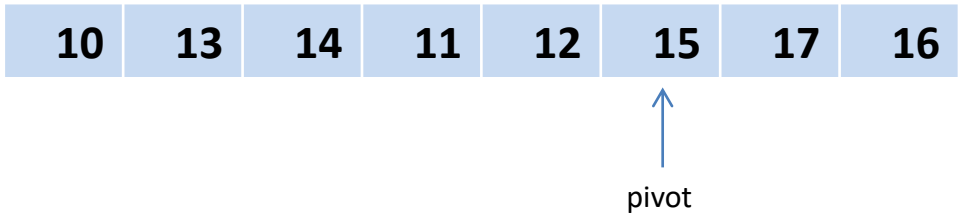
$MICROSOFT \rightarrow ANNA$

$NVIDIA \rightarrow CARL$

# Particionamento

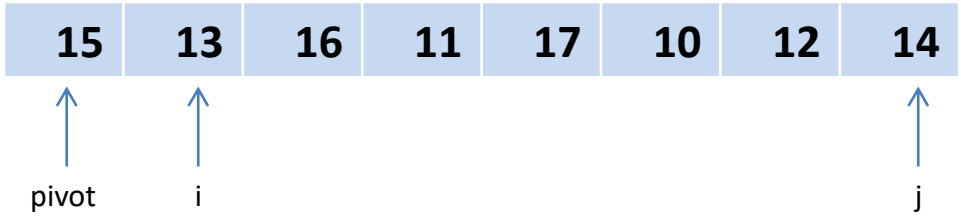



- Escolher um elemento como *pivot*
  - *Pivot* será posicionado como primeiro elemento
    - Mediana de 3 poderá ser usada como critério de seleção do *pivot*
- Elementos menores são deslocados para esquerda do *pivot*
- Elementos maiores são deslocados para direita do *pivot*
- Elementos iguais são deslocados arbitrariamente
  - Depende da implementação
  - Usualmente deslocados a direita

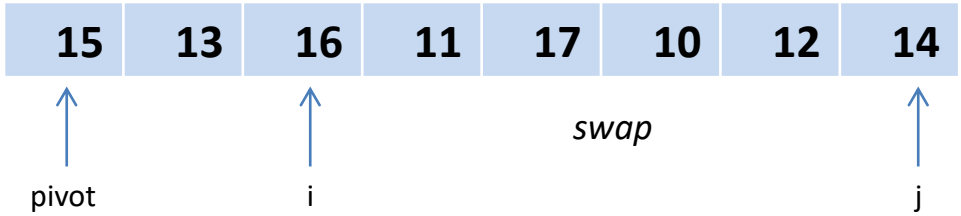


# inplace_partition

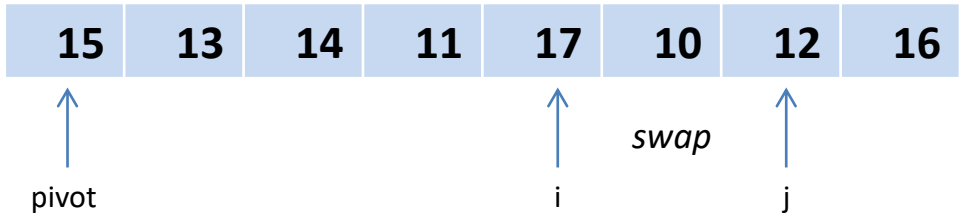
(passo a passo)




---

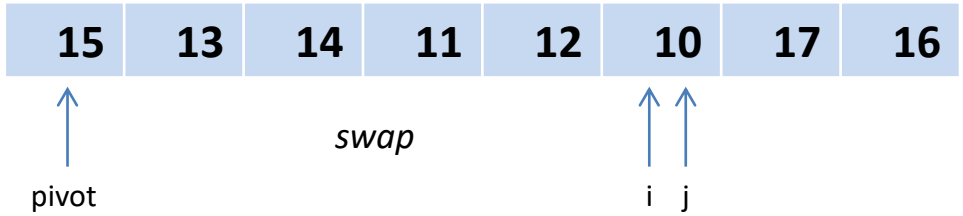


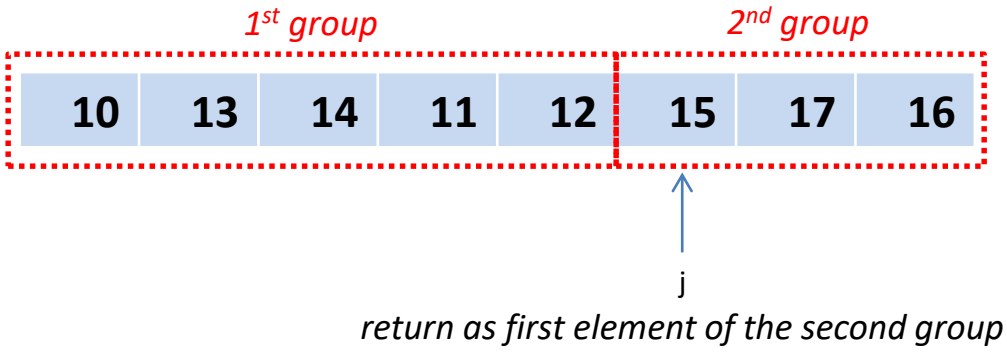

---



# inplace_partition

(passo a passo)



# inplace_partition

```cpp
template <typename T>
std::size_t inplace_partition(std::vector<T>& xs, std::size_t first, std::size_t last, std::size_t pivot)
{
    assert(first < last && first <= pivot && pivot < last);

    std::swap(xs[first], xs[pivot]);
    T& pivot_value = xs[first];
    std::size_t i = first; //left to right
    std::size_t j = last;  //right to left
    while (true)
    {
        while (xs[++i] < pivot_value && i != last - 1); //to the left from pivot
        while (pivot_value <= xs[--j] && j != first); //to the right from pivot

        if (i >= j) break; //can't swap anymore

        std::swap(xs[i], xs[j]);
    }
    std::swap(pivot_value, xs[j]); //change 'the pivot' element to 'the right to left bounded' element
    return j; //left from j is less than xs[j], otherwise is greater or equal to xs[j]
}
```

$O(N)$

# std::partition

- Usa um *predicate* para dividir os grupos
- Comportamento diferente do `inplace_partition`

```
std::vector<int> xs { 15, 13, 16, 11, 17, 10, 12, 14 };
std::partition(xs.begin(), xs.end(), [&](int x) { return x < xs[0]; });
//14 13 12 11 10 17 16 15
```

  - Adaptação para comportamento do `inplace_partition`

```
std::vector<int> xs { 15, 13, 16, 11, 17, 10, 12, 14 };
partitioning::partition(xs.begin(), xs.end());
//10 13 14 11 12 15 17 16
```

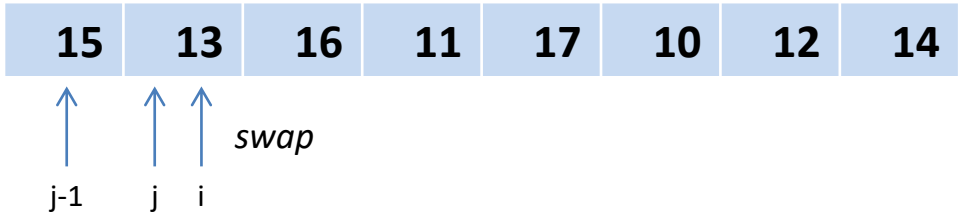
```
template <typename BidirectionalIterator>
BidirectionalIterator partition(BidirectionalIterator first, BidirectionalIterator last, BidirectionalIterator pivot)
{
    using T = typename std::iterator_traits<BidirectionalIterator>::value_type;

    std::swap(*first, *pivot);
    auto lt = [first](const T& x) { return x < *first; };
    auto bound = std::partition(std::next(first), last, lt);
    pivot = std::prev(bound);
    std::swap(*first, *pivot);
    return pivot;
}
```

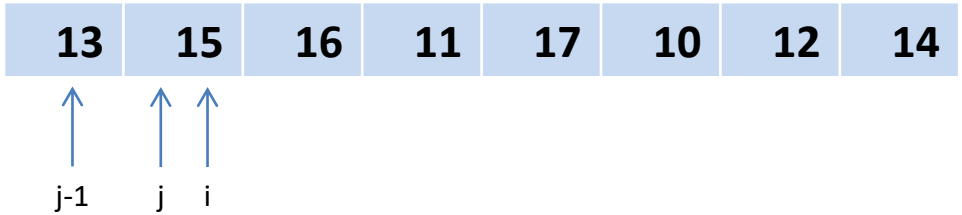


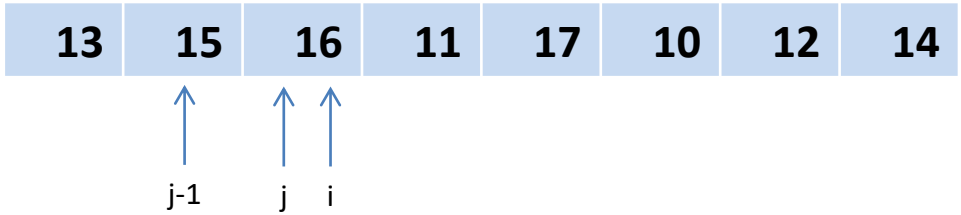
pivot equals first (default)

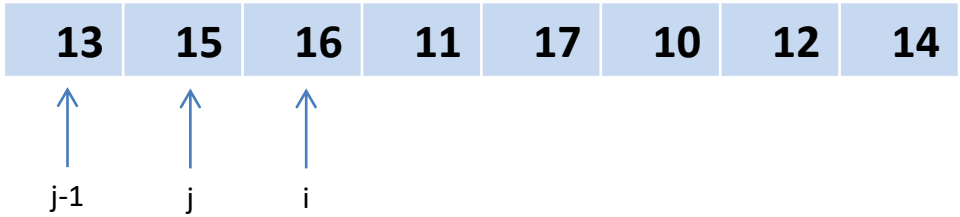
# insertsort

(ordenação)



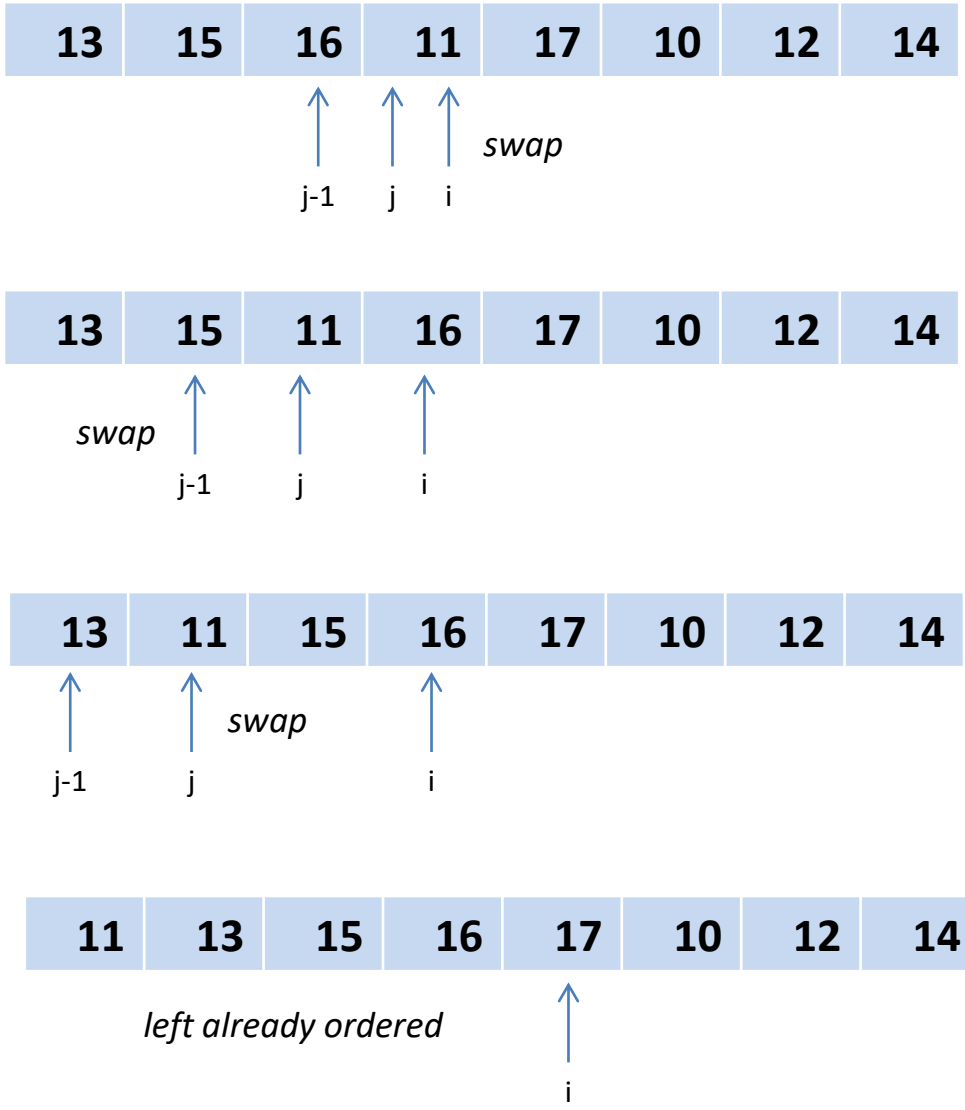



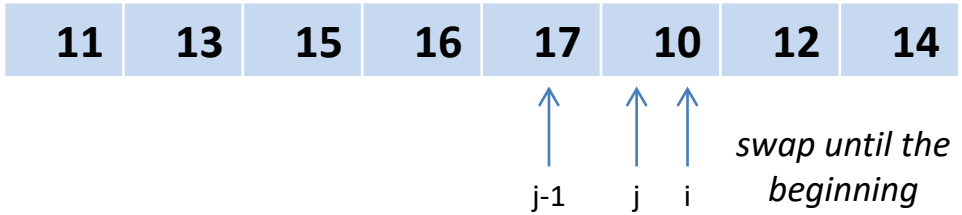
# insertsort

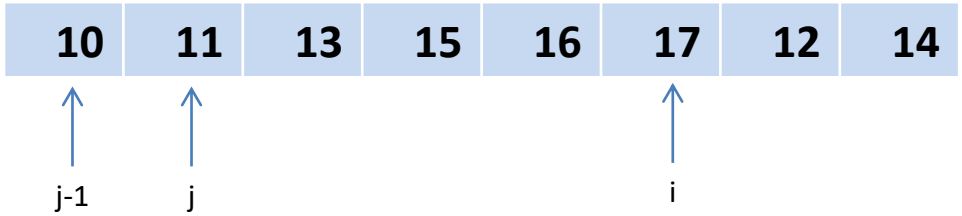
(ordenação)



# insertsort

(ordenação)




# insertsort

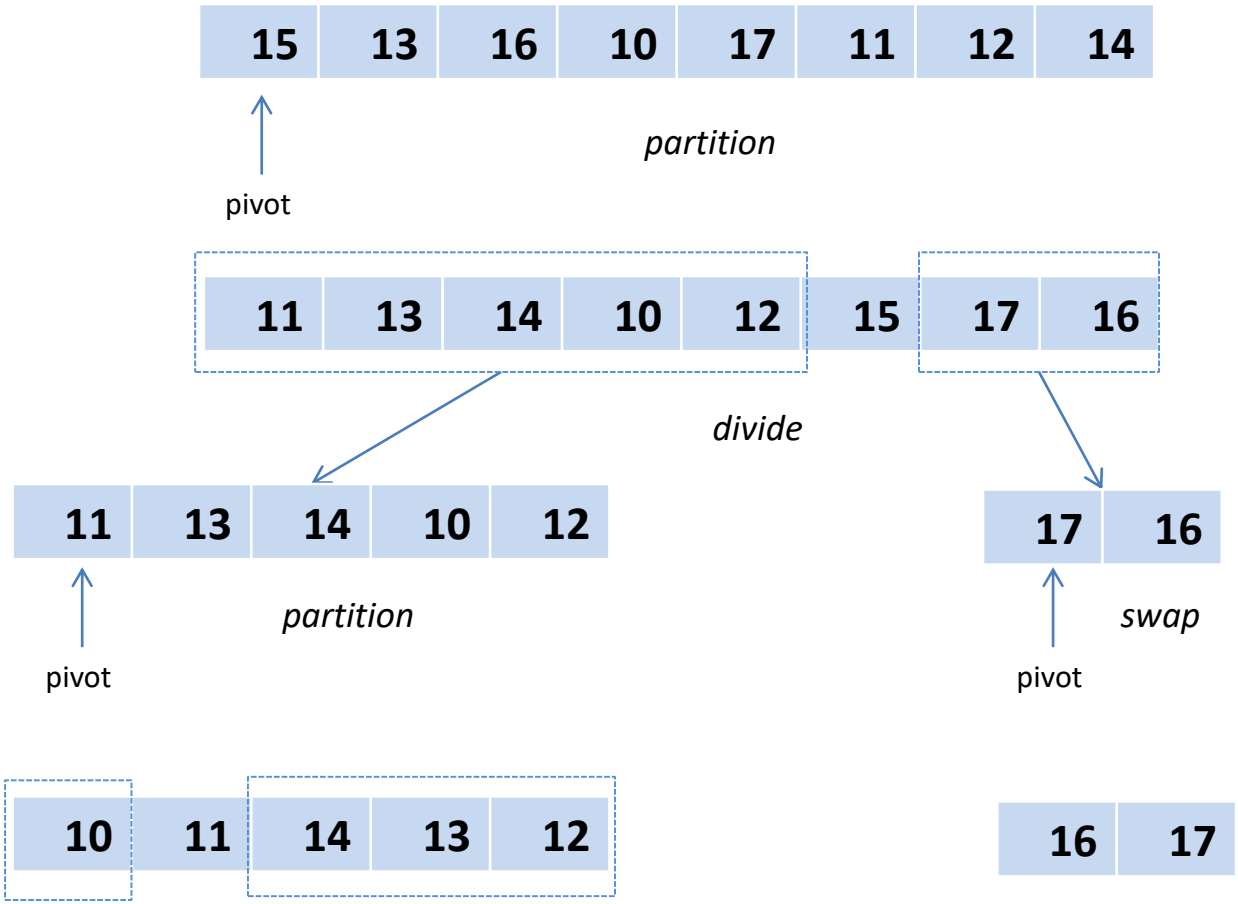
```
template <typename Container>
static inline void insertsort(Container& xs)
{
    const std::size_t N = xs.size();
    for (std::size_t i = 1; i < N; ++i)
    {
        for (std::size_t j = i; j > 0; --j)
        {
            if (xs[j] < xs[j - 1])
                std::swap(xs[j], xs[j - 1]);
            else
                break;
        }
    }
}
```

$O(N^2)$

# quicksort

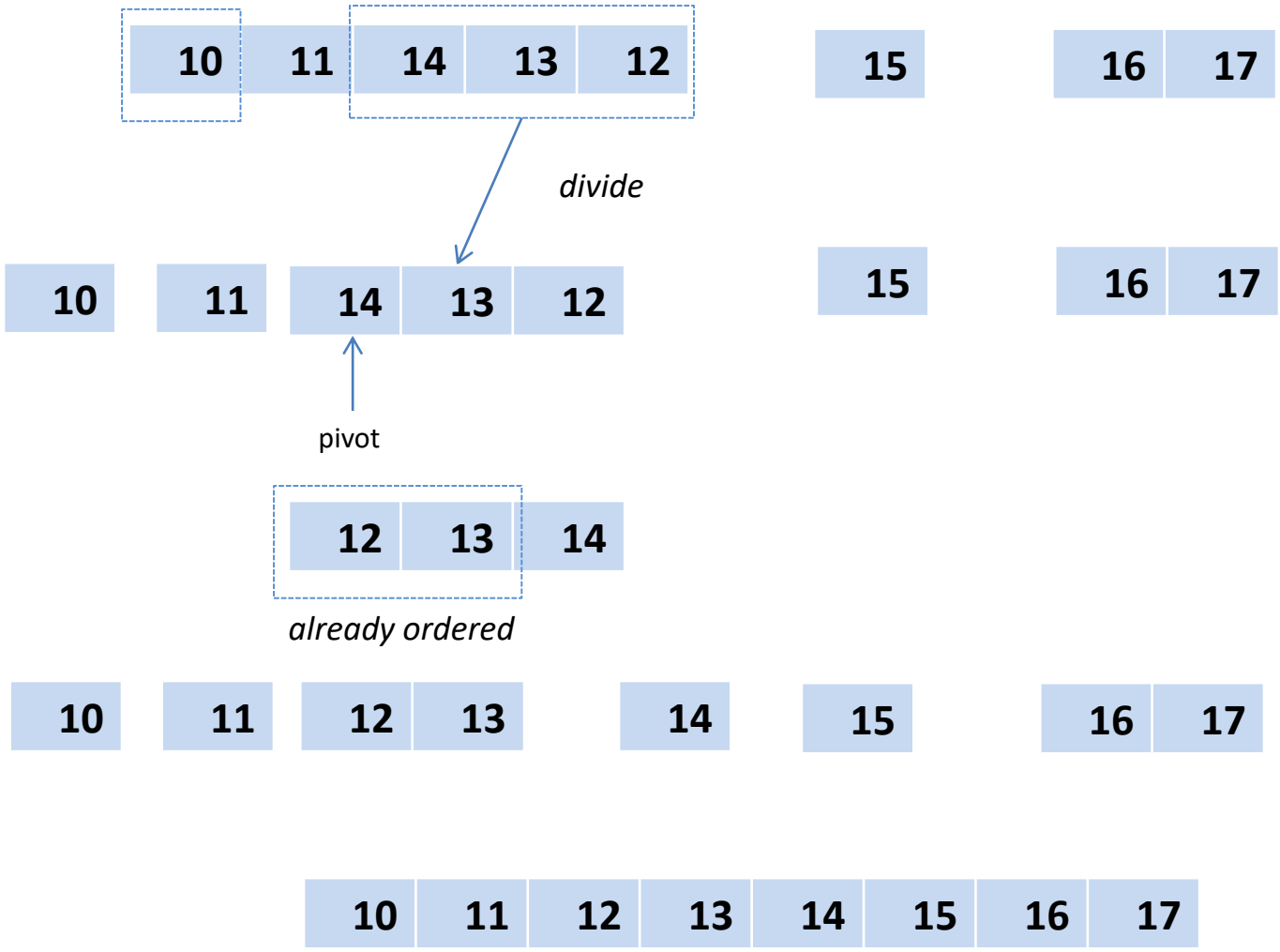
(ordenação)

- Dividir e conquistar



# quicksort

(ordenação)



# quicksort

```
template <typename Container>
static inline void quicksort(Container& xs, std::size_t first, std::size_t last)
{
    assert(first <= last);

    const std::size_t N = last - first;
    if (N < 2)
    {
        return;
    }
    if (N == 2)
    {
        if (xs[first + 1] < xs[first])
            std::swap(xs[first + 1], xs[first]);
        return;
    }

    //swap first with median of 3
    std::size_t pos0 = first, pos1 = first + N / 2, pos2 = first + N - 1;
    std::swap(xs[median_of_3(xs, pos0, pos1, pos2)], xs[first]);

    auto pivot = xs.begin() + first;
    auto end = xs.begin() + last;
    auto bound = std::partition(pivot + 1, end, [&](typename Container::const_reference x) {
        return x < *pivot;
    });
    if (pivot != bound)
    {
        std::swap(*pivot, *(bound - 1));
        std::size_t p = first + std::distance(pivot, bound);
        quicksort(xs, first, p - 1);
        quicksort(xs, p, last);
    }
}
```

$$\theta(N \times lgN)$$

# *Dynamic Programming (DP)*

- Programação Dinâmica
  - Subestrutura ótima *(Optimal Substructure)*

$$Fibonacci(5) = Fibonacci(4) + Fibonacci(3)$$

  - Sobreposição de subproblemas *(Overlapping Subproblems)*

$$Fibonacci(5) = Fibonacci(4) + Fibonacci(3)$$
$$Fibonacci(4) = Fibonacci(3) + Fibonacci(2)$$

- *Top-down*
  - Recursivo
    - Quebra em subproblemas usando recursão
      - A partir do resultado final chegando no valor inicial
- *Bottom-up*
  - Iterativo
    - Quebra em subproblemas usando iteração
      - A partir do valor inicial chegando no resultado final

# Coeficiente Binomial

(programação dinâmica)

Relação de Recorrência

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k}, & \text{if } 0 < k < n. \\ 1, & \text{otherwise } (\mathrm{k} = 0 \text{ or } \mathrm{k} = \mathrm{n}). \end{cases}$$

Relação de Stifel

| | 0 | 1 | 2 | 3 | 4 | 5 | ... | k |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | | | | | | | |
| 1 | 1 | 1 | | | | | | |
| 2 | 1 | 2 | 1 | | | | | |
| 3 | 1 | 3 | 3 | 1 | | | | |
| 4 | 1 | 4 | 6 | 4 | 1 | | | |
| 5 | 1 | 5 | 10 | 10 | 5 | 1 | | |
| ... | | | | | | | | |
| n | | | | | | | | |

Bottom-up

$$\binom{n}{k} = \binom{n}{n-k}$$

Simetria

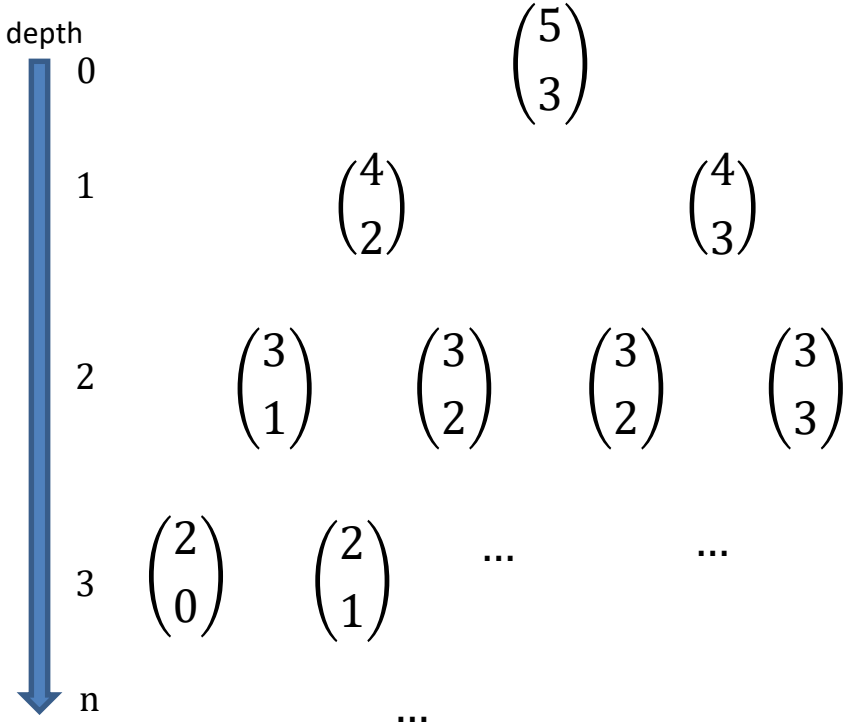
# Coeficiente Binomial

(análise – recursividade)

- $\binom{5}{3} = \frac{5!}{3! + 2!} = \begin{cases} 1, & 0 < k < n. \\ \binom{n-1}{k-1} + \binom{n-1}{k}, & otherwise. \end{cases}$

$n = 5$

$k = 3$




$$
\begin{aligned}
T(n) &\geq 2^1 T(n-1) \\
&\geq 2^2 T(n-2) \\
&\geq 2^3 T(n-3) \\
&\geq 2^4 T(n-4) \\
&\quad \dots \\
&\geq 2^{n-1} T(1) \\
&\geq 2^n T(0) = \Omega(2^n)
\end{aligned}
$$

$$\Omega(2^n)$$

# Coeficiente Binomial

(análise – programação dinâmica)

- $\binom{5}{3} =$

$n = 5$

$k = 3$

$k = 2 \rightarrow simetria$

| | | |
|---|---|---|
| 1 | | |
| 1 | 1 | |
| 1 | 2 | 1 |
| 1 | 3 | 3 |
| 1 | 4 | 6 |
| 1 | 5 | 10 |

1

n

n + 1

k

| i | 0 | 1 | 2 | … | k | k + 1 | … | n |
|---|---|---|---|---|---|---|---|---|
| *steps* | 1 | 2 | 3 | … | k | k | … | k |

$1 + 2 + 3 + \ \ldots + (k+1) = \frac{k(k+1)}{2}$

$k(n-k+1)$

$\frac{k(k+1)}{2} + k(n-k+1)$

$kn - \frac{k^2}{2} + \frac{3k}{2} \ \in \ \Theta(kn)$

$\Theta(kn)$

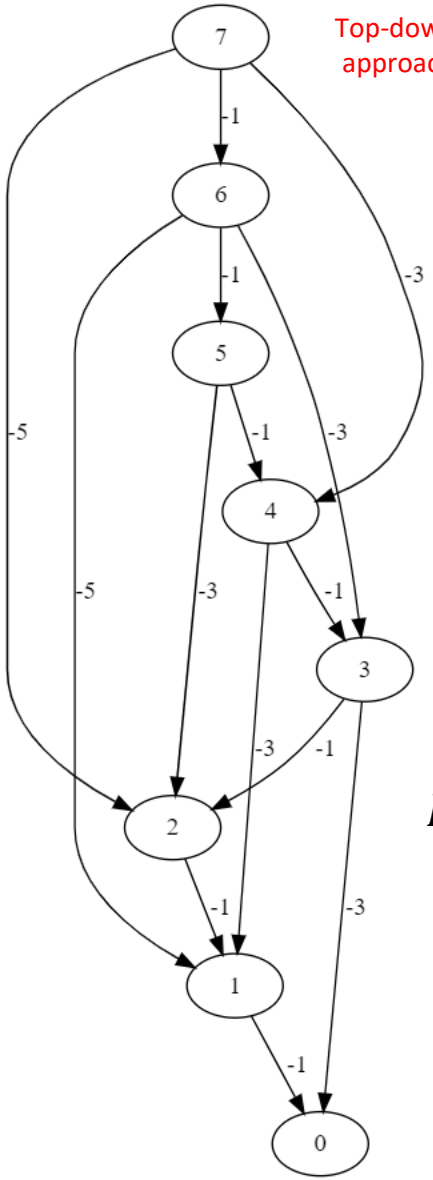
# *Water Recipient Filling Problem*

- Dada a capacidade em litros de um recipiente de água e um certo número de garrafas com capacidade distinta. Encontre o número mínimo de garrafas para preencher o recipiente. Assuma que não é possível mover esse recipiente de água até a fonte.
  - *Dynamic Programming*
  - *Array*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 |
| 5 | 0 | 1 | 1 | 2 | 2 | 1 | 2 | 2 |

# Water Recipient Filling Problem

Top-down approach

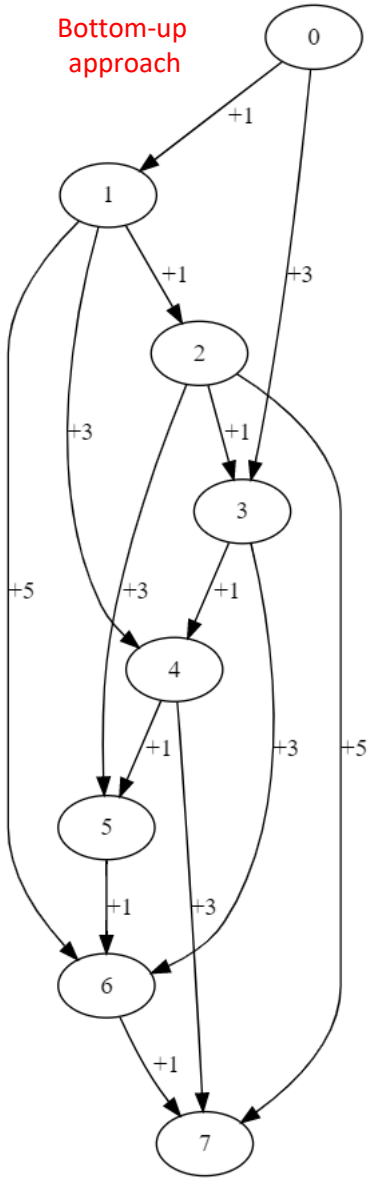



```
7 3 5 3 1

          0    1    2    3    4    5    6    7
==========================================================
  1 |     0    1    2    3    4    5    6    7
  3 |     0    1    2    1    2    3    2    3
  5 |     0    1    2    1    2    1    2    3

3 Bottle(s) needed
```

$$B(row, col, k) = min \begin{cases} 1 + B(row, col - k, k) \\ B(row - 1, col, k) \end{cases}$$

$$row = \{0 \dots 2\}$$
$$col = \{0 \dots 7\}$$
$$k = \{1, 3, 5\}$$

Bottom-up approach



# Análise de Algoritmos

- A análise de algoritmos tem como objetivo determinar os recursos (tempo ou espaço) necessários para executar um determinado algoritmo e servir como base de comparação na escolha de um algoritmo eficiente
  - Usualmente o foco da análise está no tempo de execução
    - Em quanto tempo um algoritmo com uma entrada de tamanho arbitrário (N) será processado?

- Análise assintótica é usada para descrever os limites do algoritmo tendendo ao infinito (ou um volume muito grande de dados)
  - Aproximação para pior caso com notação “Big-O”
    - N^2 + N + 32 => O(N^2)
    - 10N + N lg N => O(N lg N)
    - N/2 + 3 lg N => O(N)

# Big-Oh (O) Big-Omega (Ω) Big-Theta (Θ)

(conceito)

- *Upper bound* – 'Big-Oh' (O) – análogo a ≤

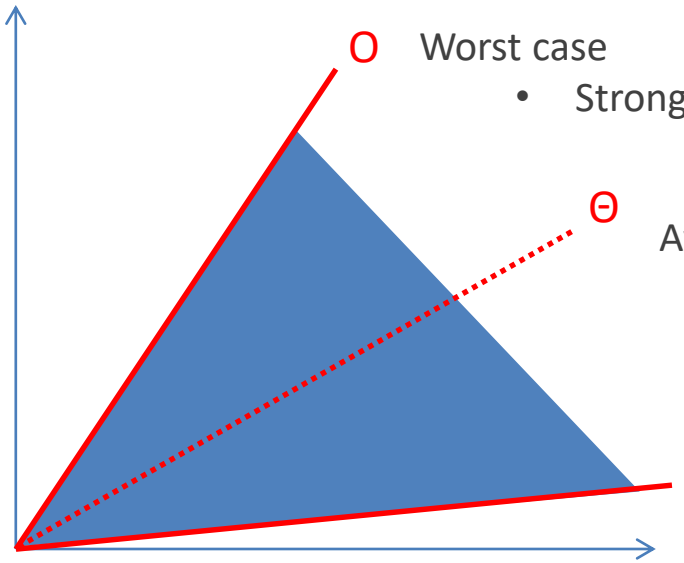
$$g(N) = O\big(f(n)\big)\ quando\ a\ proporção\ \left|\frac{g(n)}{f(n)}\right|\ é\ delimitada\ por\ cima\ quando\ N\ tende\ ao\ infinito$$

- *Lower bound* – Omega (Ω) – análogo a ≥

$$g(N) = \Omega\big(f(n)\big)\ quando\ a\ proporção\ \left|\frac{g(n)}{f(n)}\right|\ é\ delimitada\ por\ baixo\ quando\ N\ tende\ ao\ infinito$$

- Theta (Θ) – análogo a =

$$g(N) = \Theta\big(f(n)\big)\ quando\ for\ g(N) = O\big(f(n)\big)\ e\ g(N) = \Omega\big(f(n)\big)$$




O Worst case
- Strong guarantees, maybe unrealistic data

Θ Average case
- "Typical" input data

Ω Best case

# Big-Oh (O) Big-Omega (Ω) Big-Theta (Θ)

(exemplo)

$$f_1(N) = N^2 \qquad f_2(N) = 3N + 10 \qquad f_3(N) = N + 2$$

| N | f1 | f2 | f3 | f2/f1 | f1/f2 | f2/f3 | f3/f2 | f1/f3 | f3/f1 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 13 | 3 | 13.000000 | 0.076923 | 4.333333 | 0.230769 | 0.333333 | 3.000000 |
| 2 | 4 | 16 | 4 | 4.000000 | 0.250000 | 4.000000 | 0.250000 | 1.000000 | 1.000000 |
| 5 | 25 | 25 | 7 | 1.000000 | 1.000000 | 3.571429 | 0.280000 | 3.571429 | 0.280000 |
| 10 | 100 | 40 | 12 | 0.400000 | 2.500000 | 3.333333 | 0.300000 | 8.333333 | 0.120000 |
| 100 | 10000 | 310 | 102 | 0.031000 | 32.258065 | 3.039216 | 0.329032 | 98.039216 | 0.010200 |
| 250 | 62500 | 760 | 252 | 0.012160 | 82.236842 | 3.015873 | 0.331579 | 248.015873 | 0.004032 |
| 500 | 250000 | 1510 | 502 | 0.006040 | 165.562914 | 3.007968 | 0.332450 | 498.007968 | 0.002008 |
| 1000 | 1000000 | 3010 | 1002 | 0.003010 | 332.225914 | 3.003992 | 0.332890 | 998.003992 | 0.001002 |
| 10000 | 100000000 | 30010 | 10002 | 0.000300 | 3332.222592 | 3.000400 | 0.333289 | 9998.000400 | 0.000100 |
| 100000 | 10000000000 | 300010 | 100002 | 0.000030 | 33332.222259 | 3.000040 | 0.333329 | 99998.000040 | 0.000010 |
| 1000000 | 1000000000000 | 3000010 | 1000002 | 0.000003 | 333332.222226 | 3.000004 | 0.333333 | 999998.000004 | 0.000001 |
| 10000000 | 100000000000000 | 30000010 | 10000002 | 0.000000 | 3333332.222223 | 3.000000 | 0.333333 | 9999998.000000 | 0.000000 |
| 100000000 | 10000000000000000 | 300000010 | 100000002 | 0.000000 | 33333332.222222 | 3.000000 | 0.333333 | 99999998.000000 | 0.000000 |
| 1000000000 | 100000000000000000 | 3000000010 | 1000000002 | 0.000000 | 333333332.222222 | 3.000000 | 0.333333 | 999999998.000000 | 0.000000 |

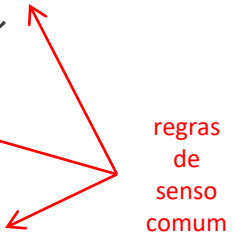
$$f_2(N) = O\big(f_1(N)\big)$$

$$f_2(N) = O\big(f_3(N)\big) \wedge f_3(N) = O\big(f_2(N)\big) \Rightarrow f_2(N) = \Theta\big(f_3(N)\big)$$ relação antisimétrica

$$f_1(N) = \Omega\big(f_3(N)\big)$$

# Dominância

- *Exponecial domina Polinomial*
- $N^a$ *domina* $N^b$ *quando* $a > b$
- *Polinomial domina Logaritmo*




$$lg\ N < \sqrt{N} < lg^2 N < N < N\ lg\ N < N^2 < 2^N < N! < 2^{N^2} < 2^{2^N} < N^{2^N}$$

| lg N | N^1/2 | N | N lg N | N^2 | N^3 |
|---|---|---|---|---|---|
| 4 | 4 | 16 | 64 | 256 | 4096 |
| 5 | 6 | 32 | 160 | 1024 | 32768 |
| 6 | 8 | 64 | 384 | 4096 | 262144 |
| 7 | 11 | 128 | 896 | 16384 | 2097152 |
| 8 | 16 | 256 | 2048 | 65536 | 16777216 |
| 9 | 23 | 512 | 4608 | 262144 | 134217728 |
| 10 | 32 | 1024 | 10240 | 1048576 | 1073741824 |
| 11 | 45 | 2048 | 22528 | 4194304 | 8589934592 |
| 12 | 64 | 4096 | 49152 | 16777216 | 68719476736 |

# Contagem de Instruções (Tempo)

(`linear_search`) – pior caso

$O(N)$

```
1   #include <vector>
2   #include <cstddef>
3
4   template <typename T>
5   static std::size_t linear_search(const std::vector<T>& xs, const T& x)
6   {
7       for (std::size_t i = 0; i < xs.size(); ++i)
8           if (xs[i] == x)
9               return i;
10      return NOT_FOUND;
11  }
```

| Linha | Tempo (Contagem) | Código |
|---|---|---|
| 7a | $T_{load} + T_{store}$ | `std::size_t i = 0` |
| 7b | $(2T_{load} + T_{<}) * (N + 1)$ | `i < xs.size()` |
| 7c | $(2T_{load} + T_{+} + T_{store}) * N$ | `++i` |
| 8 | $(4T_{load} + T_{[]} + T_{==}) * N$ | `if (xs[i] == x)` |
| 9 | $T_{load} + T_{return}$ | `return i` |
| 10 | $T_{return}$ | `return NOT_FOUND` |
| **Total (Σ)** | $N[8T_{load} + T_{store} + T_{<} + T_{+} + T_{[]} + T_{==}] + 3T_{load} + T_{store} + T_{<} + T_{return}$ | |

# Contagem de Instruções (Tempo)

(`linear_search`) – pior caso

$O(N)$

**Caso NOT_FOUND**

Troca de variáveis:

$$A + B + (2N+2)A + (N+1)C + (2N)A + (N)D + N(B) + (4N)A + (N)E + (N)F + G$$

Isolando os termos:

| + | | | | |
|---|---|---|---|---|
| | $A + (2N)A + (2)A + (2N)A + (4N)A = (8N)A + (3)A = (8N+3)A$ | | | |
| | $B + (N)B = (N+1)B$ | | $(N+1)C$ | |
| | $(N)D$ | $(N)E$ | $(N)F$ | $G$ |

Somando:

$$(8\mathrm{N}+3)\mathrm{A} + (\mathrm{N}+1)\mathrm{B} + (\mathrm{N}+1)\mathrm{C} + (\mathrm{N})\mathrm{D} + (\mathrm{N})\mathrm{E} + (\mathrm{N})\mathrm{F} + \mathrm{G}$$

Organizando:

$$N[8A + B + C + D + E + F] + 3A + B + C + G$$

$$\boldsymbol{N[8T_{load} + T_{store} + T_{<} + T_{+} + T_{[]} + T_{==}] + 3T_{load} + T_{store} + T_{<} + T_{return}}$$

# Contagem de Tempo (Simplificado)

## (`linear_search`) – pior caso

$O(N)$

```
1   #include <vector>
2   #include <cstddef>
3
4   template <typename T>
5   static std::size_t linear_search(const std::vector<T>& xs, const T& x)
6   {
7       for (std::size_t i = 0; i < xs.size(); ++i)
8           if (xs[i] == x)
9               return i;
10      return NOT_FOUND;
11  }
```

Uma unidade de tempo por instrução

| Linha | Tempo (Contagem) | Código |
|---|---|---|
| 7a | 1 | `std::size_t i = 0` |
| 7b | N + 1 | `i < xs.size()` |
| 7c | N | `++i` |
| 8 | N | `if (xs[i] == x)` |
| 9 | 1 | `return i` |
| 10 | 1 | `return NOT_FOUND` |
| **Total (∑)** | $3N + 3$ | |

# Contagem de Tempo (Simplificado)

## (`linear_search`) – melhor caso

$\Omega(1)$

```
1   #include <vector>
2   #include <cstddef>
3
4   template <typename T>
5   static std::size_t linear_search(const std::vector<T>& xs, const T& x)
6   {
7       for (std::size_t i = 0; i < xs.size(); ++i)
8           if (xs[i] == x)
9               return i;
10      return NOT_FOUND;
11  }
```

Uma unidade de tempo por instrução

| Linha | Tempo (Contagem) | Código |
|---|---|---|
| 7a | 1 | `std::size_t i = 0` |
| 7b | 1 | `i < xs.size()` |
| 7c | 0 | `++i` |
| 8 | 1 | `if (xs[i] == x)` |
| 9 | 1 | `return i` |
| 10 | 1 | `return NOT_FOUND` |
| **Total (∑)** | 4 | |

# Contagem de Espaço (Simplificado)

(linear_search)

$O(N)$

```
1   #include <vector>
2   #include <cstddef>
3
4   template <typename T>
5   static std::size_t linear_search(const std::vector<T>& xs, const T& x)
6   {
7       for (std::size_t i = 0; i < xs.size(); ++i)
8           if (xs[i] == x)
9               return i;
10      return NOT_FOUND;
11  }
```

Uma unidade de espaço

| Linha | Espaço (Contagem) | Código |
|---|---|---|
| 5a | N | `const std::vector<T>& xs` |
| 5b | 1 | `const T& x` |
| 7a | 1 | `std::size_t i = 0` |
| **Total (∑)** | $N + 2$ | |

# Contagem de Tempo (Simplificado)

## (linear_search_with_sentinel)

$O(N)$

```
1   #include <vector>
2   #include <cstddef>
3
4   template <typename T>
5   static std::size_t linear_search_with_sentinel(std::vector<T>& xs, const T& x)
6   {
7       std::size_t n = xs.size() - 1;
8       T last = xs[n];
9       xs[n] = x;
10      std::size_t i = 0;
11      while (xs[i] != x)
12          ++i;
13      xs[n] = last;
14      if (i < n || xs[n] == x)
15          return i;
16      return NOT_FOUND;
17  }
```

Uma unidade de tempo por instrução

| Linha | Tempo (Contagem) | Código |
|---|---|---|
| 7 | 1 | `std::size_t n = xs.size() - 1` |
| 8 | 1 | `T last = xs[n]` |
| 9 | 1 | `xs[n] = x` |
| 10 | 1 | `std::size_t n = 0` |
| 11 | N | `while (xs[i] != x)` |
| 12 | N | `++i` |
| 13 | 1 | `xs[n] = last` |
| 14 | 1 | `if (i < n \|\| xs[n] == x)` |
| 15 | 1 | `return i` |
| 16 | 1 | `return NOT_FOUND` |
| **Total (∑)** | $2N + 7$ | |

# Contagem de Tempo

(insertsort)

$O(N^2)$

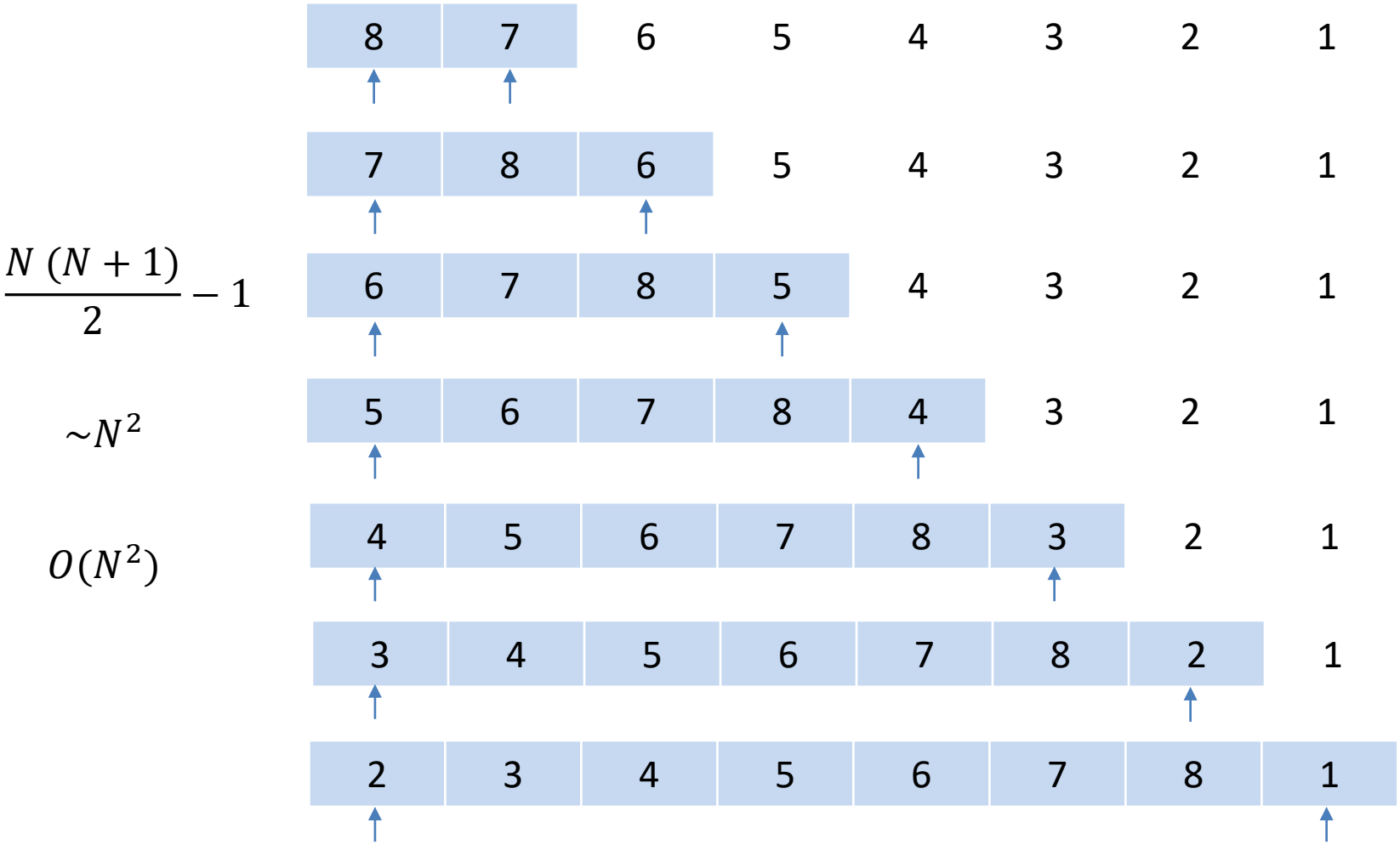
- Número de comparações

```
template <typename Container>
static inline void insertsort(Container& xs)
{
    const std::size_t N = xs.size();
    for (std::size_t i = 1; i < N; ++i)
    {
        for (std::size_t j = i; j > 0; --j)
        {
            if (xs[j] < xs[j - 1])
                std::swap(xs[j], xs[j - 1]);
            else
                break;
        }
    }
}
```

# Contagem de Tempo

(insertsort)

- *Worst-case* $O(N^2)$

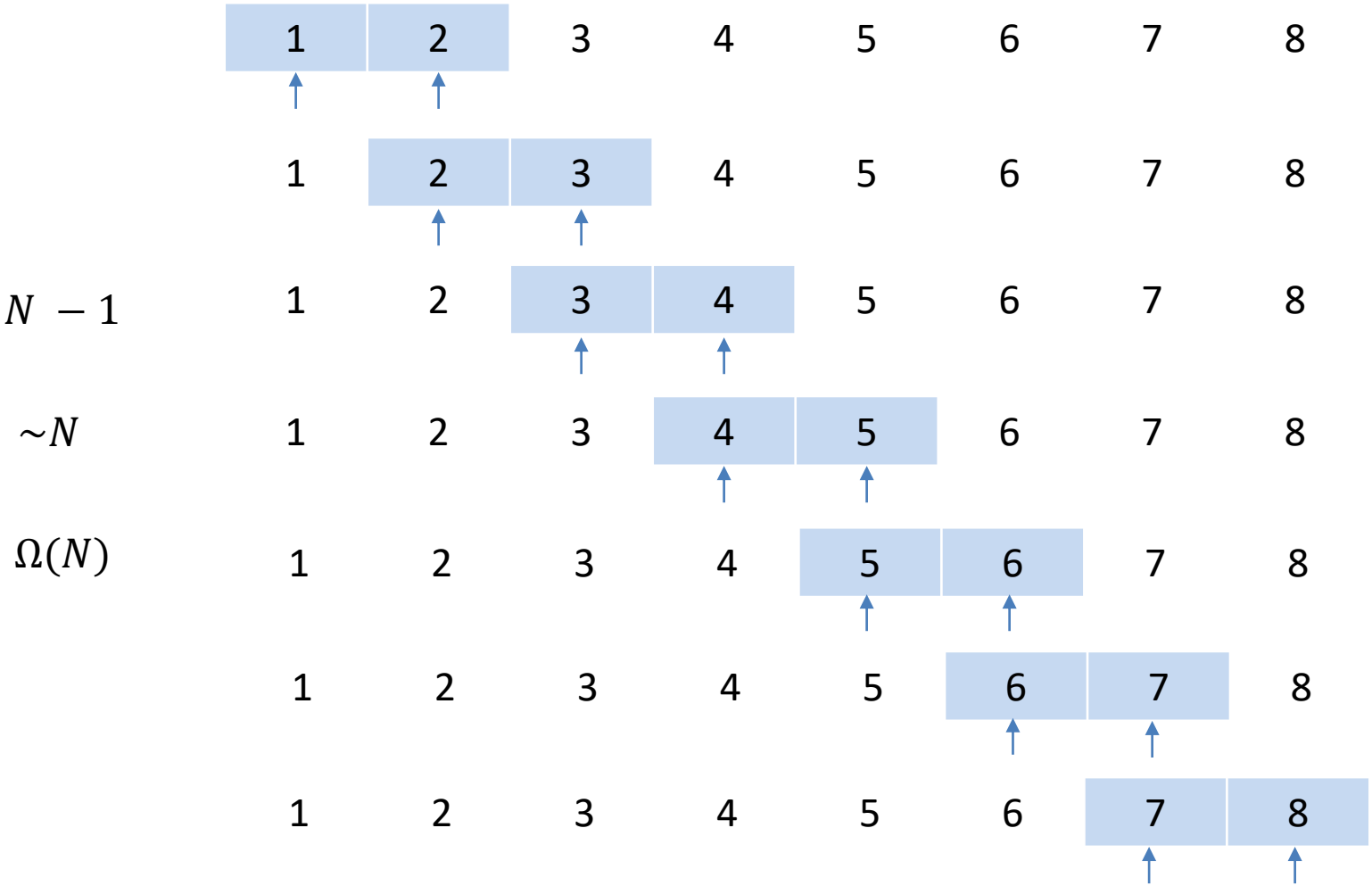


# Contagem de Tempo

(insertsort)

$\Omega(N)$

- *Best-case*



# Contagem de Tempo

(binary_search)

$O(lgN)$

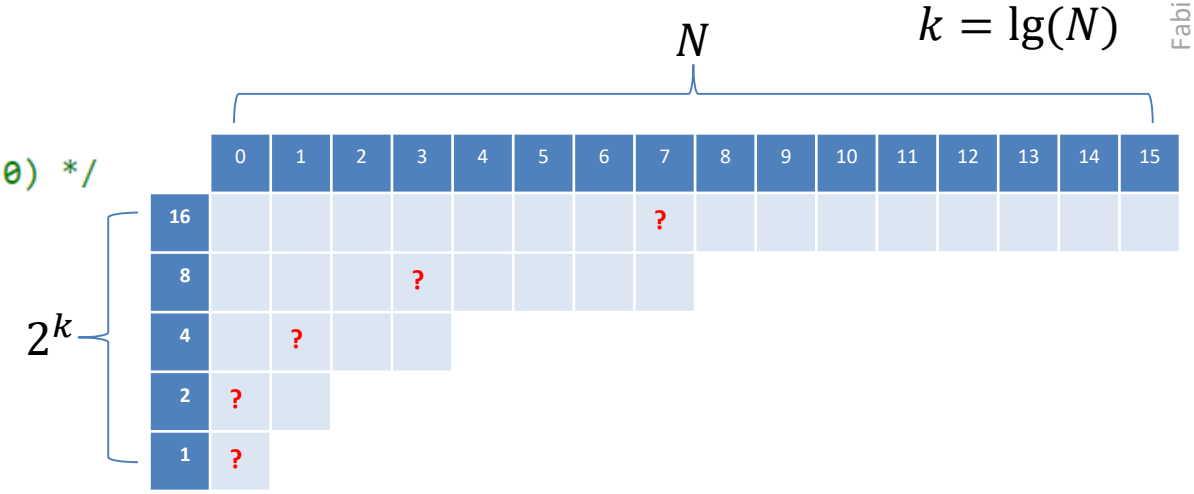
```
1   #include <vector>
2   #include <cstddef>
3
4   template <typename T, typename Comparer>
5   std::size_t binary_search(const std::vector<T>& xs, const T& x, Comparer compare)
6   {
7       std::size_t l = 0, r = xs.size(), mid;
8       while (l < r)
9       {
10          mid = l + (r - l) / 2;
11          int comp = compare(x, xs[mid]);
12          if (comp == 0)
13              return mid;
14          else if (comp < 0)
15              r = mid;
16          else /* if (comp > 0) */
17              l = mid + 1;
18      }
19      return NOT_FOUND;
20  }
```

$$2^k = N$$

$$\lg(2^k) = \lg(N)$$

$$k \lg(2) = \lg(N)$$

$$k = \lg(N)$$



# Contagem de Tempo (Simplificado)

(std::distance InputIterator)

$\Theta(N)$

- Pior caso e Melhor caso são iguais

```
78    template<typename _InputIterator>
79      inline _GLIBCXX14_CONSTEXPR
80      typename iterator_traits<_InputIterator>::difference_type
81      __distance(_InputIterator __first, _InputIterator __last,
82                 input_iterator_tag)
83      {
84        // concept requirements
85        __glibcxx_function_requires(_InputIteratorConcept<_InputIterator>)
86
87        typename iterator_traits<_InputIterator>::difference_type __n = 0;
88        while (__first != __last)
89          {
90            ++__first;
91            ++__n;
92          }
93        return __n;
94      }
```

Uma unidade de tempo por instrução

| Linha | Tempo (Contagem) | Código |
|---|---|---|
| 87 | 1 | difference_type __n = 0 |
| 88 | N + 1 | while (__first != __last) |
| 90 | 1 | ++__first |
| 91 | 1 | ++__n |
| 93 | 1 | return __n |
| **Total (∑)** | $N + 5$ | |

https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/include/bits/stl_iterator_base_funcs.h#L81

# Contagem & Complexidade de Tempo

```
long count = 0;
for (int i = 0; i < N; ++i)
    ++count;
```

$$N$$

---

```
long count = 0;
for (int i = 1; i < N; i = 2 * i)
    ++count;
```

$$\lg N$$

# Contagem & Complexidade de Tempo

```
long count = 0;
for (int i = 0; i < N; ++i)
    for (int j = 0; j < i; ++j)
        ++count;
```

$$\frac{N(N+1)}{2}$$

| i | j | contagem |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0,1 | 1 |
| 2 | 0,1,2 | 2 |
| 3 | 0,1,2,3 | 3 |
| ... | | |
| N | 0,1,2,3,...,N | N |

$1 + 2 + 3 + \cdots + N$

$$T = 1 + 2 + 3 + 4 + 5 + \cdots + N - 1 + N$$

Gauss trick

$$+ \quad \begin{aligned} T &= 1 + 2 + 3 + 4 + 5 + \cdots + N - 1 + N \\ T &= N + N - 1 + \cdots + 5 + 4 + 3 + 2 + 1 \end{aligned}$$

$$2T = \underbrace{(N+1) + (N+1) + \cdots + (N+1) + (N+1)}_{N}$$

$$2T = N(N+1) \rightarrow \boxed{T = \frac{N(N+1)}{2}}$$

# Contagem & Complexidade de Tempo

```
long count = 0;
for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
        ++count;
```

$$N^2$$

---

```
long count = 0;
for (int i = 0; i < N; ++i)
    for (int j = 0; j < N; ++j)
        for (int k = 0; k < N; ++k)
            ++count;
```

$$N^3$$

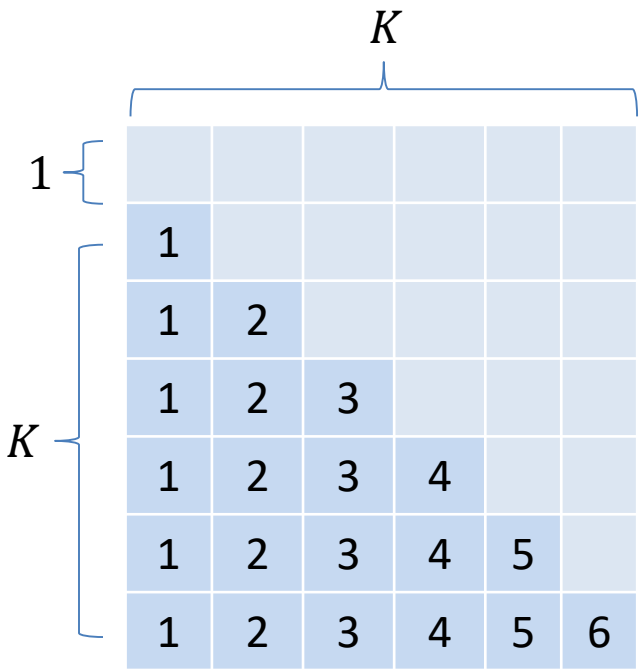
# Contagem & Complexidade de Tempo

```
long count = 0;
for (int i = 1; count <= N; ++i)
        count += i;
```

$$\sqrt{N}$$

| i | contagem |
|---|---|
| 1 | 1 |
| 2 | 3 |
| 3 | 6 |
| 4 | 10 |
| 5 | 15 |
| … | |
| K | N |

$1 + 3 + 6 + 10 + 15 + \cdots$




$$\frac{K(K+1)}{2} = N$$

$$\frac{K^2 + K}{2} = N$$

$$O\left(\frac{K^2 + K}{2}\right) = O(K^2)$$

$$K^2 \cong N \rightarrow \boxed{K \cong \sqrt{N}}$$

# Contagem & Complexidade de Tempo

```
long count = 0;
for (int i = 1; i * i <= N; ++i)
    for (int j = 1; j * j <= N; ++j)
        for (int k = 1; k * k <= N; ++k)
            for (int l = 1; l <= 4; ++l)
                ++count;
```

| i | i * i |
|---|---|
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |
| 5 | 25 |

$$4\sqrt{N^3}$$

---

```
long count = 0;
std::vector<int> xs(N, 0);
for (int i = 0; i < N; ++i) //N
{
    xs[i] = i + 1;
    ++count;
}
std::shuffle(xs.begin(), xs.end(), rnd); //N - 1
count += (N - 1);
std::binary_search(xs.begin(), xs.end(), 42); //lg N
count += std::log(N) / std::log(2);
```

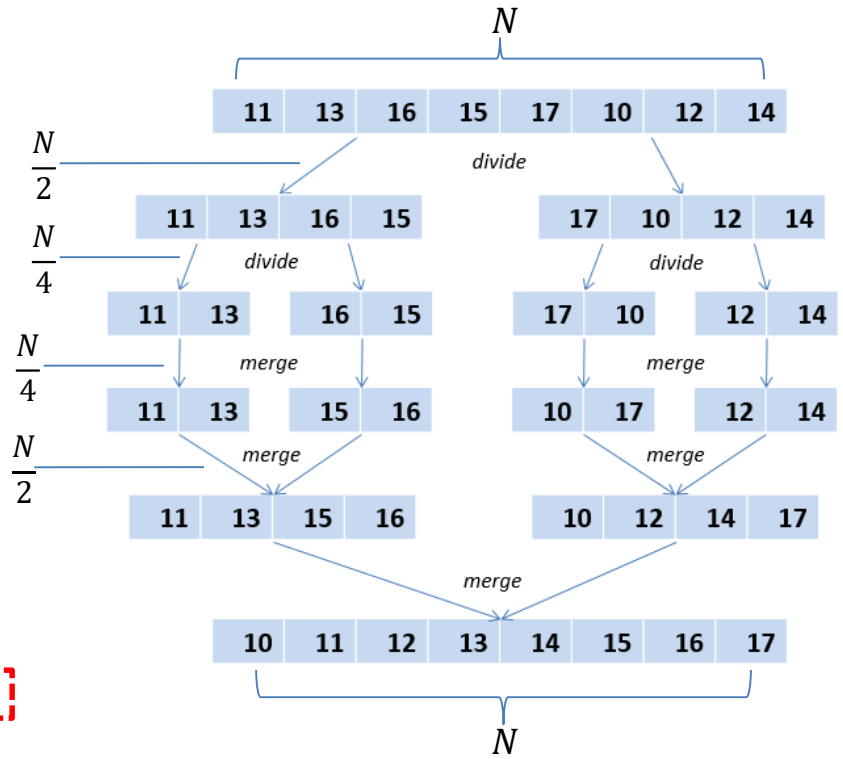
$$2N + lg\ N - 1$$

# Contagem de Tempo

## (mergesort)

$O(N\ lgN)$

```
1   #include <vector>
2   #include <cstddef>
3
4   template <typename T>
5   static inline void mergesort(std::vector<T>& xs, std::size_t first, std::size_t last)
6   {
7       const std::size_t N = last - first;
8       if (N < 2)
9       {
10          return;
11      }
12      if (N == 2)
13      {
14          if (xs[first + 1] < xs[first])
15              std::swap(xs[first + 1], xs[first]);
16          return;
17      }
18
19      const std::size_t mid = N / 2;
20      mergesort(xs, first, first + mid);
21      mergesort(xs, first + mid, last);
22
23      T* beg_ptr = &xs[first];
24      T* mid_ptr = beg_ptr + mid;
25      T* end_ptr = beg_ptr + N;
26      std::inplace_merge(beg_ptr, mid_ptr, end_ptr);
27  }
```

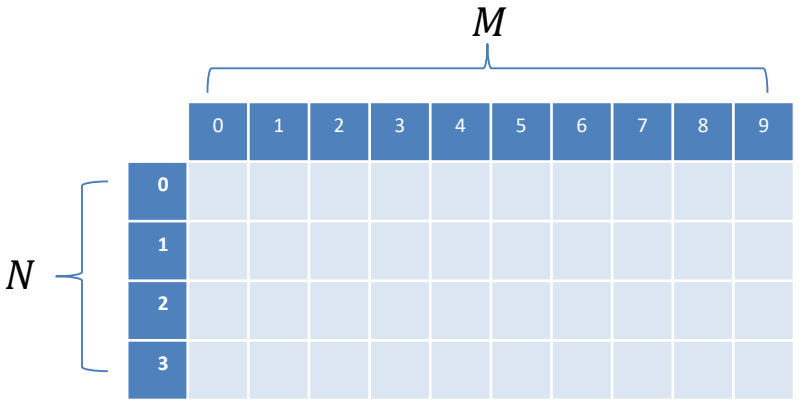


# Contagem de Tempo

## (water_recipient_filling)

$O(NM)$

```cpp
#include <vector>
#include <algorithm>
#include <cstddef>

static std::size_t water_recipient_filling(std::size_t recipient_capacity, std::vector<std::size_t> bottles)
{
    const std::size_t INF = -1;
    std::sort(bottles.begin(), bottles.end());

    std::size_t rows = bottles.size();
    std::size_t cols = recipient_capacity + 1;
    std::vector<std::size_t> table(rows * cols, 0);
    for (std::size_t row = 0; row < rows; ++row)
    {
        for (std::size_t col = 1; col < cols; ++col)
        {
            if (col < bottles[row])
            {
                table[row * cols + col] = row > 0 ? table[(row - 1) * cols + col] : INF;
            }
            else
            {
                std::size_t   up = table[(row - 1) * cols + col];
                std::size_t left = table[row * cols + (col - bottles[row])];
                table[row * cols + col] = std::min(row > 0 ? up : INF, left != INF ? 1 + left : INF);
            }
        }
    }

    std::size_t result = table[(rows - 1) * cols + (cols - 1)];
    return result;
}
```



# Pior caso e os algoritmos aplicados até o momento
(Tempo)

| Algoritmos/Funções | Complexidade de Tempo |
|---|---|
| linear_search, std::find, std::count, std::count_if | O(N) |
| merge_copy, std::merge, inplace_merge, std::inplace_merge | O(N) |
| mergesort, std::stable_sort | O(N lg N) |
| binary_search, std::binary_search | O(lg N) |
| std::lower_bound, std::upper_bound, between | O(lg N) |
| strip_left_if, strip_right_if, strip_if | O(N) |
| inplace_partition, std::partition | O(N) |
| std::swap, max, min, clamp | O(1) |
| std::max_element, std::min_element | O(N) |
| std::push_heap, std::pop_heap | O(lg N) |
| std::make_heap | O(N) |
| heapsort, std::partial_sort | O(N lg N) |
| Gale-Shapley | O(N^2) |

# Recursão

(Relação de Recorrência)

- Dividir e Conquistar

$$T(N) = \begin{cases} T(\text{solução direta}), & n \in \text{solução direta} \\ T(\text{dividir}) + \sum_{1}^{\text{qtde rec.}} T(\text{solução recursiva}) + T(\text{combinar}), & \text{senão} \end{cases}$$

- Definindo a Relação de Recorrência para Fatorial

```
unsigned int factorial_recursive(unsigned int N)
{
    //Direct solution
    if (N <= 1)
        return 1;

    //Divide
    unsigned int M = N - 1;

    //Recurse
    unsigned int partial_result = factorial_recursive(M);

    //Combine
    unsigned int result = N * partial_result;

    return result;
}
```

$$T(N) = \begin{cases} 1, & if\ n \leq 1 \\ 1 + 1\,T(N-1) + 1, & \text{senão} \end{cases}$$

# Recursão

## (Resolvendo a Relação de Recorrência para Fatorial por Substituição)

- Fatorial

$$\boxed{T(N) = \begin{cases} 1, & if\ n \leq 1 \\ 1\ + 1\ T(N-1) + 1, & senão \end{cases}}$$

$$T(N) = T(N-1) + 2 \quad \text{(equação 1)}$$

$$T(1) = 1 \qquad T(0) = 1$$

Substituir T(N -1), ... na equação 1

$$T(N-1) = T((N-1)-1) + 2 = T(N-2) + 2$$

$$T(N-2) = T(N-3) + 2$$

$$T(N-3) = T(N-4) + 2$$

...

$$T(N-N+2) = T((N-N+2)-1) + 2 = T(1) + 2$$

Substituindo de volta em T(N)

$$T(N) = (T(N-2) + 2) + 2$$

$$T(N) = ((T(N\ -3) + 2) + 2) + 2$$

$$T(N) = (((T(N-4) + 2) + 2) + 2) + 2$$

...

$$T(N) = T(N-N+1) + (N-1)\ 2 = T(1) + 2N\ -2 = 1 + 2N\ -2 = \boxed{2N\ -1}$$

$$\sim 2N$$

$$O(N)$$

# Recursão

## (Definindo a Relação de Recorrência para Mergesort)

- Mergesort

$$T(N) = \begin{cases} 1, & if\ n \leq 2 \\ 1 + \sum_{1}^{2} T\left(\frac{N}{2}\right) + N, & senão \end{cases}$$

```
template <typename T>
static inline void mergesort(std::vector<T>& xs, std::size_t first, std::size_t last)
{
    const std::size_t N = last - first;
    if (N < 2)
    {
        return;
    }
    if (N == 2)
    {
        if (xs[first + 1] < xs[first])
            std::swap(xs[first + 1], xs[first]);
        return;
    }

    const std::size_t mid = N / 2;
    mergesort(xs, first, first + mid);
    mergesort(xs, first + mid, last);

    T* beg_ptr = &xs[first];
    T* mid_ptr = beg_ptr + mid;
    T* end_ptr = beg_ptr + N;
    std::inplace_merge(beg_ptr, mid_ptr, end_ptr);
}
```

$$T(N) = 2T\left(\frac{N}{2}\right) + N + 1 \quad \text{(equação 1)}$$

$$T(2) = 1$$
$$T(1) = 1$$
$$T(0) = 1$$

Tratando o melhor caso e o pior caso da solução direta por igual

(simplificando equação 1)

$$T(N) = 2T\left(\frac{N}{2}\right) + N$$

# Recursão

## (Resolvendo a Relação de Recorrência do Mergesort por Substituição)

- Mergesort $\boxed{T(N) = 2T\left(\frac{N}{2}\right) + N}$ (equação 1)

Substituir $T(N/2)$, ... na equação 1

$$T(N/2) = 2T((N/2)/2) + N/2 = 2T(N/4) + N/2$$
$$T(N/4) = 2T((N/4)/2) + N/4 = 2T(N/8) + N/4$$
...
$$T(N/(N/4)) = 2T(N/(N/4)/2) + N/(N/4) = 2T(4/2) + 4 = 2T(2) + 4$$

Substituindo de volta em $T(N)$

$$T(N) = 2(2T(N/4) + N/2) + N = 4T(N/4) + 2N$$
$$T(N) = 4T(2T(N/8) + N/4) + 2N = 8T(N/8) + 3N$$
...
$$T(N) = NT(N/N) + (lg\ N)N = NT(1) + N\ lg\ N = N + N\ lg\ N$$

$$\left.\begin{matrix} O(N\ lgN) \\ \Omega(N\ lgN) \end{matrix}\right\} \Longrightarrow \Theta(N\ lgN)$$

# Recursão

(O Método Mestre simplificado para Recorrências)

- Recorrência na forma:

$$T(N) = aT\left(\frac{N}{b}\right) + f(N)$$

$onde: a \geq 1,\ b > 1,$
$f\ assintoticamente\ positiva$

- $f(N)\ representa\ T(dividir)\ e\ T(combinar)$

- 3 casos comuns para $f(N)$

$$T(N) = \begin{cases} \Theta(N^d), & if\ a < b^d \\ \Theta(N^d lgN), & if\ a = b^d \\ \Theta(N^{log_b a}), & if\ a > b^d \end{cases} \qquad f(N) = N^{\mathrm{d}}$$

- Exemplo Mergesort $T(N) = 2T\left(\frac{N}{2}\right) + N, a = 2, b = 2, d = 1$

$$T(N) = \Theta(N\ lgN)$$

# Estudo de Caso: Análise do `heapify`

(parte 1 de 3)

- A função `heapify` é implementado em termos de `sift_down`
- Equivalente a `std::make_heap`

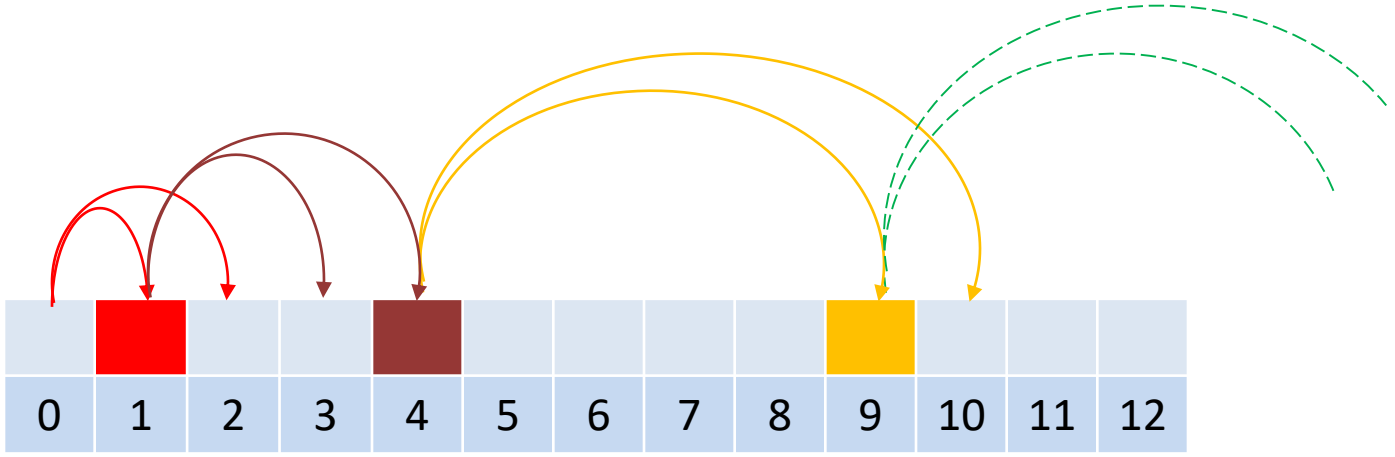
```
void heapify(std::vector<int>& xs)
{
    const std::size_t N = xs.size();
    if (N > 1)
    {
        for (std::size_t i = (N - 2) / 2; i != -1; --i)
            sift_down(xs, i);
    }
}
```

```
//sift down one
void sift_down(std::vector<int>& xs, std::size_t i)
{
    const std::size_t N = xs.size();

    if (N < 2 || (N - 2) / 2 < i)
        return;

    std::size_t j = 2 * i + 1; //child
    if (j + 1 < N && xs[j] < xs[j + 1]) ++j;

    if (xs[j] < xs[i]) //child < parent
        return;

    int temp = xs[i];
    do
    {
        if ((N - 2) / 2 < i)
            break;

        xs[i] = xs[j];
        i = j;
        j = 2 * j + 1;
        if (j + 1 < N && xs[j] < xs[j + 1]) ++j;
    }
    while (!(xs[j] < temp));
    xs[i] = temp;
}
```

# Estudo de Caso: Análise do `heapify`

(parte 2 de 3)

- A função `sift_down` tem, no pior caso, complexidade logarítmica quando usada arbitrariamente
  - Todos elementos precisam ser visitados caso a entrada não esteja organizada como *binary heap*, onde a função fará os ajustes relativos a partir do índice fornecido

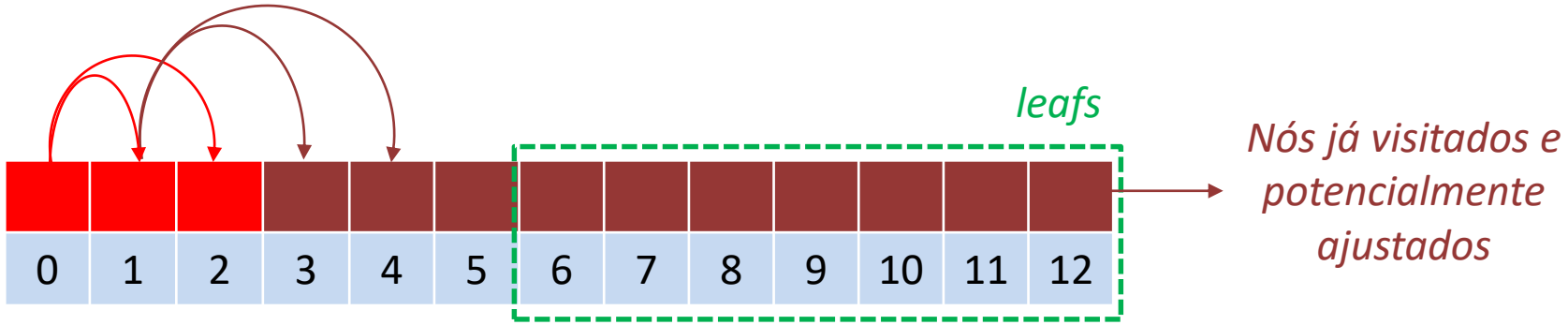



$N = 13, \qquad lgN \cong 3{,}7005$ Complexidade da `sift_down`: $\boxed{T(N) = lgN}$

# Estudo de Caso: Análise do `heapify`

(parte 3 de 3)

- A função `sift_down` quando usada dentro da função `heapify` é amortizada. Todos os nós pais são processados em ordem decrescente, mantendo os nós filhos ajustados, isso elimina as comparações subsequentes
  - Logo, a função `sift_down` se torna constante em 3
    - Se `i = 0`, somente os indices 1 e 2 se necessário sofrerão ajustes, visto que os nós filhos foram previamente organizados durante o *loop* da função `heapify`




Complexidade da `heapify`: $T(N) = 3\left(\frac{N-2}{2}\right)$

# Equação da Reta e Inclinação

$$Ax + By = C$$

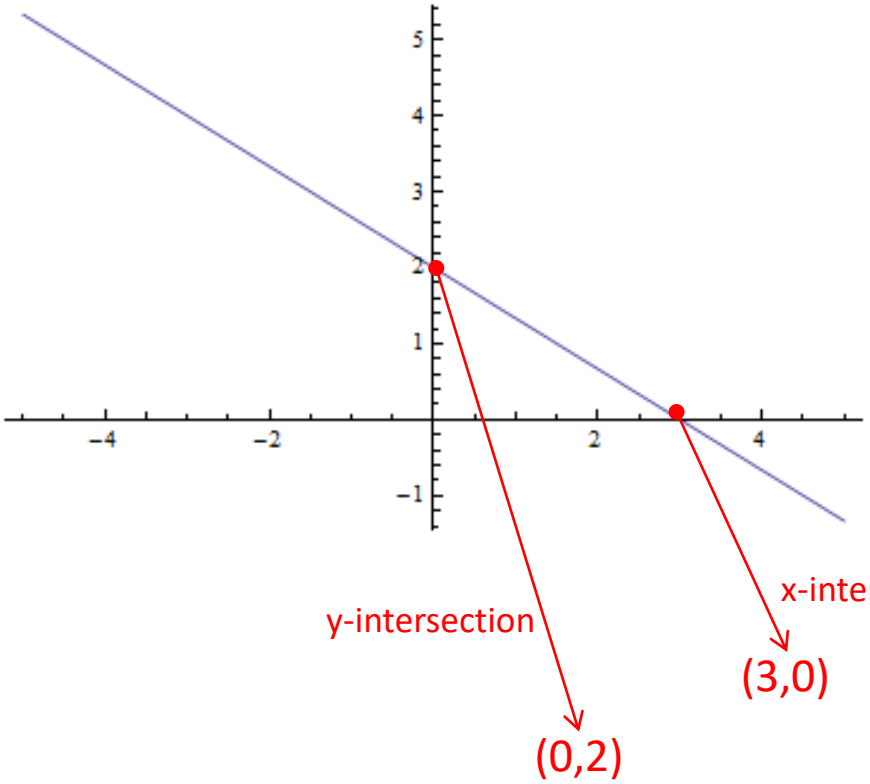
$$Y = m \times X + b$$

slope intercept form

$$2x + 3y = 6$$

$$Y = -\frac{2}{3} \times X + 2$$

```
Plot[{(-2 x + 6) / 3}, {x, -5, 5}]
```




```
A: 2
B: 3
C: 6
direction: decreasing
intersection point(s): (3, 0) (0, 2)
angle in degrees:
-33.6901
graph:
Plot[{(-2x+6)/3},{x,-5,5}]
```

y-intercept $b = \frac{6}{3} = 2$

slope $m = -\frac{2}{3} = \sim -0.66667$

$$\frac{\arctan(-0.66667) * 180}{\pi} = -33.6901997^{\circ}$$

# Equação da Reta e Inclinação

$$Ax + By = C$$

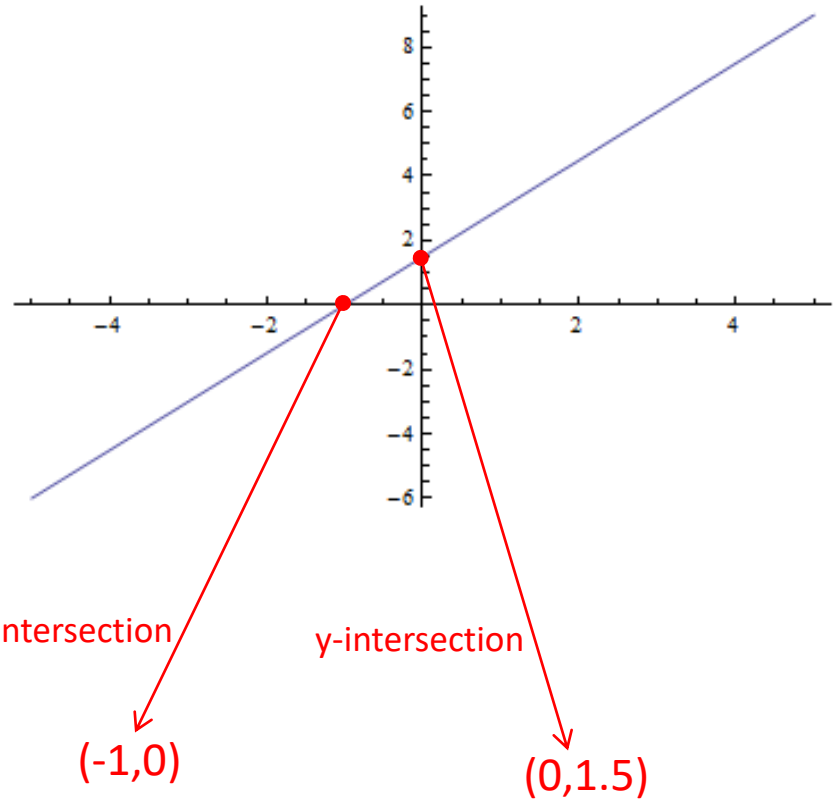
$$Y = m \times X + b$$

slope intercept form

$$-3x + 2y = 3$$

$$Y = 1.5 \times X + 1.5$$

```
Plot[{(3 x + 3) / 2}, {x, -5, 5}]
```




```
A: -3
B: 2
C: 3
direction: increasing
intersection point(s): (-1, 0) (0, 1.5)
angle in degrees:
56.3099
graph:
Plot[{(3x+3)/2},{x,-5,5}]
```

y-intercept $b = \frac{3}{2} = 1.5$

slope $m = +\frac{3}{2} = 1.5$

$$\frac{\arctan(1.5) * 180}{\pi} = 56.3099325^{\circ}$$

# Análise Empírica

(*benchmark*)

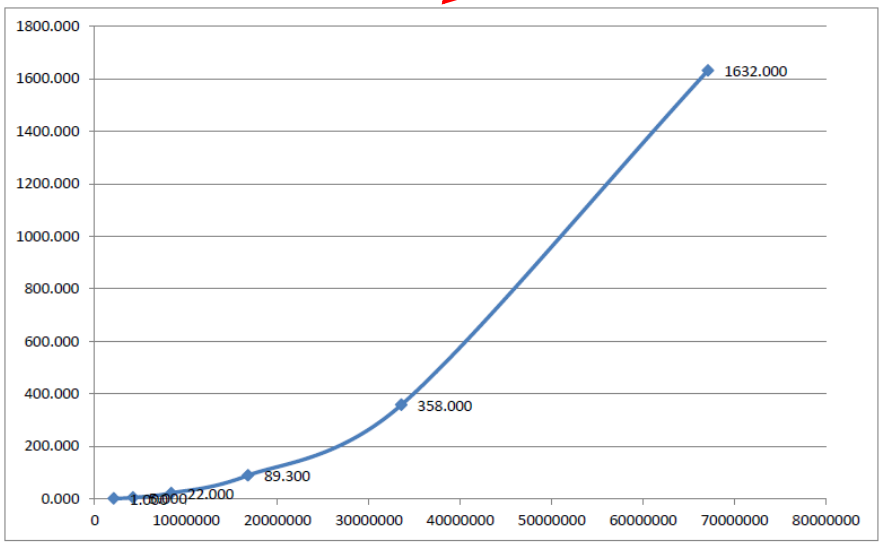
- Medir seu algoritmo (ou coletar os tempos de execução) variando a entrada de maneira crescente, mantendo a base e variando o expoente linearmente :
  - Por exemplo: $2^8, 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}, 2^{15}, \ldots$

| Name (baseline is *) | Dim | Total ms | ns/op | Baseline | Ops/second |
|---|---|---|---|---|---|
| bench_count * | 2048 | 0.072 | 35 | - | 28296672.9 |
| bench_count * | 4096 | 0.147 | 35 | - | 27889938.2 |
| bench_count * | 8192 | 0.324 | 39 | - | 25316690.4 |
| bench_count * | 16384 | 0.611 | 37 | - | 26802993.7 |
| bench_count * | 32768 | 1.201 | 36 | - | 27273915.2 |
| bench_count * | 65536 | 2.360 | 36 | - | 27765244.4 |
| bench_count * | 131072 | 4.718 | 35 | - | 27782994.5 |
| bench_count * | 262144 | 9.590 | 36 | - | 27336526.1 |
| bench_count * | 524288 | 18.822 | 35 | - | 27854657.7 |
| bench_count * | 1048576 | 39.652 | 37 | - | 26444416.2 |

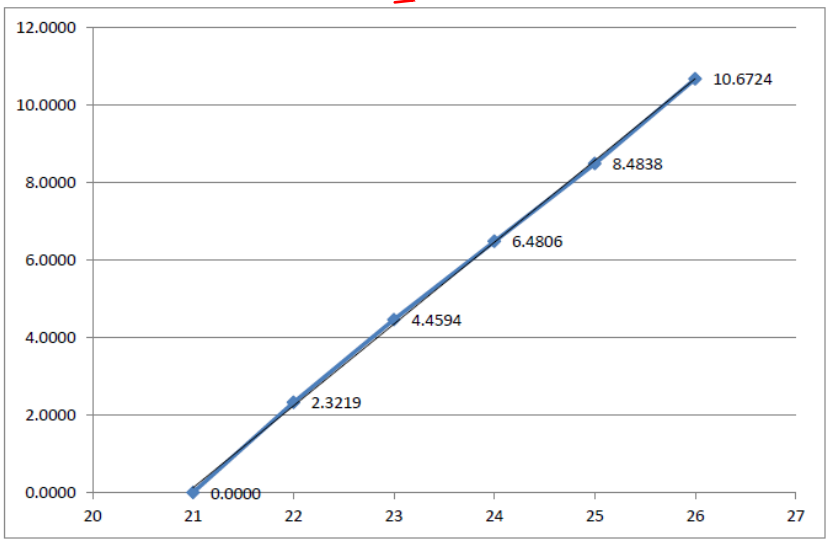
# Análise Empírica

## (análise dos dados)

| N | T | LG N | LG T | RATIO | LG RATIO |
|---|---|---|---|---|---|
| 1048576 | 0.220 | 20 | -2.1844 | #VALUE! | #VALUE! |
| 2097152 | 1.000 | 21 | 0.0000 | 4.5455 | 2.1844 |
| 4194304 | 5.000 | 22 | 2.3219 | 5.0000 | 2.3219 |
| 8388608 | 22.000 | 23 | 4.4594 | 4.4000 | 2.1375 |
| 16777216 | 89.300 | 24 | 6.4806 | 4.0591 | 2.0212 |
| 33554432 | 358.000 | 25 | 8.4838 | 4.0090 | 2.0032 |
| 67108864 | 1632.000 | 26 | 10.6724 | 4.5587 | 2.1886 |




Standard plot: T(N) x N




Log-log plot: : log(T(N)) x log(N)

# Análise Empírica

## (tempo de execução estimado)

binary_search

| N | T | log N | log T | ratio | log ratio |
|---|---|---|---|---|---|
| 2048 | 5.6670 | 11 | 2.5026 | -inf | -inf |
| 4096 | 5.8940 | 12 | 2.5592 | 1.0401 | 0.0567 |
| 8192 | 6.6100 | 13 | 2.7247 | 1.1215 | 0.1654 |
| 16384 | 6.8100 | 14 | 2.7677 | 1.0303 | 0.0430 |
| 32768 | 7.4360 | 15 | 2.8945 | 1.0919 | 0.1269 |
| 65536 | 7.9470 | 16 | 2.9904 | 1.0687 | 0.0959 |
| 131072 | 8.3950 | 17 | 3.0695 | 1.0564 | 0.0791 |
| 262144 | 9.5800 | 18 | 3.2600 | 1.1412 | 0.1905 |
| 524288 | 10.0480 | 19 | 3.3288 | 1.0489 | 0.0688 |
| 1048576 | 10.5730 | 20 | 3.4023 | 1.0522 | 0.0735 |

Estimated runnning time is 3.8179e+000 x N^0.0735 ms

count

| N | T | log N | log T | ratio | log ratio |
|---|---|---|---|---|---|
| 2048 | 0.0720 | 11 | -3.7959 | -inf | -inf |
| 4096 | 0.1470 | 12 | -2.7661 | 2.0417 | 1.0297 |
| 8192 | 0.3240 | 13 | -1.6259 | 2.2041 | 1.1402 |
| 16384 | 0.6110 | 14 | -0.7108 | 1.8858 | 0.9152 |
| 32768 | 1.2010 | 15 | 0.2642 | 1.9656 | 0.9750 |
| 65536 | 2.3600 | 16 | 1.2388 | 1.9650 | 0.9746 |
| 131072 | 4.7180 | 17 | 2.2382 | 1.9992 | 0.9994 |
| 262144 | 9.5900 | 18 | 3.2615 | 2.0326 | 1.0234 |
| 524288 | 18.8220 | 19 | 4.2343 | 1.9627 | 0.9728 |
| 1048576 | 39.6520 | 20 | 5.3093 | 2.1067 | 1.0750 |

Estimated runnning time is 1.3375e-005 x N^1.0750 ms

# Análise Empírica

(considerações sobre medição)

- Medir com precisão é um grande desafio

- Efeitos independentes do sistema
  - Algoritmo e sua complexidade
  - Quantidade de dados (N)
  - Influência o expoente e a constante na lei da potência

- Efeitos dependentes do sistema
  - Hardware
    - CPU, Memória, Cache, …
  - Software
    - Compilador, Máquina virtual, GC, …
    - Sistema operacional, Rede, …
  - Influência a constante na lei da potência

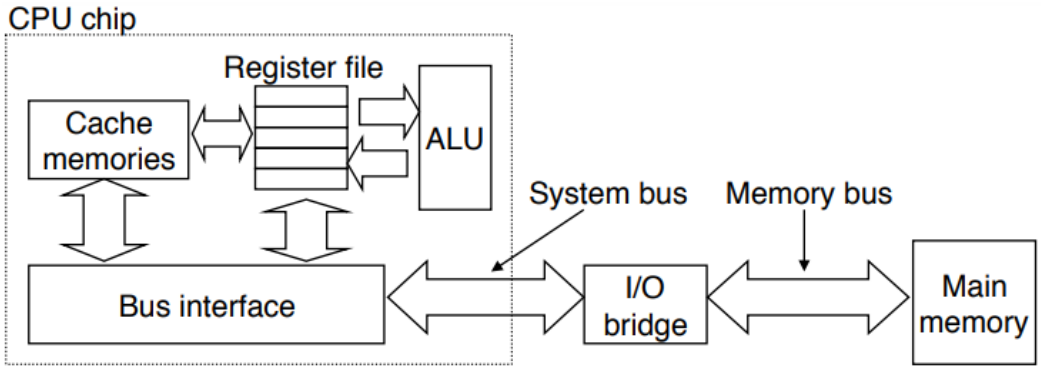
Ref.: http://algs4.cs.princeton.edu/14analysis/

$Lei\ da\ Potência\colon\ aN^b$

# Arquitetura de Computadores

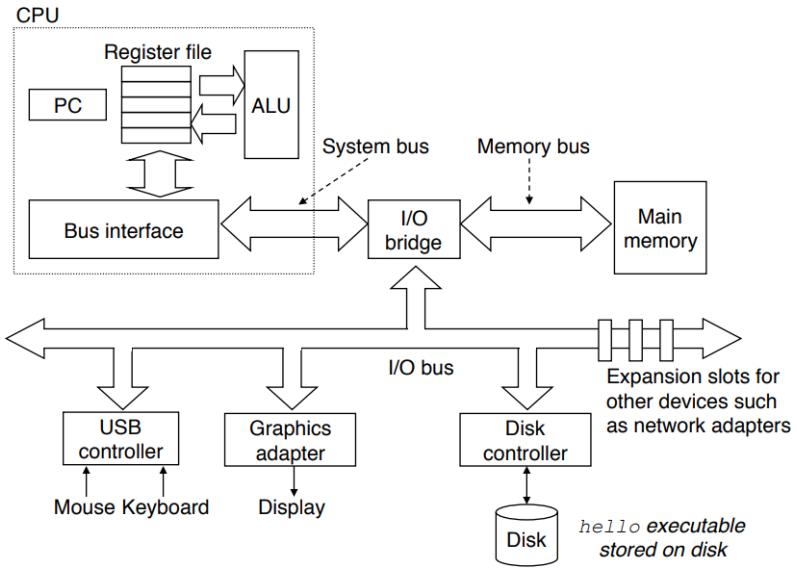
(*hardware*)

- Cache + Bus




http://csapp.cs.cmu.edu/3e/ics3/intro/cachebus.pdf

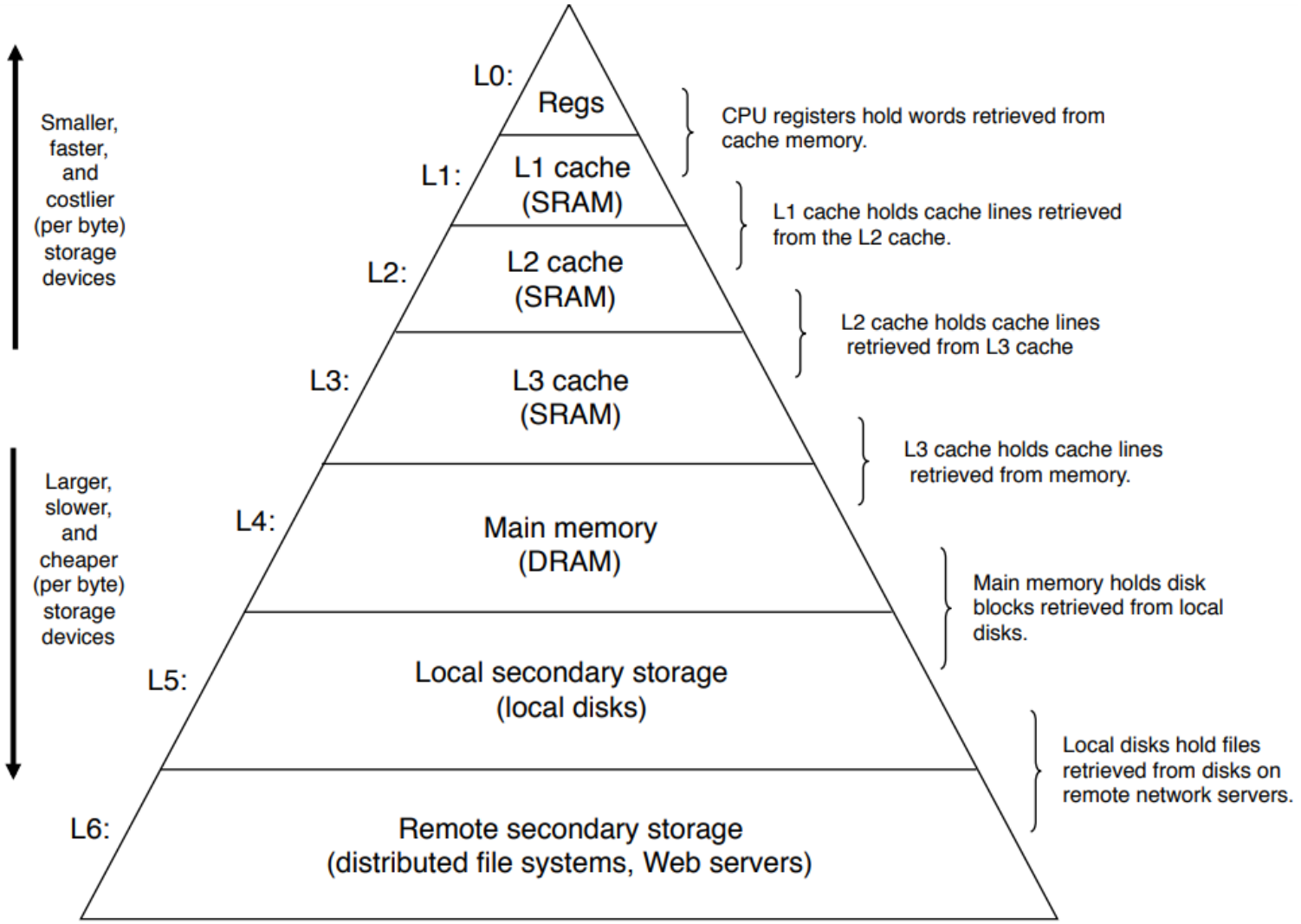
- Hardware




http://csapp.cs.cmu.edu/3e/ics3/intro/hardware.pdf

# Arquitetura de Computadores

## (hierarquia de memória)




http://csapp.cs.cmu.edu/3e/ics3/intro/memhier.pdf

# Arquitetura de Computadores

(*Latency Numbers Every Programmer Should Know - 2012*)

| | | | | | |
|---|---|---|---|---|---|
| L1 cache reference | 0.5 | ns | | | |
| Branch mispredict | 5 | ns | | | |
| L2 cache reference | 7 | ns | | | 14x L1 cache |
| Mutex lock/unlock | 25 | ns | | | |
| Main memory reference | 100 | ns | | | 20x L2 cache, 200x L1 cache |
| Compress 1K bytes with Zippy | 3,000 | ns | 3 us | | |
| Send 1K bytes over 1 Gbps network | 10,000 | ns | 10 us | | |
| Read 4K randomly from SSD* | 150,000 | ns | 150 us | | ~1GB/sec SSD |
| Read 1 MB sequentially from memory | 250,000 | ns | 250 us | | |
| Round trip within same datacenter | 500,000 | ns | 500 us | | |
| Read 1 MB sequentially from SSD* | 1,000,000 | ns | 1,000 us | 1 ms | ~1GB/sec SSD, 4X memory |
| Disk seek | 10,000,000 | ns | 10,000 us | 10 ms | 20x datacenter roundtrip |
| Read 1 MB sequentially from disk | 20,000,000 | ns | 20,000 us | 20 ms | 80x memory, 20X SSD |
| Send packet CA->Netherlands->CA | 150,000,000 | ns | 150,000 us | 150 ms | |

Notes

---

1 ns = 10^-9 seconds

1 us = 10^-6 seconds = 1,000 ns

1 ms = 10^-3 seconds = 1,000 us = 1,000,000 ns

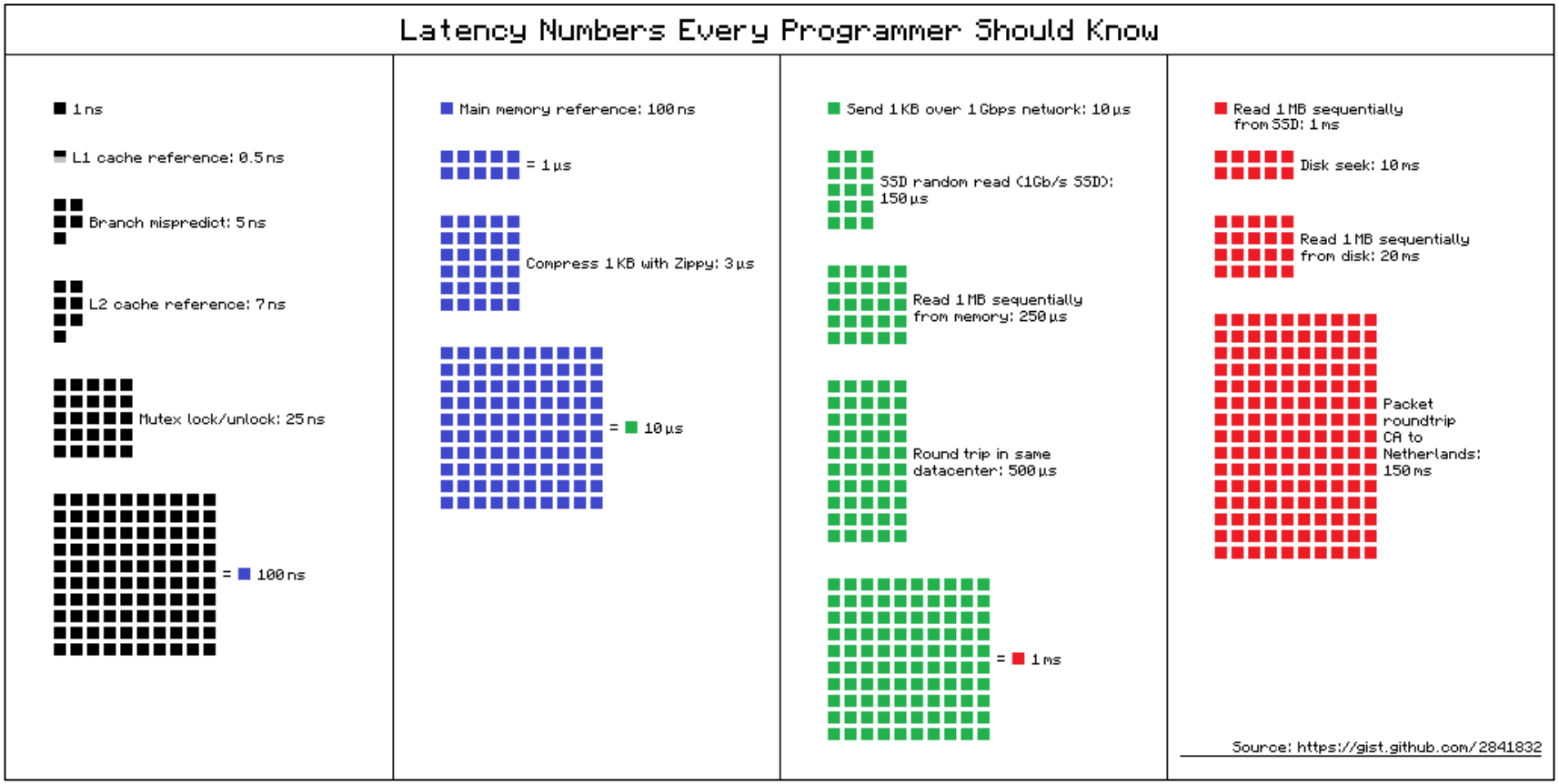
https://gist.github.com/jboner/2841832

http://norvig.com/21-days.html#answers

# Arquitetura de Computadores

(*Latency Numbers Every Programmer Should Know - 2012*)

- Representação Visual




https://gist.github.com/hellerbarde/2843375

https://colin-scott.github.io/personal_website/research/interactive_latency.html

# *Hashtables*

- *Dictionaries (Key + Value)*
  - `std::unordered_map`
- *Sets (Key)*
  - `std::unordered_set`
- Necessita de uma função *hash*
  - Mapeia um objeto para um inteiro positivo
    - Método para obter um índice através da chave
    - Por exemplo: `std::string` para `std::size_t`
  - Chaves diferentes podem gerar valores iguais
    - Colisão
    - Para minimizar as colisões, é necessário computar valores com distribuição uniforme
      - Usar números primos para compor a geração de um *hash*

# *Hash function*

(exemplo)

- `#include <functional>`
  - `std::unary_function` (*deprecated*)

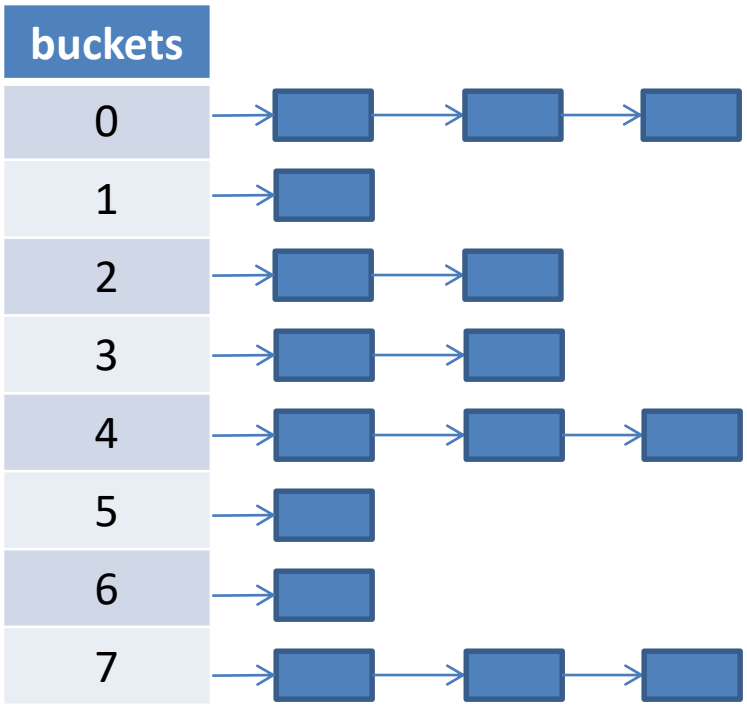
```
using coord2d = std::tuple<size_t, size_t>;

struct coord2d_hash : public std::unary_function<coord2d, size_t>
{
        std::size_t operator()(coord2d key) const
        {
                const int PRIME = 199;
                size_t h = std::get<0>(key) + PRIME * std::get<1>(key);
                return h;
        }
};

static std::unordered_map<coord2d, size_t, coord2d_hash> path_cost;
```

  - `std::hash`

```
std::hash<std::string> h;
std::size_t hash_of_hello_world = h("Hello World");
```

# *Separate Chain Hashtable*

(*unbounded*)




```
template <typename TKey>
class set
//separate_chain_hashtable::set (unbounded) interface
/*
void add(const TKey&);
bool exists(const TKey& key);
bool remove(const TKey&);
std::vector<TKey> keys() const;
*/
```

---

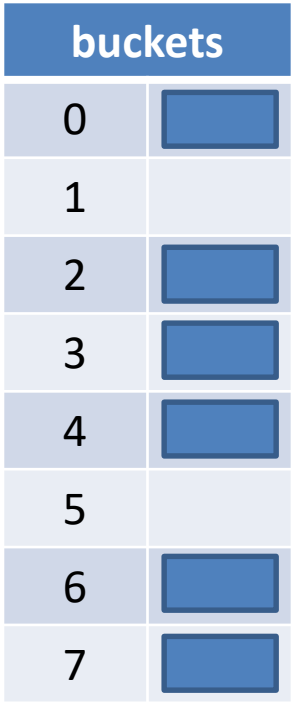
```
bucket_reference get_bucket(const TKey& key)
{
    std::size_t h = hasher_(key) % buckets_.size();
    return buckets_[h];
}
```

---

```
//separate_chain_hashtable::dictionary (unbounded) interface
/*
void add_or_update(const TKey&, const TValue&);
bool get(const TKey&, TValue&);
bool remove(const TKey&);
std::vector<TKey> keys() const;
bool empty() const;
*/

template <typename TKey, typename TValue>
class dictionary
```

# *Linear Probing Hashtable*

(*bounded, but resizable*)




```
//linear_probing_hashtable::dictionary (bounded, but resizable) interface
/*
void add_or_update(const TKey&, const TValue&);
bool get(const TKey&, TValue&);
bool remove(const TKey&);
std::vector<TKey> keys() const;
bool empty() const;
std::size_t size() const;
std::size_t capacity() const;
*/
```

```
std::size_t bucket_position(const TKey& key) const
{
    return hasher_(key) % buckets_.size();
}

std::size_t next_bucket_position(std::size_t pos) const
{
    return (pos + 1) % buckets_.size();
}
```

```
//linear_probing_hashtable::dictionary (bounded, but resizable) interface
/*
void add(const TKey&);
bool exists(const TKey&) const;
bool remove(const TKey&);
std::vector<TKey> keys() const;
bool empty() const;
std::size_t size() const;
std::size_t capacity() const;
*/
```

# *Linear Probing Hashtable*

(adicionar e remover)

- Adicionar
  - Em caso de colisão, deverá encontrar o próximo *bucket* livre de forma incremental e cíclica
- Remover
  - Deverá efetuar *rehashing* para cada elemento no caminho até chegar num *bucket* livre
- *Rehash*
  - Recalcular o *hash* do elemento no *bucket* e reposicioná-lo
- Redimensionar
  - Implicará em *Rehashing*

# *Bloom Filter*

- Problema do *Membership*
  - Se um elemento faz parte de um conjunto
- Estrutura de Dados Probabilística
  - Diversos *hashes* são aplicados ao elemento para distribuí-los num *array* de *bits* de tamanho fixo
    - Para buscar um elemento, basta aplicar a mesma sequência de *hashes* e verificar se os *bits* estão ligados no *array*
  - Redução drástica do espaço de armazenamento (memória), porém com resultado que pode ser um falso positivo (não gera falso negativo)
    - Tamanho constante de *bits*, poucos *bits* por elementos
- Falso positivo pode ser reduzido pelo tamanho do *array* de *bits*, quantidade e qualidade dos *hashes*

# *Bloom Filter*

(adicionar)

```
std::vector<bool> bits;
```

- Adicionar

```
bool add(const T& value)
{
    std::uint64_t h1, h2;
    std::tie(h1, h2) = Base::hashes_of(value);
    bool changed = false;
    int i = number_of_hash_functions;
    while (i--)
    {
        auto index = static_cast<std::size_t>((h1 & MAX_LONG) % bits.size());
        changed |= !bits[index];
        bits[index] = true;
        h1 += h2;
    }
    return changed;
}
```

h1 = murmurhash3_128 (64-bit high)
h2 = murmurhash3_128 (64-bit low)

# *Bloom Filter*

(testar)

```
std::vector<bool> bits;
```

- Testar

```
bool exists(const T& value) const
{
    std::uint64_t h1, h2;
    std::tie(h1, h2) = Base::hashes_of(value);
    int i = number_of_hash_functions;
    while (i--)
    {
        auto index = static_cast<std::size_t>((h1 & MAX_LONG) % bits.size());
        if (!bits[index])
            return false;
        h1 += h2;
    }
    return true;
}
```

h1 = murmurhash3_128 (64-bit high)
h2 = murmurhash3_128 (64-bit low)

# *Bloom Filter*

(exemplo)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*add("São Paulo") → a = 1, b = 3, c = 7*

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

*add("Rio de Janeiro") → a = 0, b = 2, c = 7*

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

*test("Salvador") → a = 0, b = 5, c = 6*

*False (OK)*

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

*test("Curitiba") → a = 1, b = 2, c = 7*

*True (Not OK – False Positive)*

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

*test("São Paulo") → a = 1, b = 3, c = 7*

*True (OK)*

# *Bloom Filter*

(probabilidade de falsos positivos e *bits*)

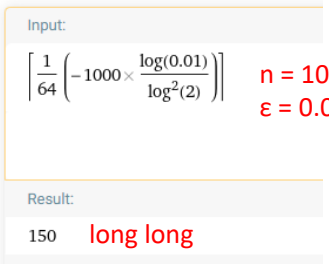
- Para obter o número ótimo de *bits* com uma determinada taxa de falsos positivos ($\varepsilon$) onde, ela pode variar entre $0.0 < \varepsilon < 1.0$, segue a fórmula:

$$m = -\frac{n \ln \varepsilon}{(\ln 2)^2}$$

$\varepsilon \rightarrow probabilidade\ de\ falsos\ positivos$

$n \rightarrow número\ de\ elementos\ inseridos$

$m \rightarrow número\ de\ bits\ requeridos$




exemplo: https://www.wolframalpha.com/input/?i=ceil%28%28-1000*ln%280.01%29%2F%28ln+2%29%5E2%29%2F64%29

https://en.wikipedia.org/wiki/Bloom_filter#Optimal_number_of_hash_functions

# *Bloom Filter*

(quantidade de funções *hash*)

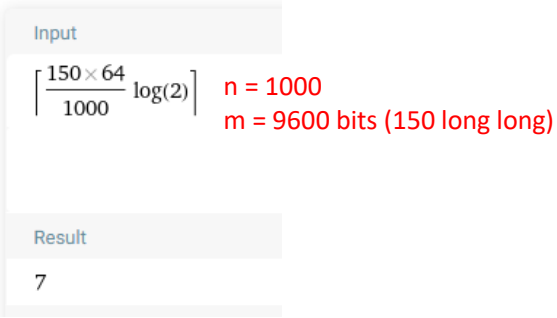
- Para obter o número ótimo de aplicações da função *hash* a partir do número ótimo de *bits* calculado anteriormente, segue a fórmula:

$$k = \frac{m}{n} \ln 2$$

$$: \quad m \rightarrow número\ de\ bits$$

$$n \rightarrow número\ de\ elementos\ inseridos$$

$$k \rightarrow número\ de\ funções\ hash\ requeridas$$



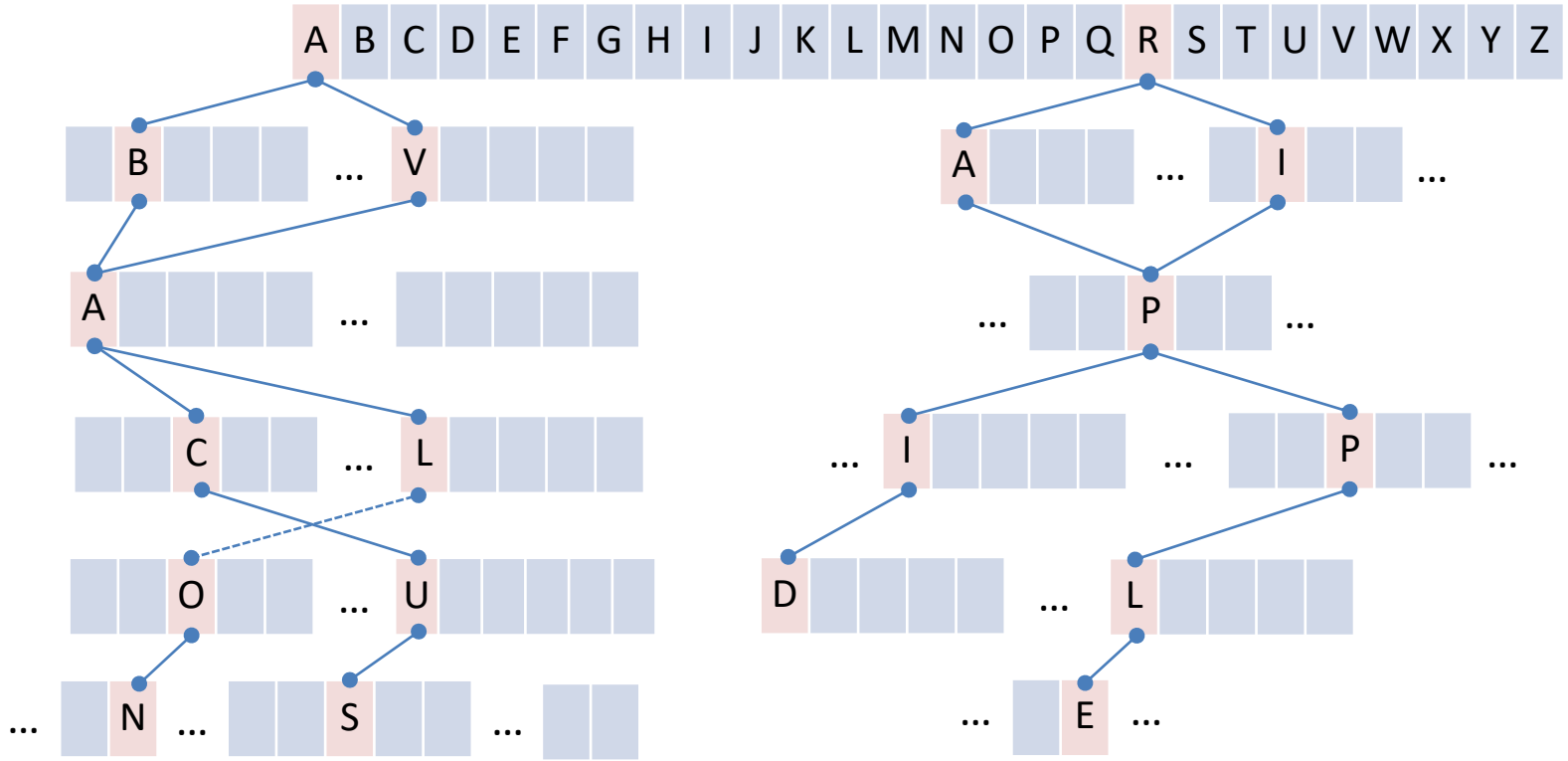

exemplo: https://www.wolframalpha.com/input?i=ceil%28%28%28150*64%29%2F1000%29%28ln+2%29%29

https://en.wikipedia.org/wiki/Bloom_filter#Optimal_number_of_hash_functions

# *Trie*

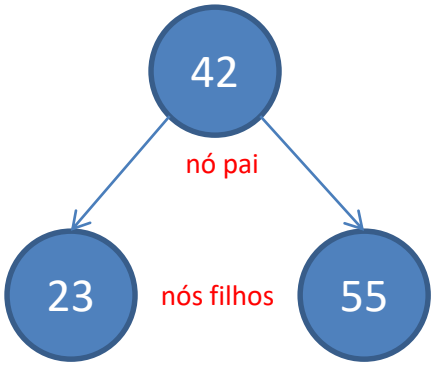
(*reTRIEval* ou *prefix tree*)

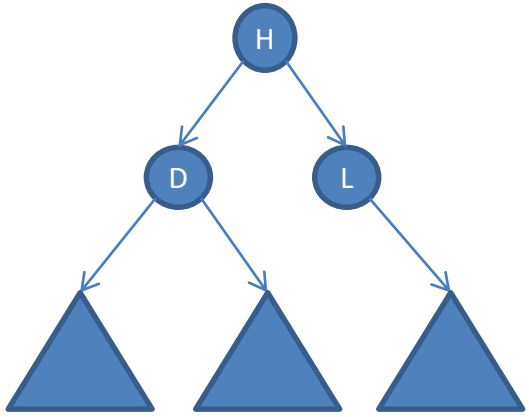
- Alternativa a *Hashtable* para chaves onde o tipo é *string*. Estrutura baseada em árvore
  - Suporte a busca por prefixo ou por padrão



# *Binary Search Tree* (*BST*)

- Cada nó da árvore possui um elemento comparável, onde:
  - O nó filho de menor valor está conectado a esquerda
  - O nó filho de maior (ou igual) valor está conectado a direita
- Uma BST pode estar: Balanceada ou Desbalanceada
- As sub-árvores a esquerda sempre estarão com os valores menores
- As sub-árvores a direita sempre estarão com os valores maiores



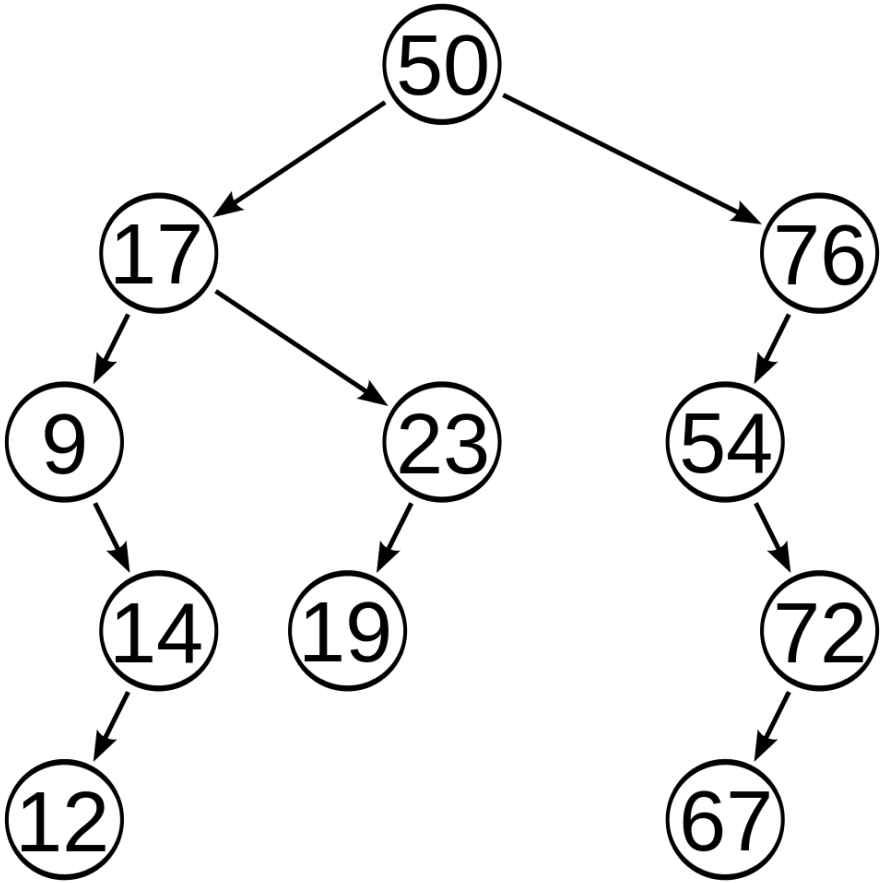




```
template <typename T>
struct bst_node
{
    T value;
    bst_node* left = nullptr;
    bst_node* right = nullptr;
};
```

# *Binary Search Tree* (*BST*)

(desbalanceada)

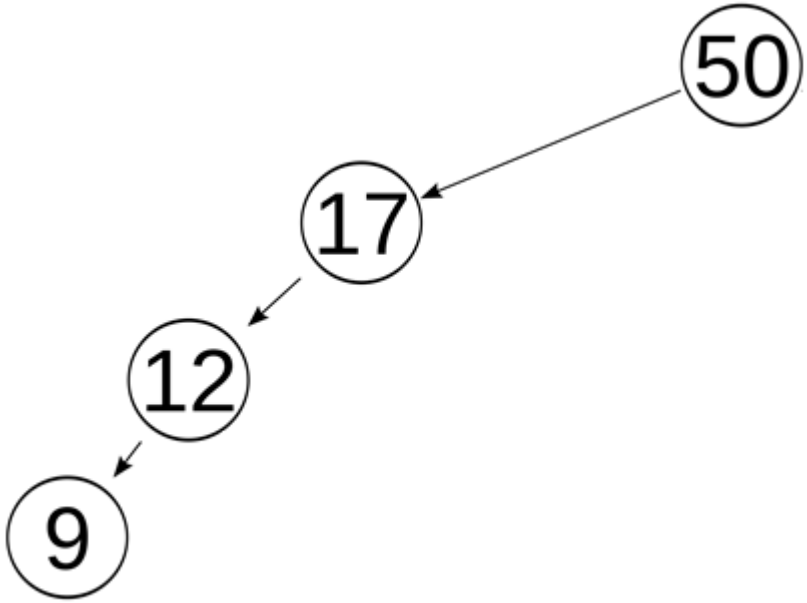
- Caso típico:



# *Binary Search Tree* (*BST*)

(desbalanceada)

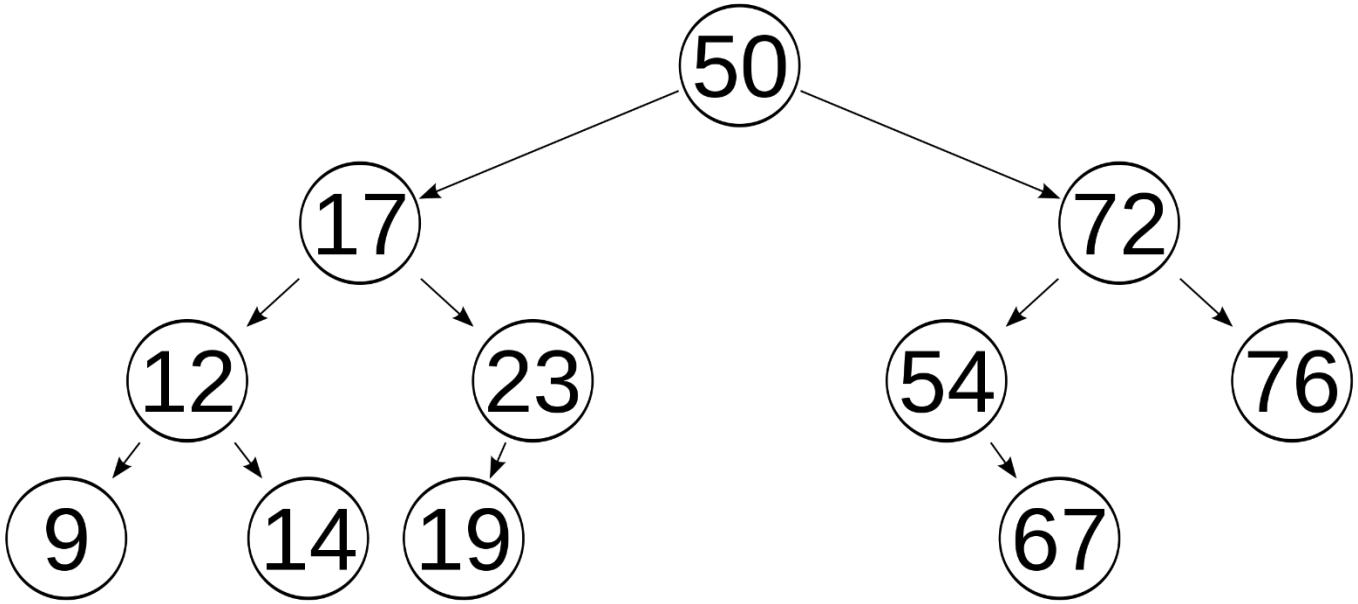
- Pior caso:
  - Busca linear



# *Binary Search Tree* (*BST*)

(balanceada)

- Melhor caso:
  - Busca logarítmica



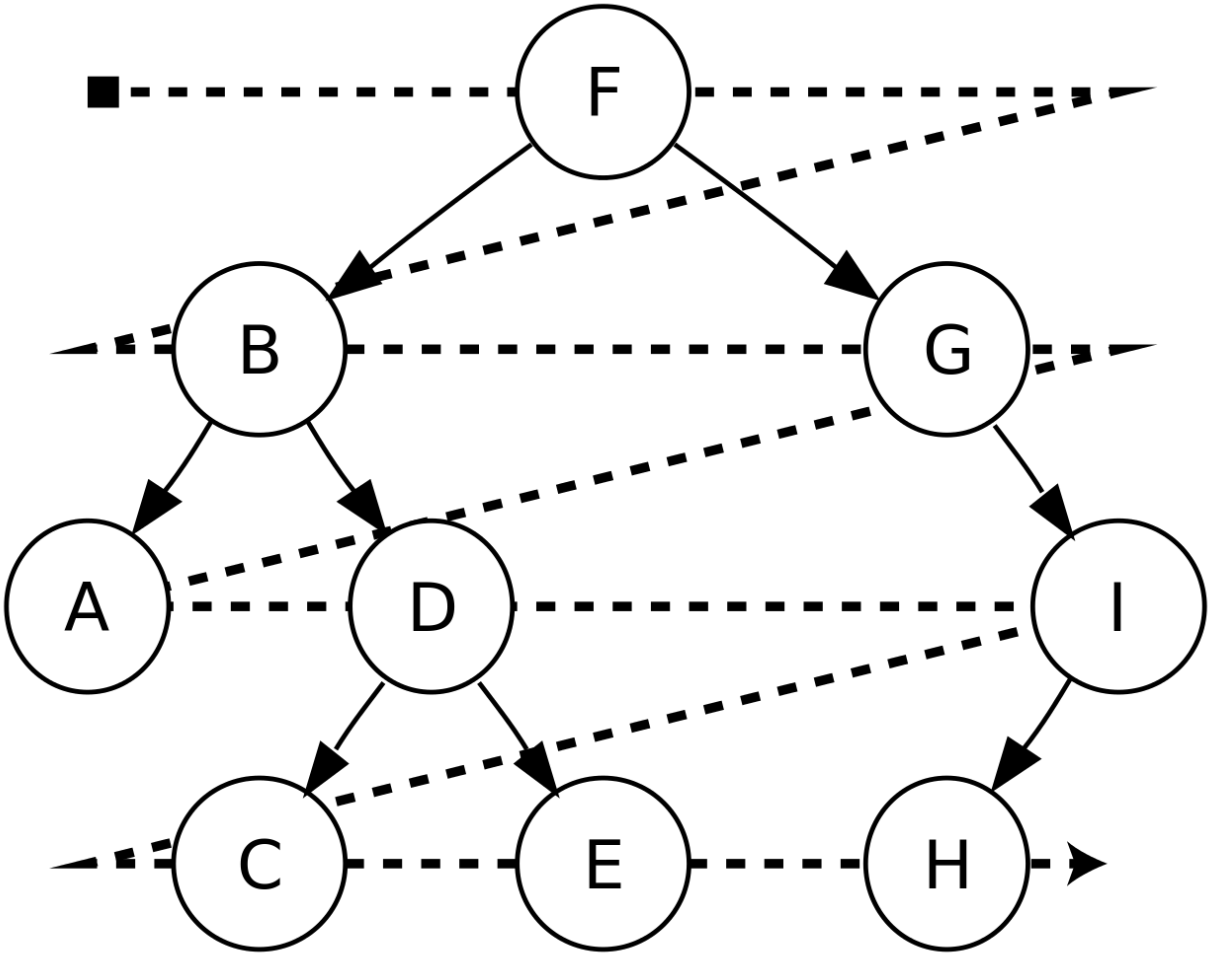
# *Binary Search Tree* (*BST*)

(*traversal*)

- Visitar os nós da árvore binária
  - *Breadth-first*
    - *level-order*
  - *Depth-first*
    - *pre-order*
    - *in-order*
      - *sorted order*
    - *out-order*
      - *reverse sorted order*
    - *post-order*

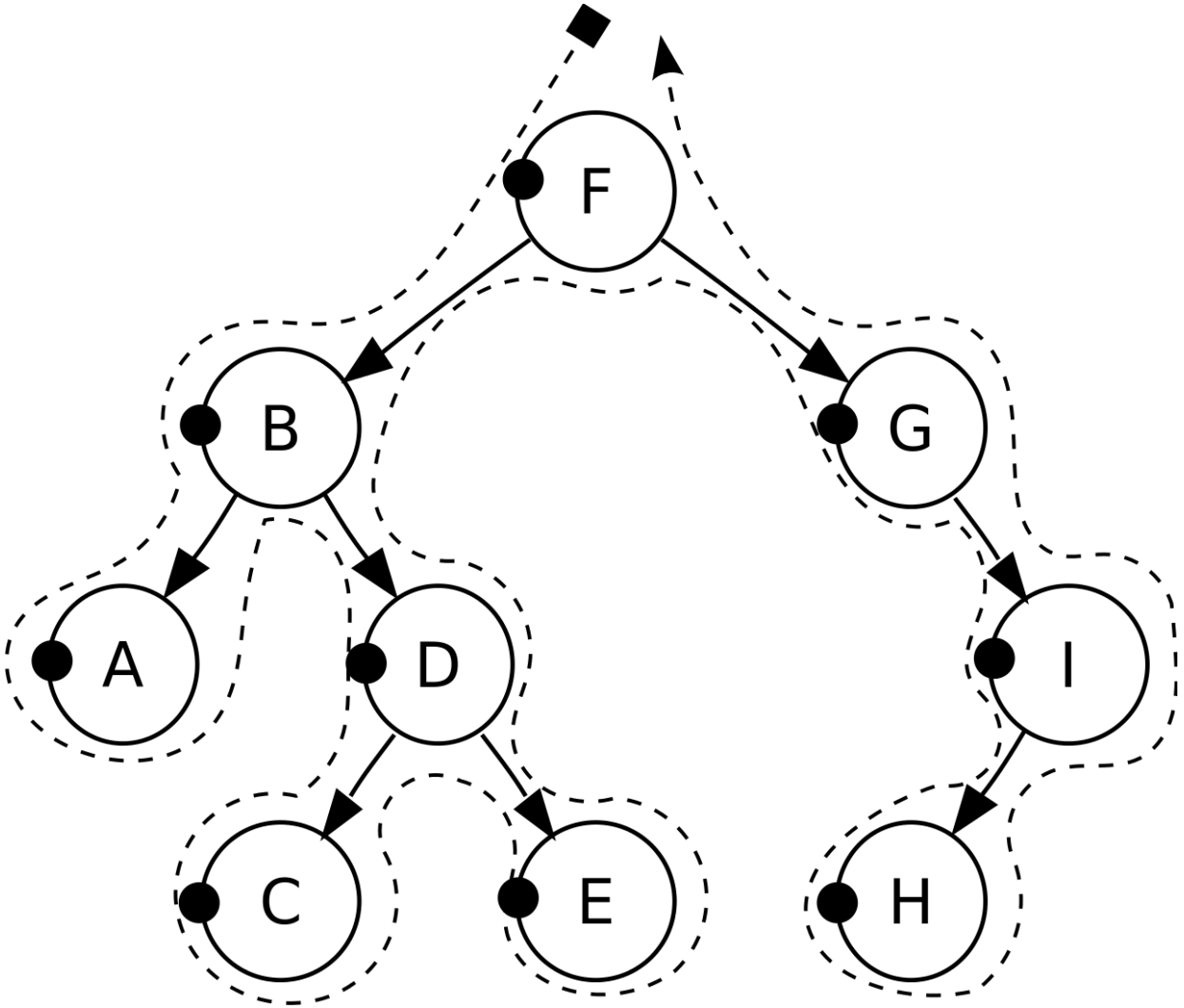
# *Binary Search Tree* (*BST*)

(*level-order traversal*)



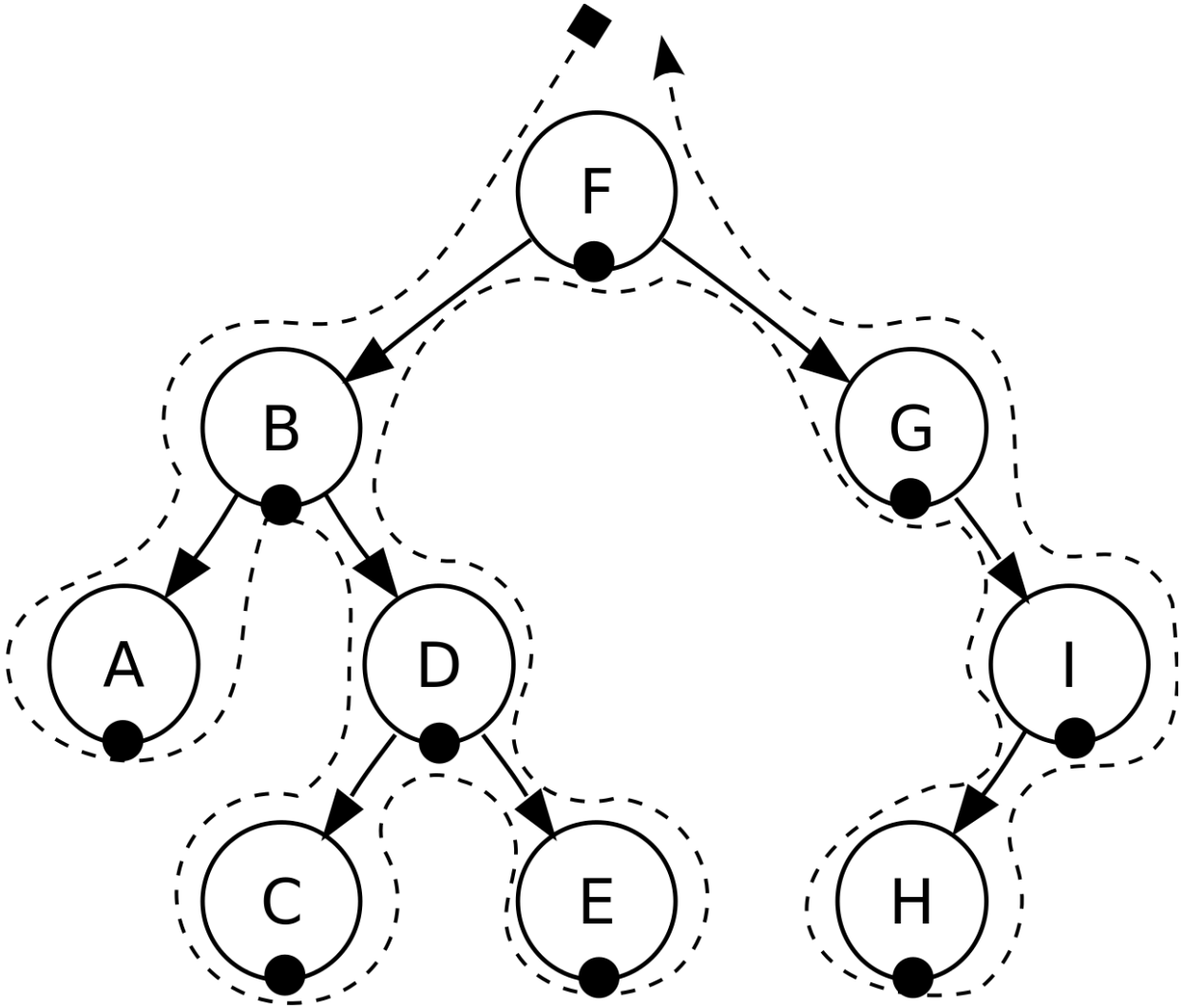
# *Binary Search Tree* (*BST*)

(*pre-order traversal*)



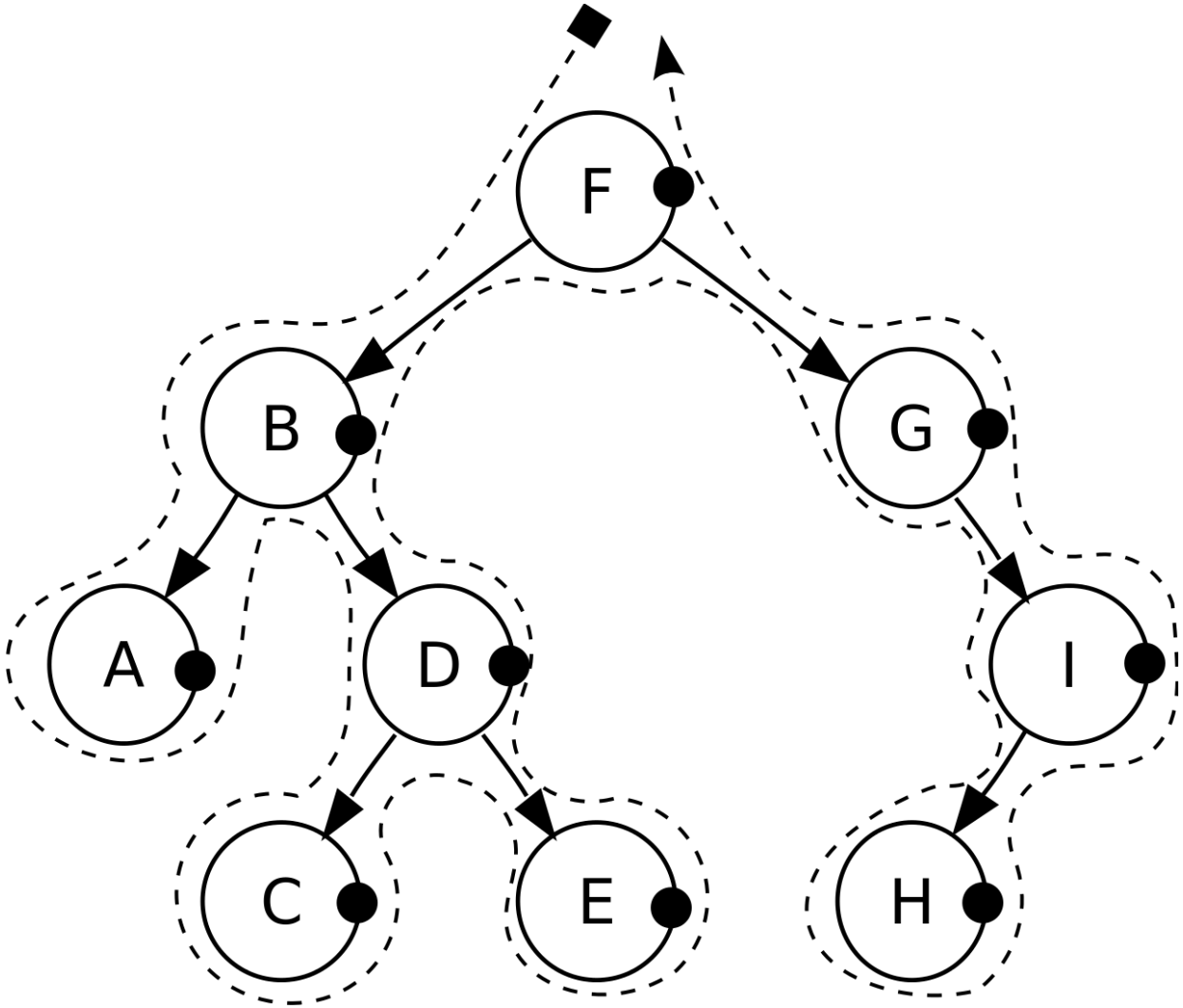
# *Binary Search Tree* (*BST*)

(*in-order traversal*)



# *Binary Search Tree* (*BST*)

(*post-order traversal*)



# *Iterative Traversal* (*BST*)

(*breadth-first*)

- Fila

```
template <typename T>
static bool level_order_traversal(const bst_node<T>* node, std::function<VisitStatus(const T&)> visit)
{
    if (node)
    {
        std::queue<bst_node<T>*> q;
        q.push(const_cast<bst_node<T>*>(node));
        while (!q.empty())
        {
            bst_node<T>* x = q.front();
            q.pop();
            if (visit(x->value) == VisitStatus::Stop)
                return true;
            if (x->left)  q.push(x->left);
            if (x->right) q.push(x->right);
        }
    }
    return false;
}
```

# *Iterative Traversal* (*BST*)

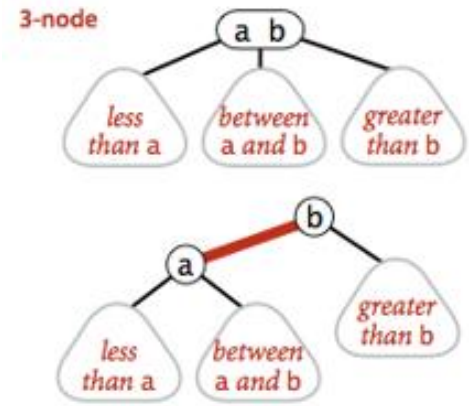
(*depth-first*)

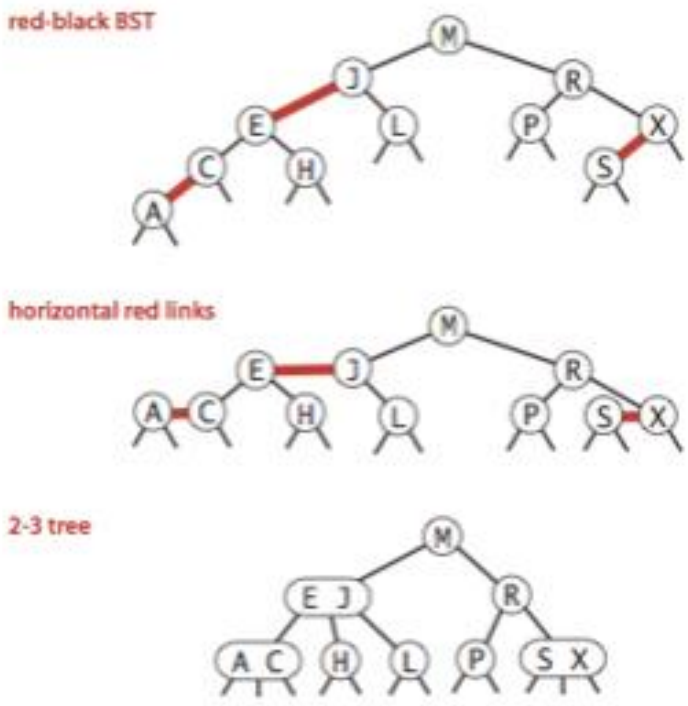
- Pilha

```
template <typename T>
bool pre_order_traversal_iterative(const bst_node<T>* node, std::function<VisitStatus(const T&)> visit)
{
    if (node)
    {
        std::stack<bst_node<T>*> s;
        s.push(const_cast<bst_node<T>*>(node));
        while (!s.empty())
        {
            bst_node<T>* x = s.top();
            s.pop();
            if (visit(x->value) == VisitStatus::Stop)
                return true;
            if (x->right) s.push(x->right);
            if (x->left)  s.push(x->left);
        }
    }
    return false;
}
```

# *Red-Black Tree* (*RB*)

- É uma *binary search tree* balanceada
  - Após inserção, 3 operações para balancear a árvore: *rotate left*, *rotate right*, *flip colors*



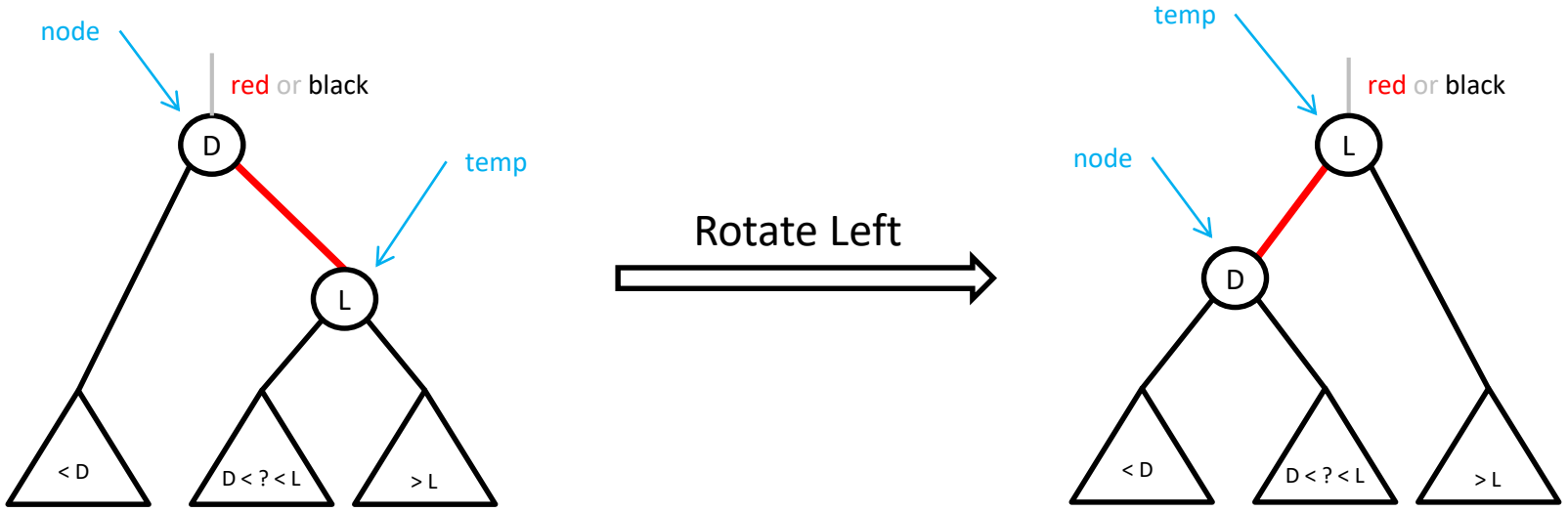

```
template <typename T>
struct rb_node
{
    T value;
    rb_node* left = nullptr;
    rb_node* right = nullptr;
    Color color = Color::Red;
    bool red() const
    {
        return color == Color::Red;
    }
};
```

Ref.: https://algs4.cs.princeton.edu/33balanced/

# *Red-Black Tree (RB)*

(*rotate left*)

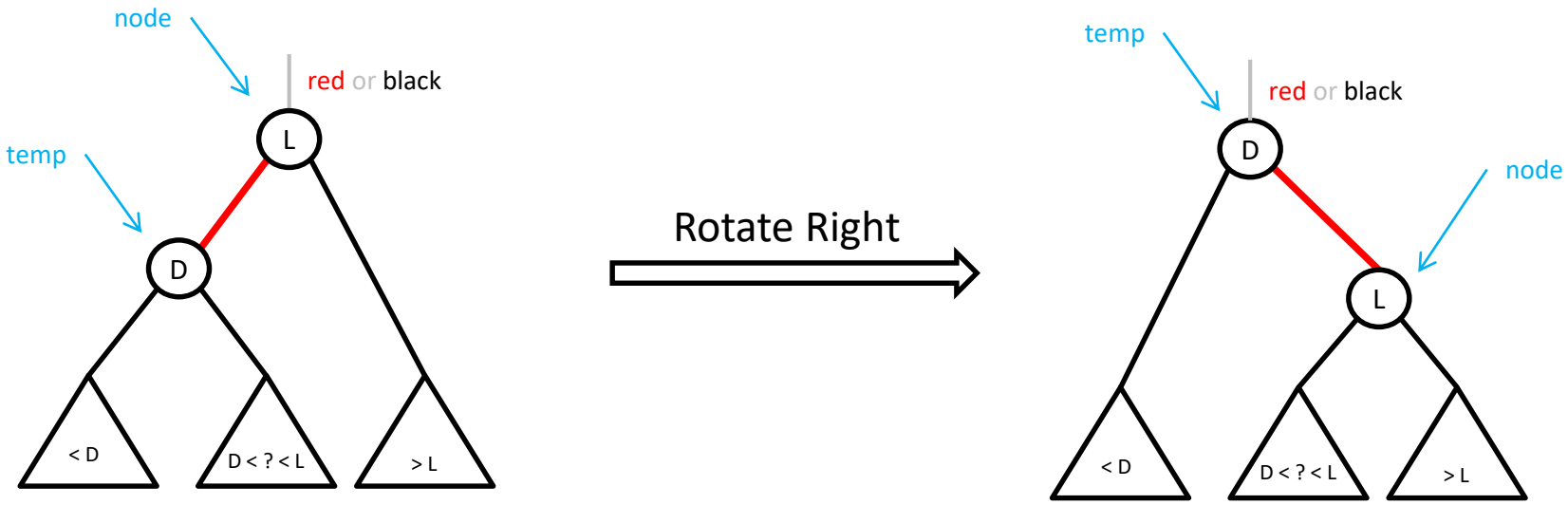
```
static rb_node<T>* rotate_left(rb_node<T>* node)
{
    rb_node<T>* temp = node->right;
    node->right = temp->left;
    temp->left = node;
    temp->color = node->color;
    node->color = Color::Red;
    return temp;
}
```



# *Red-Black Tree (RB)*

(*rotate right*)

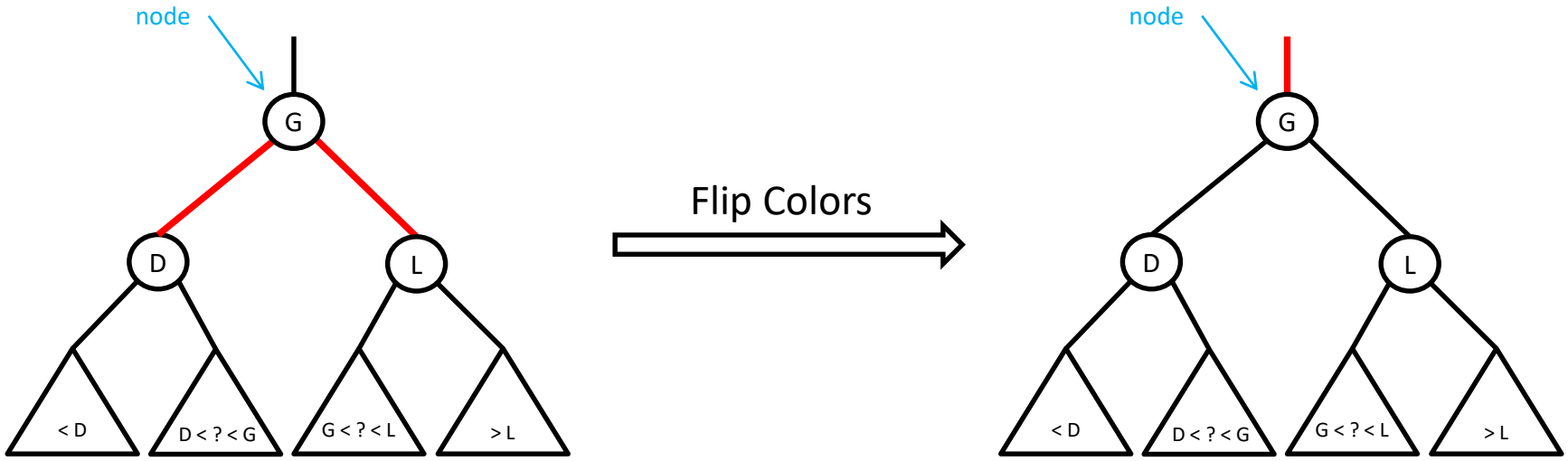
```
static rb_node<T>* rotate_right(rb_node<T>* node)
{
    rb_node<T>* temp = node->left;
    node->left = temp->right;
    temp->right = node;
    temp->color = node->color;
    node->color = Color::Red;
    return temp;
}
```



# *Red-Black Tree (RB)*

(*flip colors*)

```
static rb_node<T>* flip_colors(rb_node<T>* node)
{
    node->left->color = Color::Black;
    node->right->color = Color::Black;
    node->color = Color::Red;
    return node;
}
```



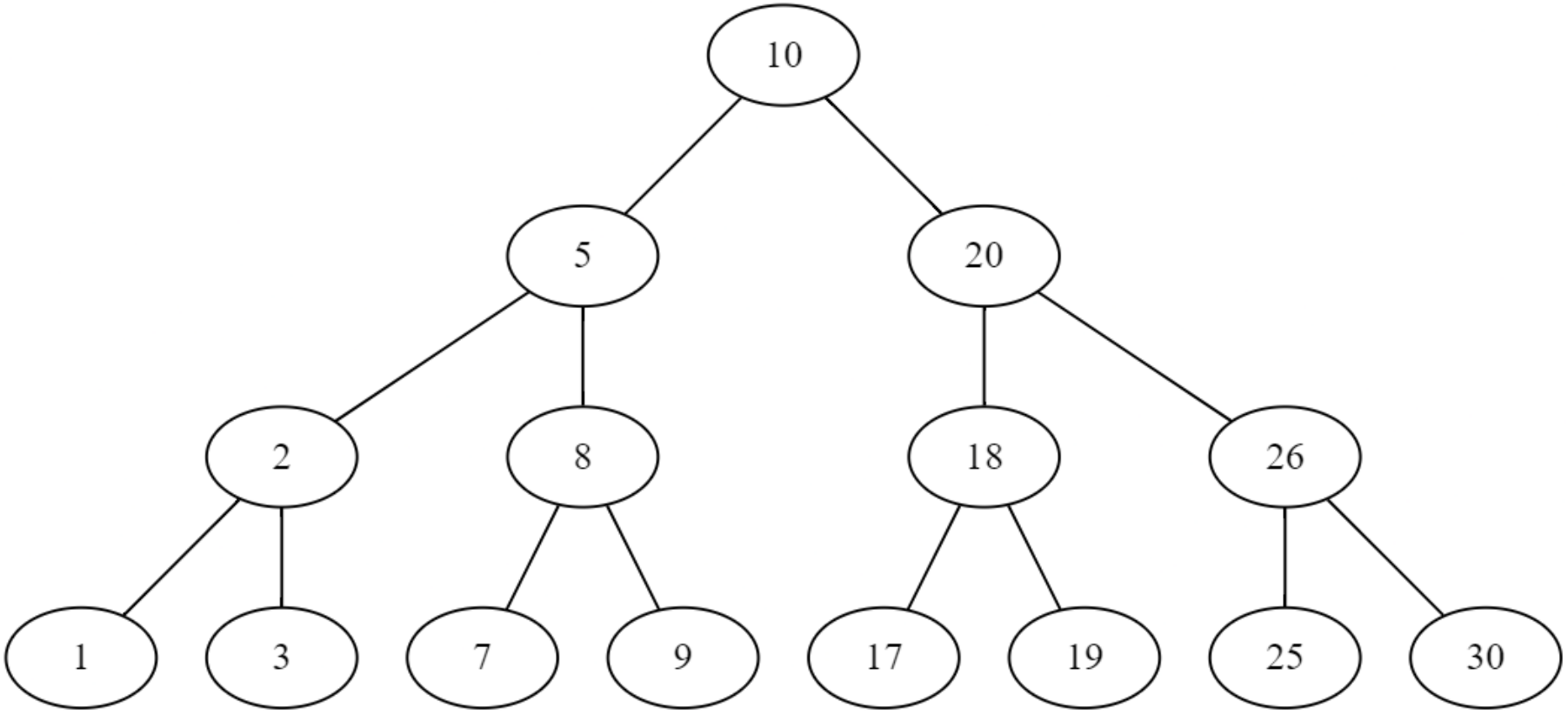
# *Red-Black Tree* (*RB*)

(ações do balanceamento ao inserir de 1 a 10)

```
1:
2: rotate left;
3: flip colors;
4: rotate left;
5: flip colors; rotate left;
6: rotate left;
7: flip colors; flip colors;
8: rotate left;
9: flip colors; rotate left;
10: rotate left;
4 *
 |_2
   |_1
   |_3
 |_8
   |_6 *
     |_5
     |_7
   |_10
     |_9 *

level-order: 4 2 8 1 3 6 10 5 7 9
```

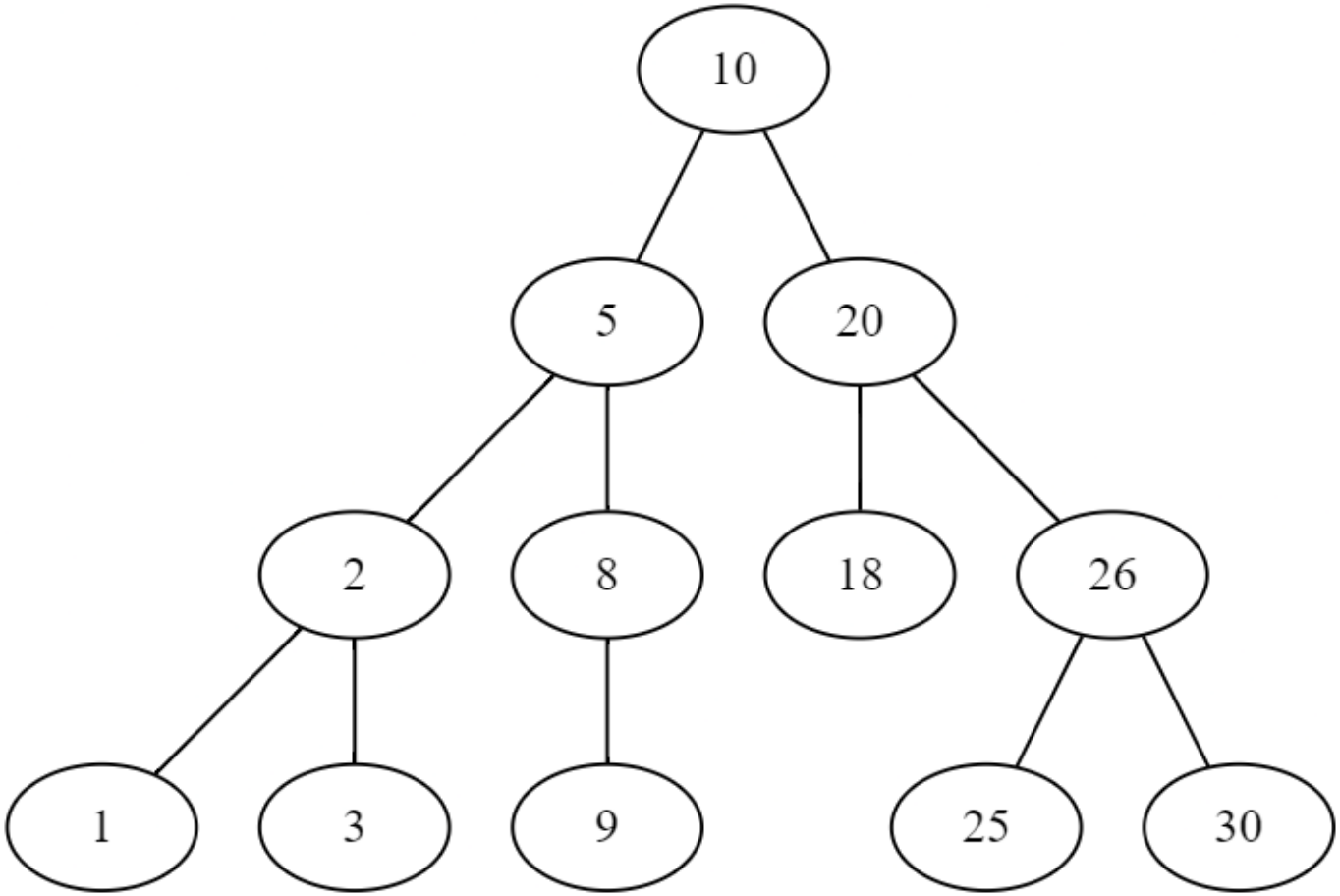
# Principais Tipos de *Binary Search Tree*

- Perfeita (ou Completa)



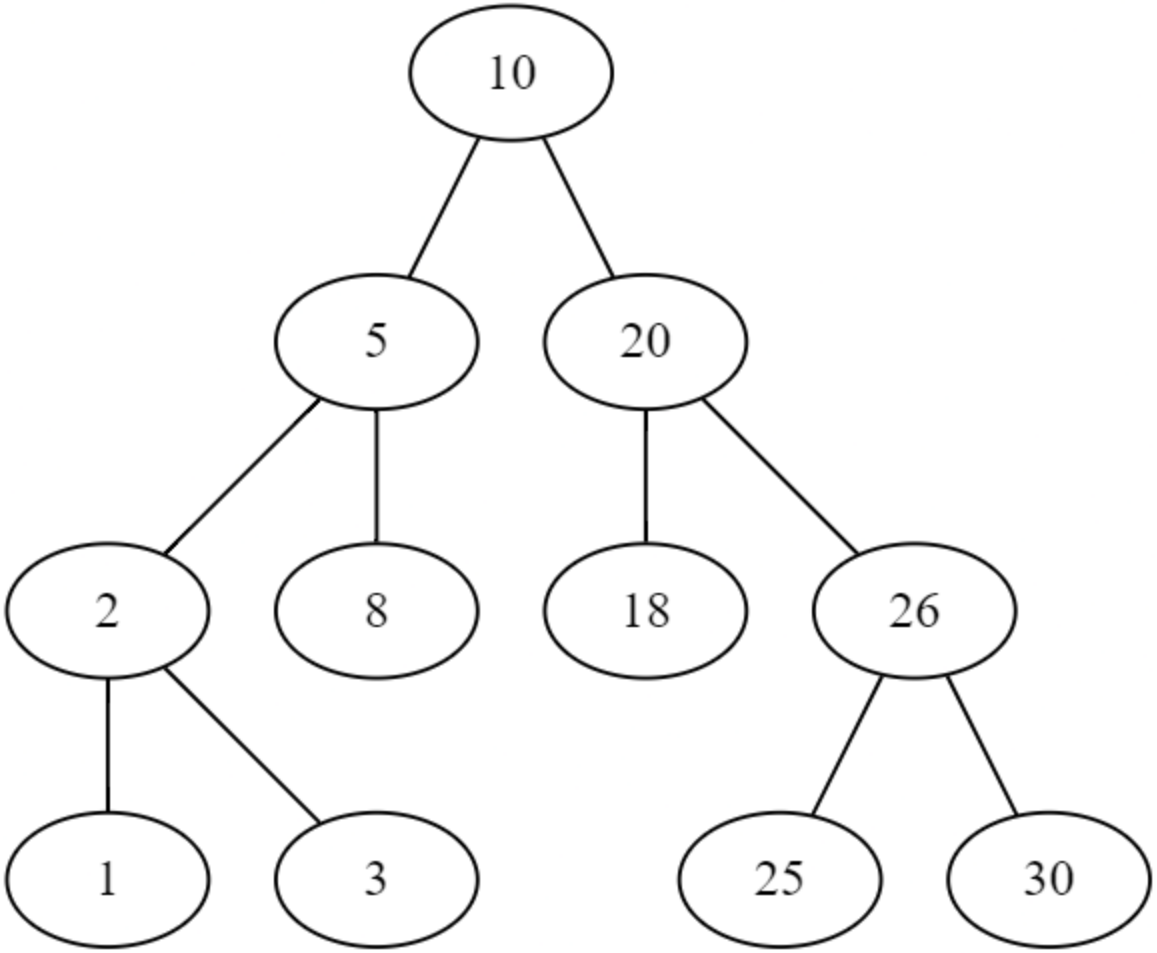
# Principais Tipos de *Binary Search Tree*

- Quase completa (ou Completa)



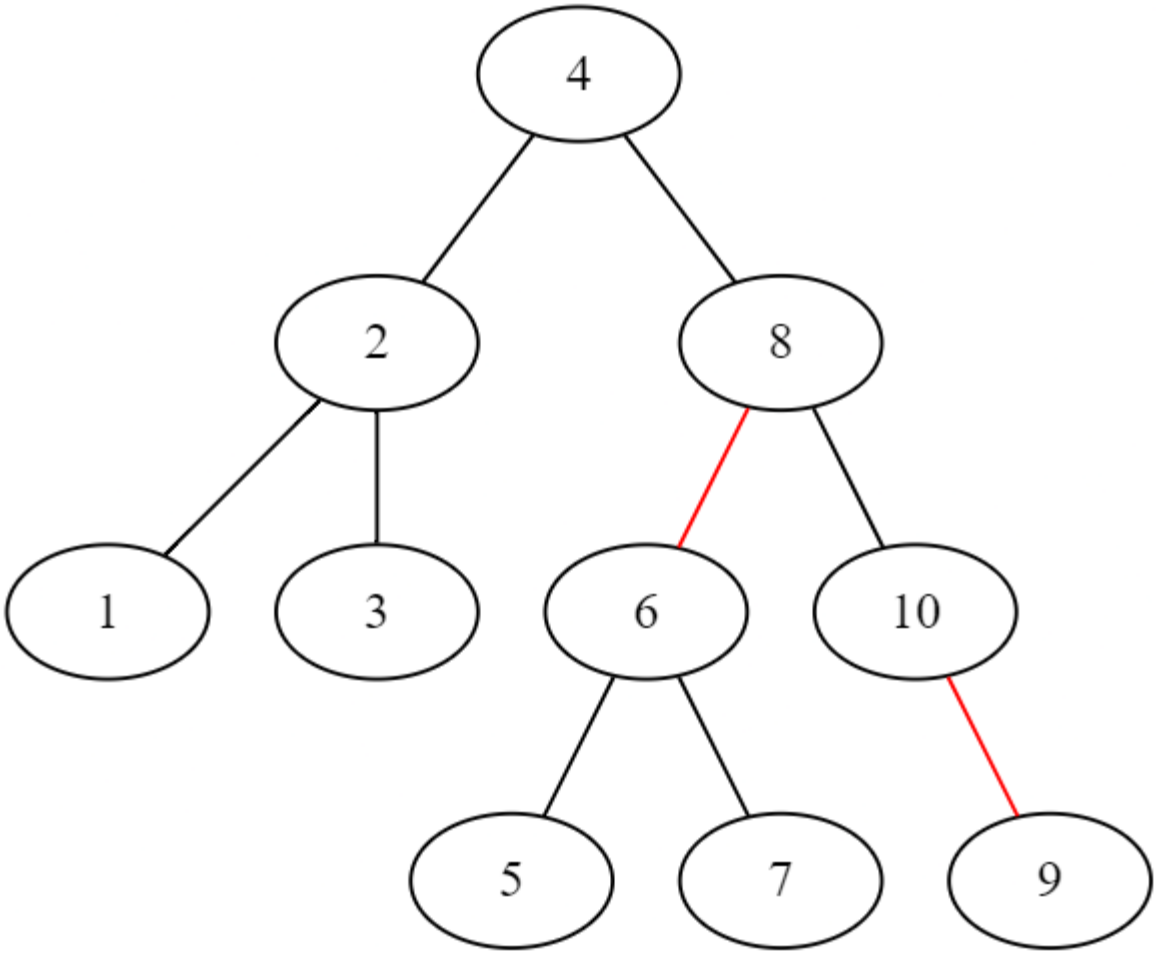
# Principais Tipos de *Binary Search Tree*

- Cheia



# Principais Tipos de *Binary Search Tree*

- Balanceada



# *Backtracking*

- Estratégia ou heurística de resolução de problema ao qual os candidatos da solução são escolhidos mediante a satisfação de um determinado critério
  - Problemas de satisfação de restrição
- Um candidato é abandonado quando fica claro que ele não poderá alcançar uma solução válida
- Parte do caminho percorrido pelo candidato será deixado para trás, então um novo rumo será tomado
  - *Backtrack*

# *Sudoku Problem*

- Candidatos a solução & *Backtracking*

$RGN_0 = \{1,3,4,7,8,9\}$

$COL_0 = \{1,2,3,4,6,8\}$

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

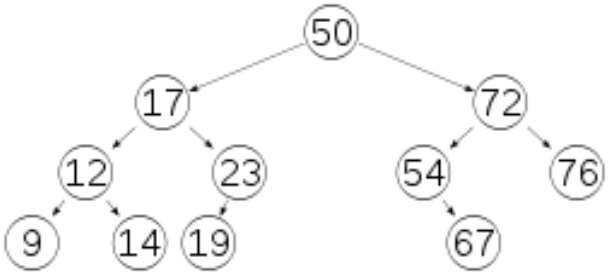
$ROW_0 = \{1,2,4,5,6,7,8,9\}$

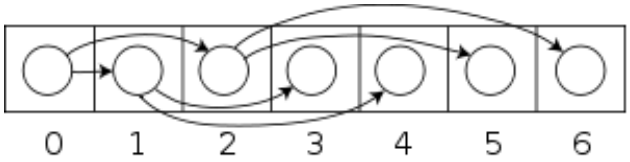
- Satisfaz:

$ROW_0 \cap COL_0 \cap RGN_0 = \{1,4,8\}$

- *Solver*:

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




# #include <algorithm>

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com

http://simplycpp.com/

http://github.com/fabiogaluppo

fabiogaluppo@acm.org

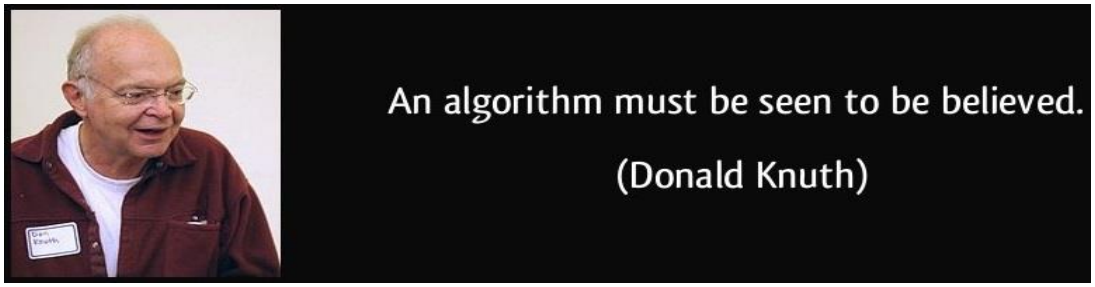
@FabioGaluppo

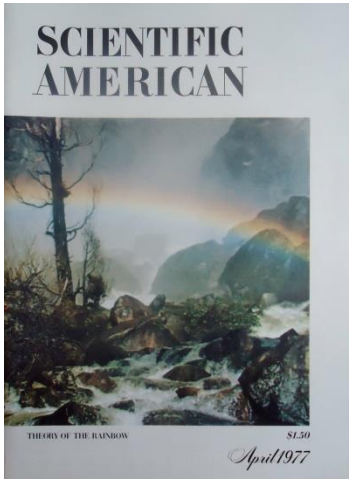
# O que é Algoritmo?

## Algorithms

*An algorithm is a set of rules for getting a specific output from a specific input. Each step must be so precisely defined it can be translated into computer language and executed by machine*

by Donald E. Knuth




https://www.scientificamerican.com/magazine/sa/1977/04-01/#article-algorithms

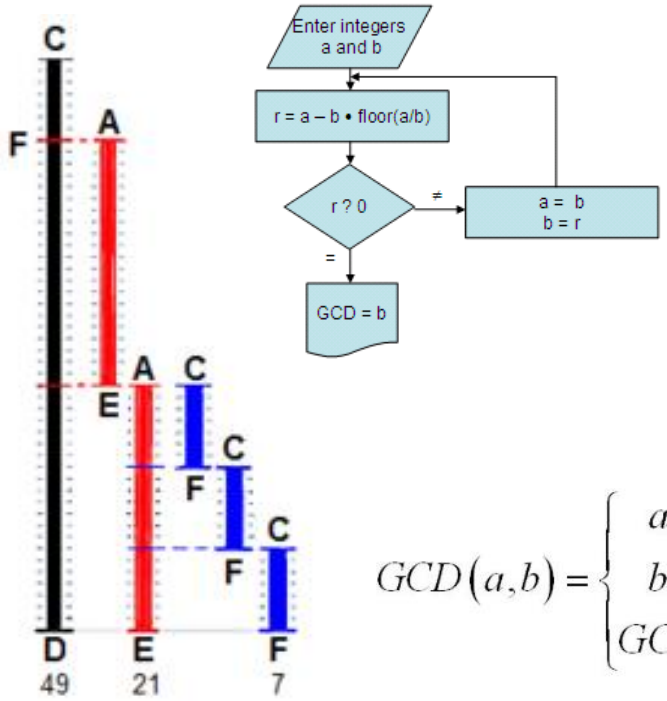
Algorithms

# WARM UP

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

# Algoritmos e Linguagens de Programação

- Algoritmo é um conceito mental que existe independemente de qualquer representação




$$7854 = 1 \cdot 4746 + 3108$$
$$4746 = 1 \cdot 3108 + 1638$$
$$3108 = 1 \cdot 1638 + 1470$$
$$1638 = 1 \cdot 1470 + 168$$
$$1470 = 8 \cdot 168 + 126$$
$$168 = 1 \cdot 126 + 42$$
$$126 = 3 \cdot 42 + 0.$$

$$GCD(a,b) = \begin{cases} a & \text{if } b=0 \\ b & \text{if } a=0 \\ GCD(b, a \bmod b) & \text{otherwise} \end{cases}$$

```
1: procedure EUCLID(a, b)
2:     r ← a mod b
3:     while r ≠ 0 do
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     end while
8:     return b
9: end procedure
```

https://en.wikipedia.org/wiki/Euclidean_algorithm

# Algoritmos e Linguagens de Programação

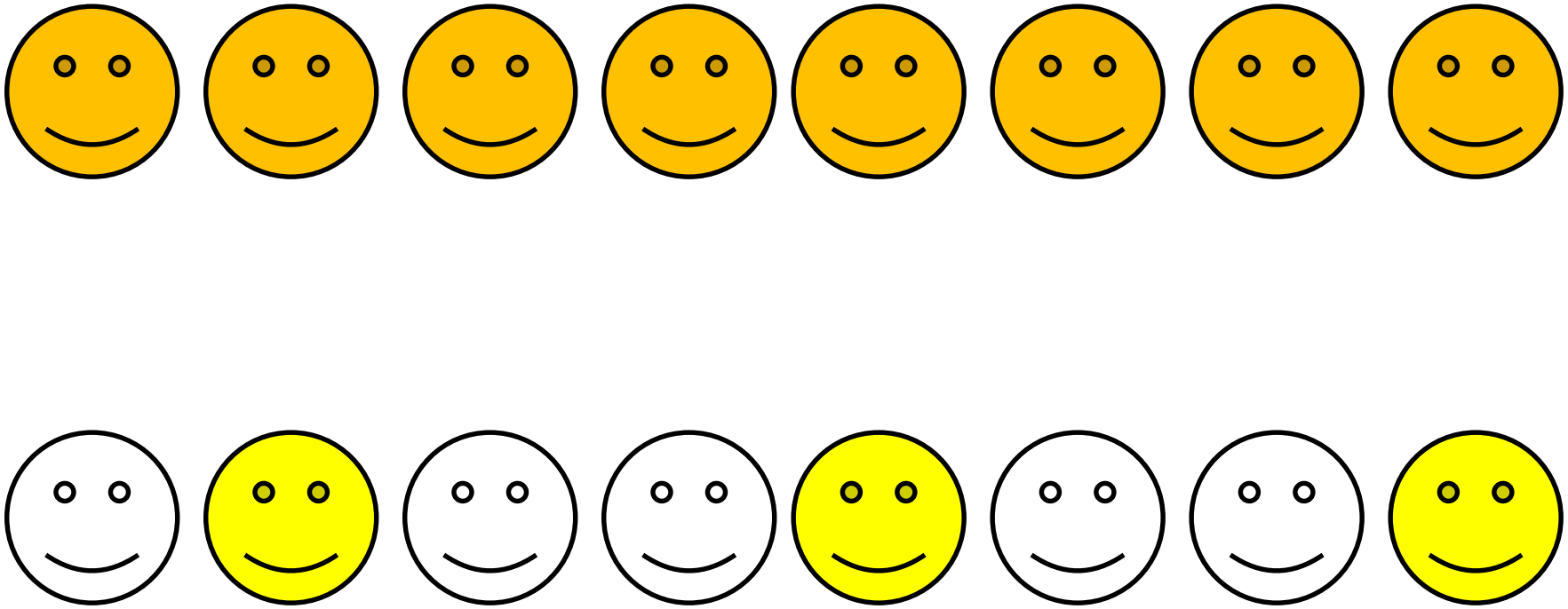
- Greatest Common Divisor (GCD) in C++:

```
template <typename T>
T gcd_iterative(T a, T b)
{
    T temp = a % b;
    while (!(temp == T(0)))
    {
        a = b;
        b = temp;
        temp = a % b;
    }
    return b;
}
```

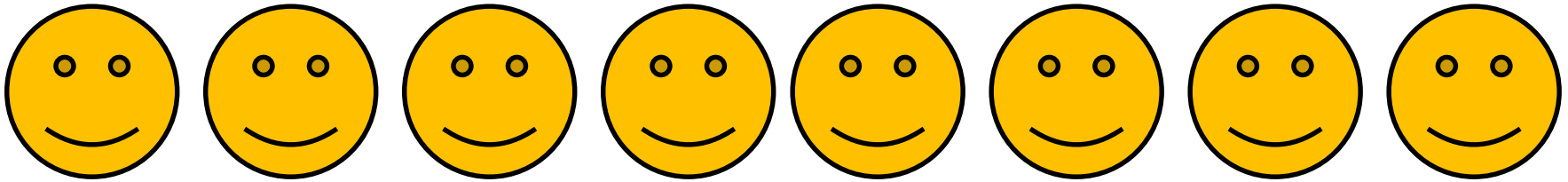
```
template <typename T>
T gcd_recursive(T a, T b)
{
    if (b == T(0)) return a;
    if (a == T(0)) return b;
    return gcd_recursive(b, a % b);
}
```

- Você utiliza um código para dizer ao computador o que ele deve fazer. No entanto, antes você precisa elaborar um algoritmo

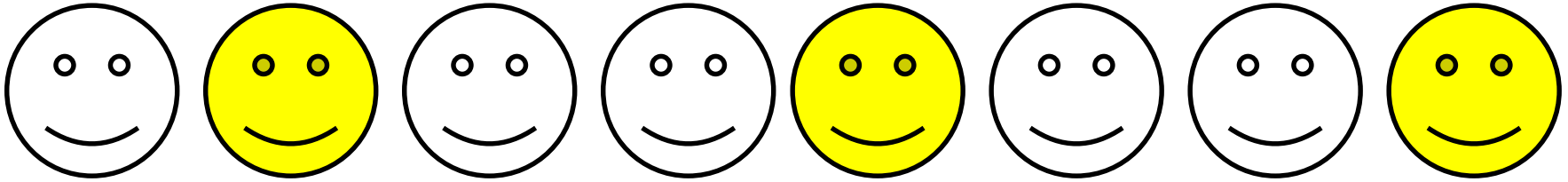
# Sobre esforço computacional



# Sobre esforço computacional



$$N = 8 \rightarrow Contagem = 8$$



$$N = 8 \rightarrow Contagem = 3$$

$$3 = log_2 8$$

# Análise de desempenho

- Assintótica – limite tendendo ao infinito

```
std::max_element(xs.begin(), xs.end());
```
$O(N)$

```
for (int i = 0; i < n - 1; ++i)
    for (int j = 0; j < n - i - 1; ++j)
        if (xs[j] > xs[j + 1])
            std::swap(xs[j], xs[j + 1]);
```
$O(N^2)$

- Empírica – experimentos ou observações

```
Estimated runnning time is 2.4438e-06 x N^1.0000 ms
```

```
Estimated runnning time is 1.9701e-07 x N^2.2192 ms
```

# Algoritmo e Memória

(versão imperativa)

- Contar a incidência de um valor em uma estrutura de dados

```
size_t count_occurrences(const std::vector<int>& xs, int target)
{
    size_t counter = 0;
    for (size_t i = 0; i < xs.size(); ++i)
        if (xs[i] == target)
            ++counter;
    return counter;
}
```

index-based

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 3 | 1 | 4 | 5 | 1 | 2 | 5 | 2 |

# Algoritmo e Memória

(versão imperativa)

- Contar a incidência de um valor em uma estrutura de dados

pointer-based

```
size_t count_occurrences(node* ptr, int target)
{
    size_t counter = 0;
    while (ptr != nullptr)
    {
        if (ptr->value == target)
            ++counter;
        ptr = ptr->next;
    }
    return counter;
}
```

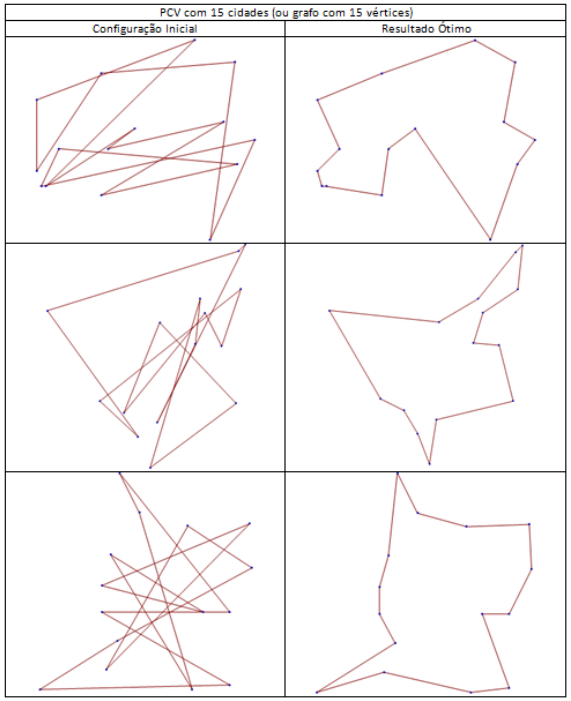
```
struct node
{
    node(int value, node* next = nullptr) :
        value(value), next(next) {}
    int value;
    node* next;
};
```

# Sobre eficiência e utilidade

- Um algoritmo é plenamente útil se e somente se for eficiente!

| | Tempo (ms) |
|---|---|
| Número de Cidades | Força Bruta |
| 13 | 743691 |
| 12 | 53093 |
| 11 | 4056 |
| 10 | 331 |
| 9 | 39 |
| 8 | 2 |
| 7 | 1 |

| PCV com 15 Cidades | | |
|---|---|---|
| | Força Bruta | |
| Solução Ótima | Tempo (ms) | Tempo (h) |
| 359,399 | 165340592 | 45,928 |
| 317,232 | 165590540 | 45,997 |
| 368,79 | 165517424 | 45,977 |




15!

Result:

1 307 674 368 000

WolframAlpha computational intelligence.

Number name:

1 trillion 307 billion 674 million 368 thousand

$47!/2 = 1{,}29311 \times 10^{59}$ $47! = 2{,}58623 \times 10^{59}$ $48! = 1{,}24139 \times 10^{60}$

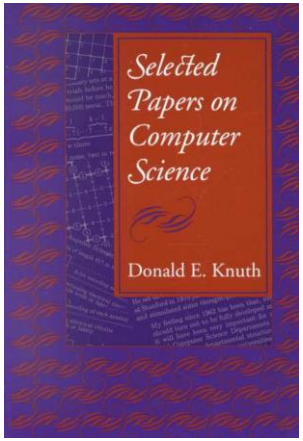
Algorithms

# SHOW ME THE C++ CODE

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

# Algorithms

```
1: Searching a Computer's Memory
2: The Advantage of Order
3: Binary Tree Search
4: Hashing (linear probing)
5: Improving Unsuccessful Searches
```




**I find that I don't understand things unless I try to program them.**

**Donald E. Knuth**
Professor Emeritus at Stanford University

https://www-cs-faculty.stanford.edu/~knuth/cs.html

# Searching a Computer's Memory

```
static inline std::size_t sequencial_search
(const std::vector<word_count>& words, const std::string& word)
{
    std::size_t N = words.size();
    while (N)
    {
        if (words[--N].word == word)
            return N;
    }
    return NOT_FOUND;
}
```

---

```
static inline const std::vector<word_count>& words()
{
    //monotonically decreasing order by count
    static std::vector<word_count> _words
    {
        { "THE", 15568 }, { "OF", 9767 },   { "AND", 7638 },   { "TO", 5739 },
        { "A", 5074 },    { "IN", 4312 },   { "THAT", 3017 },  { "IS", 2509 },
        { "I", 2292 },    { "IT", 2255 },   { "FOR", 1869 },   { "AS", 1853 },
        { "WITH", 1849 }, { "WAS", 1761 },  { "HIS", 1732 },   { "HE", 1727 },
        { "BE", 1535 },   { "NOT", 1469 },  { "BY", 1392 },    { "BUT", 1379 },
        { "HAVE", 1344 }, { "YOU", 1336 },  { "WHICH", 1291 }, { "ARE", 1222 },
        { "ON", 1155 },   { "OR", 1101 },   { "HER", 1093 },   { "HAD", 1062 },
        { "AT", 1053 },   { "FROM", 1039 }, { "THIS", 1021 }
    };
    return _words;
}
```

# The Advantage of Order

```
static inline std::size_t binary_search
(const std::vector<word_count>& words, const std::string& word)
{
    std::size_t l = 0, r = words.size(), mid;
    while (l != r)
    {
        mid = (r + l) / 2;
        int comp = word.compare(words[mid].word);
        if (comp == 0)
            return mid;
        else if (comp < 0)
            r = mid;
        else /* if (comp > 0) */
            l = mid + 1;
    }
    return NOT_FOUND;
}
```

# The Advantage of Order
# Achilles' hell of Binary Search

```
mid = (r + l) / 2;
```

```
79      std::size_t l_ = 0;
80      std::size_t r_ = std::numeric_limits<std::size_t>::max(); //4294967295
81  overflow_case:
82      std::size_t mid_ = (r_ + l_) / 2; //4294967295 + 2147483648 = 6442450943 / 2 = 3221225471
83      l_ = mid_ + 1; //2147483648 ≤ 1ms elapsed
84      goto overflow_case;
```

| Name | Value | Type |
|---|---|---|
| l | 0 | unsi |
| l_ | 2147483648 | unsi |
| mid | 3435973836 | unsi |
| mid_ | 1073741823 | unsi |
| r | | |
| r_ | | |
| word | | |
| words | | |

Calculator
Scientific
1,073,741,823

Calculator
Scientific
3,221,225,471.5

F-E
M+ M- MS M˅

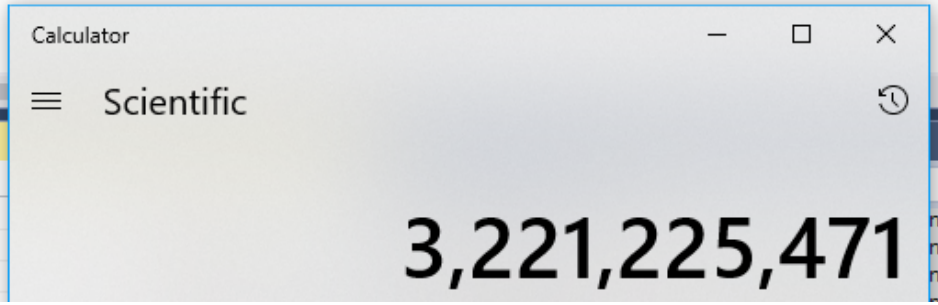
# The Advantage of Order Fixing Binary Search

```
mid = l + (r - l) / 2;
```

```
79      /*
80      >>> from math import floor
81      >>> def lerp(begin, end, percent):
82      ...     return begin + floor(percent * (end - begin))
83      ...
84      >>> lerp(2147483648, 4294967295, 0.5)
85      3221225471
86      */
87      std::size_t l_ = 0;
88      std::size_t r_ = std::numeric_limits<std::size_t>::max(); //4294967295
89  no_overflow_case:
90      std::size_t mid_ = l_ + (r_ - l_) / 2; //lerp(2147483648, 4294967295, 50%) = 3221225471
91      l_ = mid_ + 1; //2147483648 ≤ 1ms elapsed
92      goto no_overflow_case;
93
```

ls

| me | Value |
|---|---|
| l_ | 2147483648 |
| mid | 3435973836 |
| mid_ | 3221225471 |

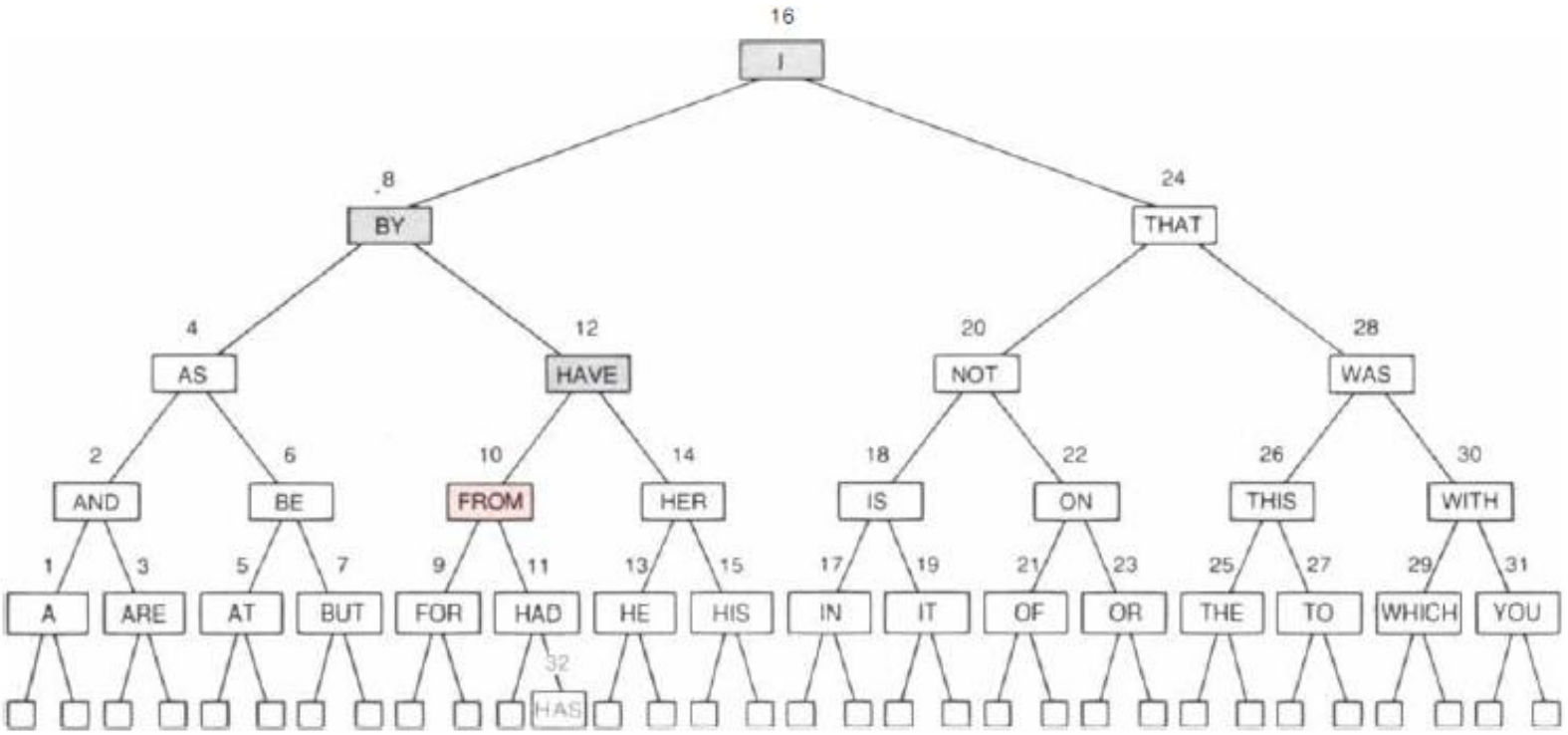


# The Advantage of Order

```
static inline std::size_t binary_search
(const std::vector<word_count>& words, const std::string& word)
{
    std::size_t l = 0, r = words.size(), mid;
    while (l != r)
    {
        mid = l + (r - l) / 2;
        int comp = word.compare(words[mid].word);
        if (comp == 0)
            return mid;
        else if (comp < 0)
            r = mid;
        else /* if (comp > 0) */
            l = mid + 1;
    }
    return NOT_FOUND;
}
```

# Binary Tree Search

```
struct bst_node final
{
    bst_node(size_t index) : index(index) {}
    std::size_t index = NOT_FOUND;
    std::unique_ptr<bst_node> left = nullptr;
    std::unique_ptr<bst_node> right = nullptr;
};
```




https://www.scientificamerican.com/magazine/sa/1977/04-01/#article-algorithms

# Binary Tree Search

```
static std::size_t bst_search_recursive
(const bst_node* ptr, const std::vector<word_count>& words, const std::string& word)
{
    if (ptr == nullptr)
        return NOT_FOUND;
    std::size_t index = ptr->index;
    int comp = word.compare(words[index].word);
    if (comp == 0)
        return index;
    if (comp < 0)
        return bst_search_recursive(ptr->left.get(), words, word);
    return bst_search_recursive(ptr->right.get(), words, word);
}
```

# Hashing

```
0:[THE,0]
1:[HAVE,3]
2:[TO,2]
3:[HIS,3]
4:[]
5:[BE,6]
6:[FOR,6]
7:[THIS,23]
8:[I,8]
9:[BUT,10]
10:[WAS,10]
11:[HAD,12]
12:[HE,12]
13:[FROM,19]
14:[AT,20]
15:[NOT,16]
```

```cpp
static inline std::size_t hash
(const std::string& word, std::size_t bucket_size)
{
    std::size_t h = 0;
    for (char ch : word)
        h += (ch - 'A' + 1);
    return --h % bucket_size;
}
```

```
16:[THAT,16]
17:[WHICH,18]
18:[AND,18]
19:[AS,19]
20:[OF,20]
21:[ON,28]
22:[IN,22]
23:[ARE,23]
24:[YOU,28]
25:[BY,26]
26:[WITH,27]
27:[IS,27]
28:[IT,28]
29:[HER,30]
30:[OR,0]
31:[A,0]
```

# Hashing

```
0:[THE,0]
1:[HAVE,3]
2:[TO,2]
3:[HIS,3]
4:[]
5:[BE,6]
6:[FOR,6]
7:[THIS,23]
8:[I,8]
9:[BUT,10]
10:[WAS,10]
11:[HAD,12]
12:[HE,12]
13:[FROM,19]
14:[AT,20]
15:[NOT,16]
```

```
static inline std::size_t hash_table_search
(const std::vector<word_count>& dic, const std::string& word)
{
    std::size_t bucket_size = dic.size();
    std::size_t h = hash(word, bucket_size);
    while (!dic[h].word.empty())
    {
        if (word == dic[h].word)
            return h;
        if (--h == NOT_FOUND)
            h = bucket_size - 1;
    }
    return NOT_FOUND;
}
```

```
16:[THAT,16]
17:[WHICH,18]
18:[AND,18]
19:[AS,19]
20:[OF,20]
21:[ON,28]
22:[IN,22]
23:[ARE,23]
24:[YOU,28]
25:[BY,26]
26:[WITH,27]
27:[IS,27]
28:[IT,28]
29:[HER,30]
30:[OR,0]
31:[A,0]
```

# Improving Unsuccessful Searches

```
0:[THE,0]
1:[HAVE,3]
2:[TO,2]
3:[HIS,3]
4:[]
5:[BE,6]
6:[FOR,6]
7:[AND,18]
8:[I,8]
9:[BUT,10]
10:[WAS,10]
11:[HAD,12]
12:[HE,12]
13:[ARE,23]
14:[AS,19]
15:[NOT,16]
```

```
static inline void ordered_hash_table_insert
(std::vector<word_count>& dic, word_count wc)
{
    std::size_t bucket_size = dic.size();
    std::size_t h = hash(wc.word, bucket_size);
    while (!dic[h].word.empty())
    {
        if (wc.word.compare(dic[h].word) > 0)
            std::swap(dic[h], wc);
        if (--h == NOT_FOUND)
            h = bucket_size - 1;
    }
    dic[h] = wc;
}
```

```
16:[THAT,16]
17:[AT,20]
18:[WHICH,18]
19:[FROM,19]
20:[OF,20]
21:[BY,26]
22:[IN,22]
23:[THIS,23]
24:[IS,27]
25:[IT,28]
26:[ON,28]
27:[WITH,27]
28:[YOU,28]
29:[A,0]
30:[HER,30]
31:[OR,0]
```

# Improving Unsuccessful Searches

```
0:[THE,0]
1:[HAVE,3]
2:[TO,2]
3:[HIS,3]
4:[]
5:[BE,6]
6:[FOR,6]
7:[AND,18]
8:[I,8]
9:[BUT,10]
10:[WAS,10]
11:[HAD,12]
12:[HE,12]
13:[ARE,23]
14:[AS,19]
15:[NOT,16]
```

```
static inline std::size_t ordered_hash_table_search
(const std::vector<word_count>& dic, const std::string& word)
{
    std::size_t bucket_size = dic.size();
    std::size_t h = hash(word, bucket_size);
    while (!dic[h].word.empty())
    {
        int comp = word.compare(dic[h].word);
        if (comp > 0)
            break;
        if (comp == 0)
            return h;
        if (--h == NOT_FOUND)
            h = bucket_size - 1;
    }
    return NOT_FOUND;
}
```

```
16:[THAT,16]
17:[AT,20]
18:[WHICH,18]
19:[FROM,19]
20:[OF,20]
21:[BY,26]
22:[IN,22]
23:[THIS,23]
24:[IS,27]
25:[IT,28]
26:[ON,28]
27:[WITH,27]
28:[YOU,28]
29:[A,0]
30:[HER,30]
31:[OR,0]
```

Algorithms

# COOL DOWN

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

# Programas

- Eles são rodados pelo computador para executar tarefas específicas
  - Solucinar problemas

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, ...

Computers. Circuit layout, file system, compilers, ...

Computer graphics. Movies, video games, virtual reality, ...

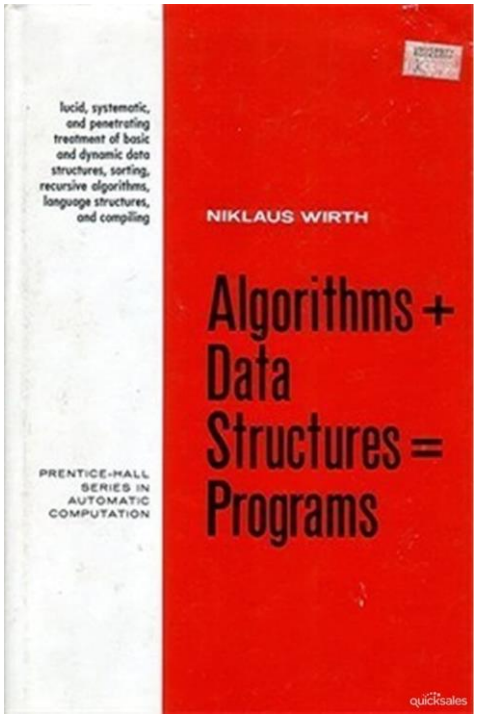
Security. Cell phones, e-commerce, voting machines, ...

Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

Social networks. Recommendations, news feeds, advertisements, ...

Physics. N-body simulation, particle collision simulation, ...

⋮




- Algoritmos + Estrutura de Dados

https://algs4.cs.princeton.edu/lectures/00Intro-2x2.pdf


# #include <algorithm>

Table 100 — Algorithms library summary

| Subclause | | Header(s) |
|---|---|---|
| 28.5 | Non-modifying sequence operations | |
| 28.6 | Mutating sequence operations | <algorithm> |
| 28.7 | Sorting and related operations | |
| 28.8 | C library algorithms | <cstdlib> |

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4713.pdf

- Existem mais de 100 algoritmos padrões bem testados
- Serve como base para novos algoritmos
- Compreensível e mais simples do que um *raw loop*
- Mantém *side-effects* dentro de uma interface bem definida
- Facilita o raciocínio sobre o problema
- Atua em conjunto com *iterators* ou *left-closed interval* $[begin, end)$

# Algoritmo

(versão declarativa com STL)

- Contar a incidência de um valor em uma estrutura de dados

```
std::vector<int> ys{ 1, 2, 2, 3, 1, 4, 5, 1, 2, 5, 2 };
std::cout << std::count(std::begin(ys), std::end(ys), 2) << "\n";

std::forward_list<int> zs{ 1, 2, 2, 3, 1, 4, 5, 1, 2, 5, 2 };
std::cout << std::count(std::begin(zs), std::end(zs), 2) << "\n";


std::count_if(std::begin(ys), std::end(ys),
              [](int y) { return !(y & 0x1) /* is even? */; })
```

# Algorithms

`#include <algorithm>`

- Searching a Computer's Memory

  `std::find`
- The Advantage of Order

  `std::lower_bound`
- Binary Tree Search

  `std::map` (Red-Black Tree → Balanced BST)
- Hashing

  `std::unordered_map`

  – *Separate Chaining* instead of *Linear Probing*
- Improving Unsuccessful Searches

# Algorithms

## #include <algorithm>

- between

```
std::tuple<std::vector<std::string>::const_iterator, std::vector<std::string>::const_iterator>
between(const std::vector<std::string>& words, const std::string& lhs, const std::string& rhs)
```

---

```
const std::vector<word_count> wcs = words_ordered_by_word();
std::vector<std::string> words;
std::transform(wcs.begin(), wcs.end(),
    std::back_inserter(words), [](const word_count& wc) { return wc.word; });
std::vector<std::string>::const_iterator first, last;

std::tie(first, last) = between(words, "HAVE", "NOTHING");
for (auto iter = first; iter != last; ++iter)
    std::cout << *iter << "  ";
std::cout << "\n";
```

---

```
HAVE  HE  HER  HIS  I  IN  IS  IT  NOT
```

# Algorithms

## #include <algorithm>

- between

```
std::tuple<std::vector<std::string>::const_iterator, std::vector<std::string>::const_iterator>
between(const std::vector<std::string>& words, const std::string& lhs, const std::string& rhs)
{
    //assert(lhs < rhs);
    //assert(std::is_sorted(words.begin(), words.end()));
    auto first = std::lower_bound(words.begin(), words.end(), lhs);
    auto last  = std::upper_bound(words.begin(), words.end(), rhs);
    return std::make_tuple(first, last);
}
```

# Algorithms

`#include <algorithm>`

- Generic Programming
  - 1988

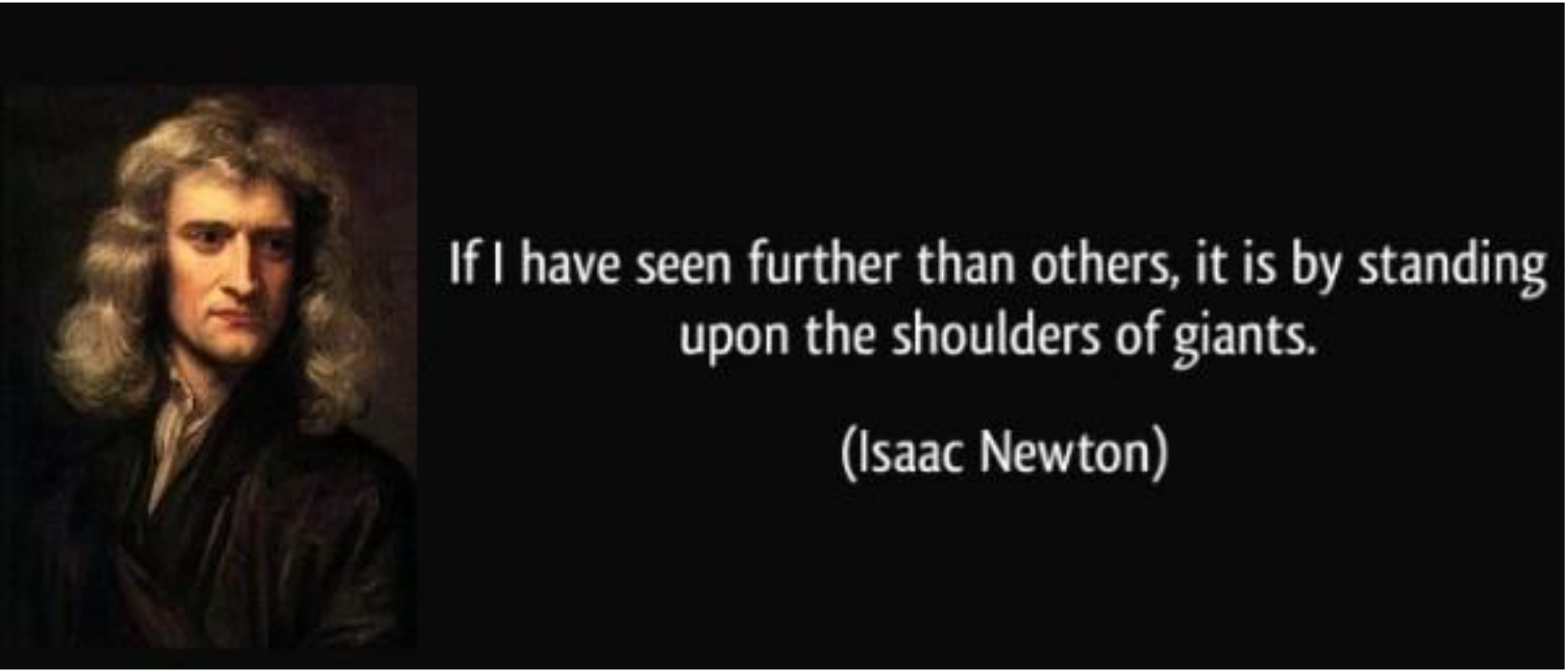
Generic programming centers around the idea of abstracting from concrete, efficient algorithms to obtain generic algorithms that can be combined with different data representations to produce a wide variety of useful software. For example, a class of generic sorting algorithms can be defined which work with finite sequences but which can be instantiated in different ways to produce algorithms working on arrays or linked lists.

```
std::vector<int> vec{ 33, 55, 11, 99, 88, 44, 22 };
std::list<int>   lst{ 33, 55, 11, 99, 88, 44, 22 };

std::stable_sort(vec.begin(), vec.end());
std::stable_sort(lst.begin(), lst.end());
```

http://stepanovpapers.com/genprog.pdf

https://link.springer.com/book/10.1007/3-540-51084-2

# On the Shoulders of Giants



# Stepanov on the Shoulders of Knuth

- Generic Programming is about abstracting and classifying algorithms and data structures
- It gets its inspiration from Knuth

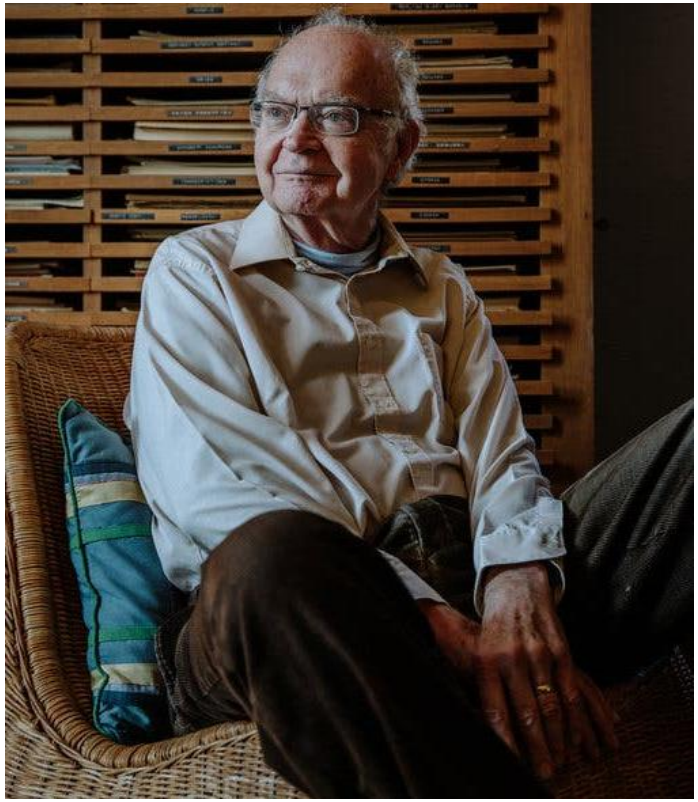
STL is only a limited success. While it became a widely used library, its central intuition did not get across. People confuse generic programming with using (and abusing) C++ templates. Generic programming is about abstracting and classifying algorithms and data structures. It gets its inspiration from Knuth and not from type theory. Its goal is the incremental construction of systematic catalogs of useful, efficient and abstract algorithms and data structures. Such an undertaking is still a dream.

Short History of STL

http://stepanovpapers.com/history%20of%20STL.pdf

# On the Shoulders of Giants

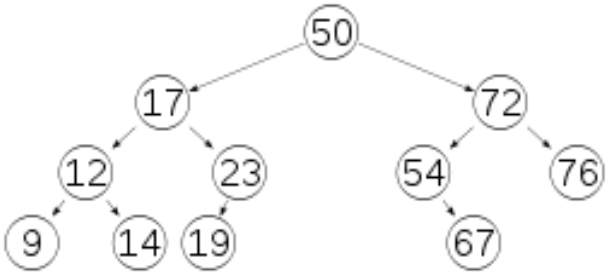
## Donald Knuth

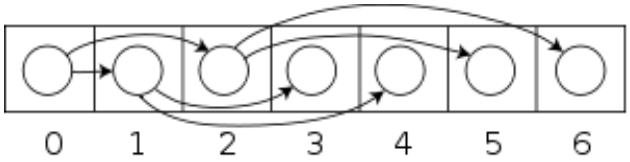


https://en.wikipedia.org/wiki/Donald_Knuth

## Alex Stepanov

https://en.wikipedia.org/wiki/Alexander_Stepanov




# #include <algorithm>

Fabio Galuppo, M.Sc.

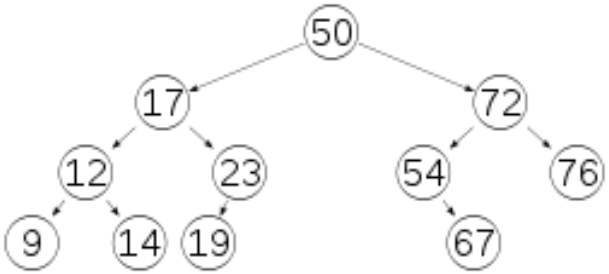
http://fabiogaluppo.com

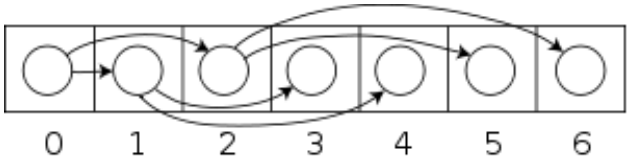
http://simplycpp.com/

http://github.com/fabiogaluppo

fabiogaluppo@acm.org

@FabioGaluppo




# Algoritmos com C++

# Recomendações

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com

http://simplycpp.com/

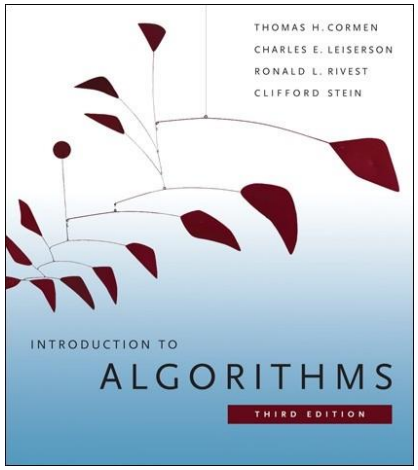
http://github.com/fabiogaluppo

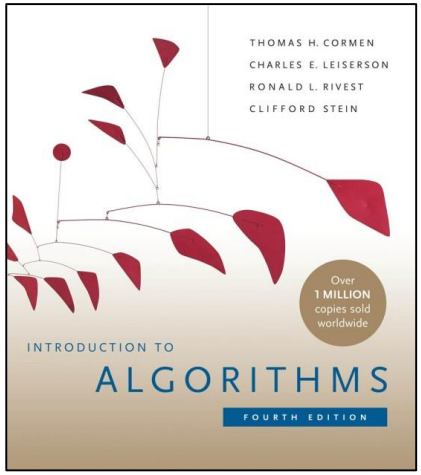
fabiogaluppo@acm.org

@FabioGaluppo

# Recomendações

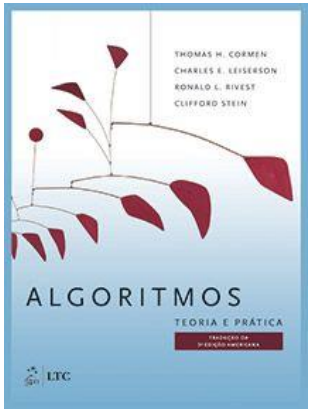
(livros)




**Introduction to Algorithms, Third Edition**
**Introduction to Algorithms, Fourth Edition**

- Um clássico!  Conhecido como livro do MIT, ou CLRS, ou livro do Cormen
- Ele aborda algoritmos, estrutura de dados e análise de algoritmos
- Pseudo-código
- 3a edição traduzida para língua portuguesa
- 4a edição lançada em 2022



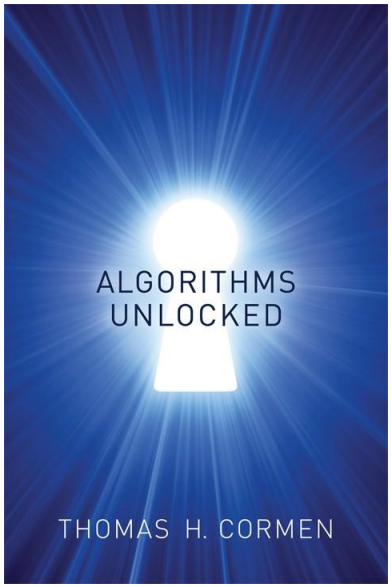

https://mitpress.mit.edu/books/introduction-algorithms-third-edition

https://mitpress.mit.edu/books/introduction-algorithms-fourth-edition

https://www.grupogen.com.br/algoritmos

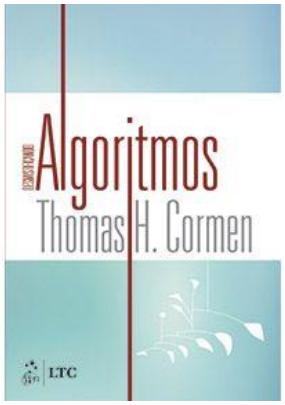
# Recomendações

(livros)




**Algorithms Unlocked**

- Uma “breve” introdução sobre algoritmos, estrutura de dados e análise de algoritmos
- Escrito por Thomas Cormen (C do CLRS)
- Pode ser considerado uma prelúdio ao livro do MIT
- Pseudo-código
- Tem traduzido em língua portuguesa
- Recomendo ler antes do livro do MIT

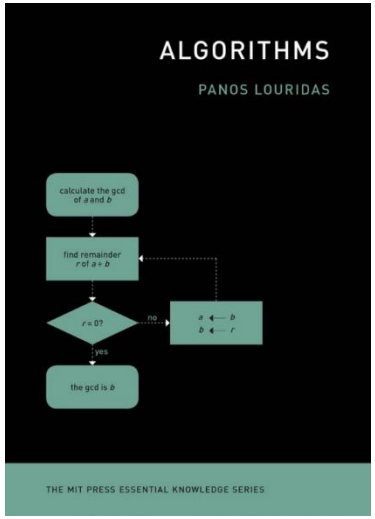



https://mitpress.mit.edu/books/algorithms-unlocked

https://www.grupogen.com.br/e-book-desmistificando-algoritmos

# Recomendações

(livros)



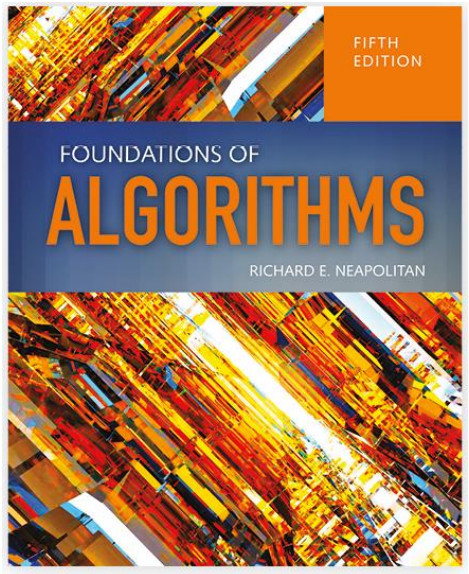

**Algoritms**

- Um *pocket book* da série The MIT Press Essential Knowledge
- Ele é sobre algoritmos e suas aplicações
  - Inclui tópicos sobre *Deep Learning* e *PageRank* (que deu origem ao Google)
- Praticamente um livro no estilo *popular science*
- Teoricamente, acessível para o público em geral. Isso não quer dizer que é um livro básico
- Logo no primeiro capítulo, ele trás exemplos emblemáticos que conectam música e algoritmos
- Não há código, mas tem diversas ilustrações como apoio

https://mitpress.mit.edu/books/algorithms

# Recomendações

(livros)



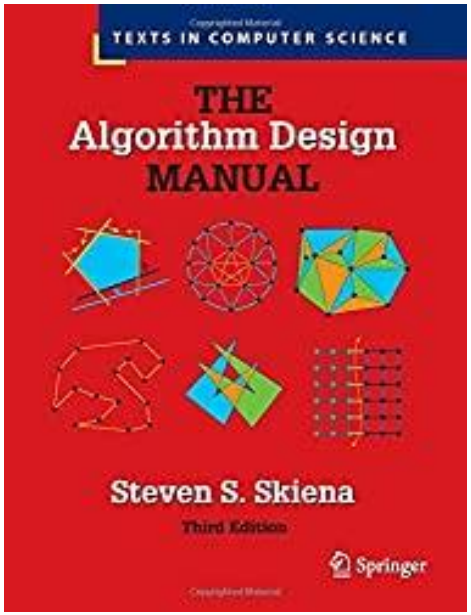

**Foundations of Algorithms**

- Ele aborda algoritmos, estrutura de dados e análise de algoritmos
- Os tópicos mais relevantes do livro do MIT são cobertos aqui, só que a linguagem adotada pelo autor me parece mais acessível
- Muitos exemplos, que ajudam na compreensão do algoritmos
- Pseudo-código mais próximo de C++
- Livro muito bem organizado

https://www.jblearning.com/science-technology/computing/data-structures-theory-of-computation/productdetails/9781284049190

# Recomendações

(livros)




**The Algorithm Design Manual**

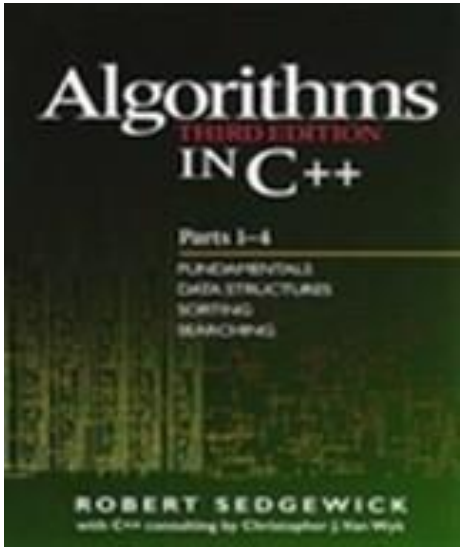
- Um clássico!  Conhecido como livro do Skiena
- Ele aborda algoritmos, estrutura de dados e análise de algoritmos
- Contém um catálogo de problemas resolvidos por algoritmos
  - A maioria desses problemas não estão resolvidos no livro, mas há dicas e referências para resolvê-los
- Pseudo-código e C

https://algorist.com/

# Recomendações

(livros)

**Algorithms in C++**




- Um clássico!  Conhecido como livro do Sedgewick
- Ele aborda algoritmos, estrutura de dados e análise de algoritmos
- Código em C++
- Livro 1 (Partes 1-4) trata das estruturas de dados (listas ligadas, filas, pilhas, *heaps*, …) e algoritmos (busca, ordenação, …) fundamentais
- Livro 2 (Parte 5) trata de grafos
- A edição mais recente: Algorithms 4th Edition
  - Junta e amplia o conteúdo dos livro Algorithms in C++
  - Código em Java
  - Possui um web site riquissimo com códigos e exemplos

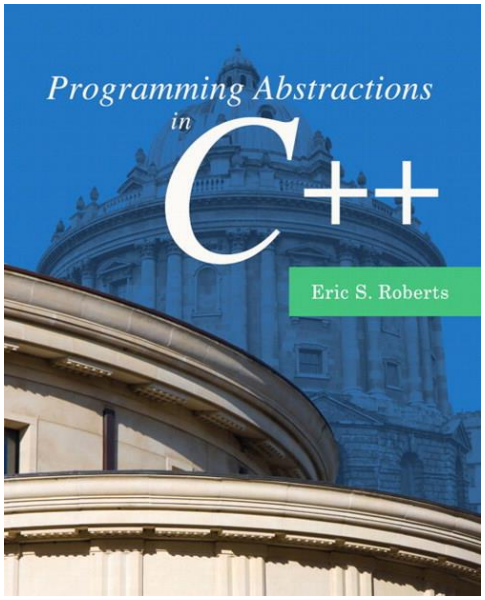
https://www.informit.com/store/algorithms-in-c-plus-plus-parts-1-4-fundamentals-data-9780134288604

https://www.informit.com/store/algorithms-in-c-plus-plus-part-5-graph-algorithms-9780201361186

https://algs4.cs.princeton.edu/home/

# Recomendações

(livros)




**Programming Abstractions in C++**

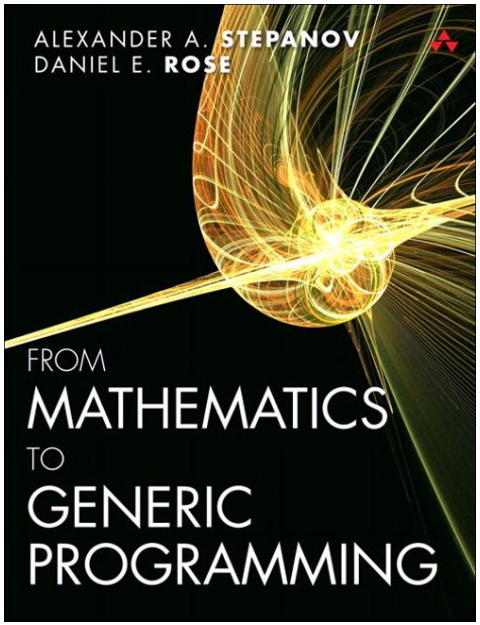
- Livro do Eric Roberts que ensinava na Universidade Stanford essa disciplina usando C++
- Ele aborda algoritmos, estrutura de dados e análise de algoritmos. Também é uma boa introdução a linguagem de programação C++. Muito focado na prática
- Código em C++
- Ele desenvolve uma biblioteca de algoritmos e estrutura de dados em C++, onde a base desse trabalho culminou na evolução para *The Stanford C++ Libraries*
- Também cobre tópicos da STL

https://www.informit.com/store/programming-abstractions-in-c-plus-plus-9780133454840

https://web.stanford.edu/dept/cs_edu/cppdoc/

# Recomendações

(livros)




**From Mathematics to Generic Programming**

- Livro do Alexander Stepanov, criador da STL
- Ele aborda algoritmos (em maior parte, os númericos)
  - O *rotate* é um algoritmos que se destacam no livro
- Não aborda STL
  - Aborda o racional por trás, por exemplo: *Iterators*
- Aborda o que inspirou a Programação Genérica: Algébra Abstrata e suas estrutura algébricas
- Muitos exemplos envolvendo Teoria dos Números
- Conecta fortemente Programação e Matemática
- Código em C++

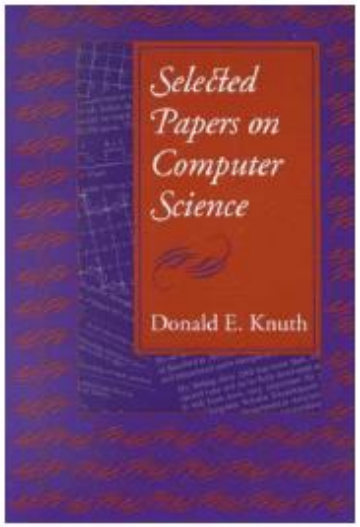
https://www.informit.com/store/from-mathematics-to-generic-programming-9780321942043

https://www.fm2gp.com/

# Recomendações

(livros)

**Selected Papers on Computer Science**

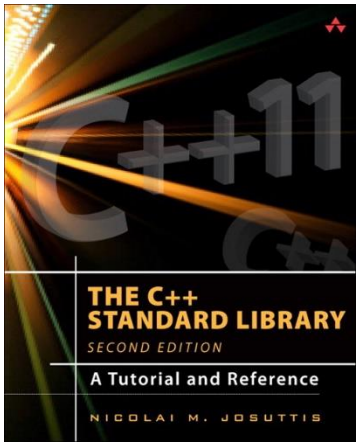


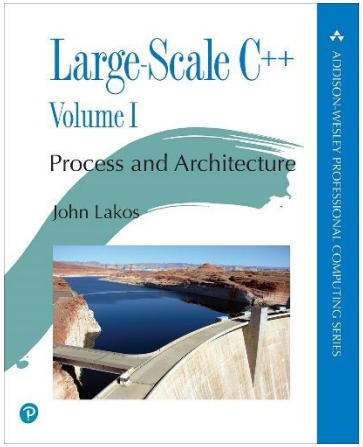

- Livro do lendário Donald Knuth, *The Art of Computer Programming (TAOCP)*
- Seleção de *papers* e artigos sobre algoritmos, estrutura de dados e análise de algoritmos
- Segundo livro da coleção *Selected Papers* que contém 8 livros na coleção
- A reimpressão do artigo *Algorithms* que saiu na revista *Scientific American* em Abril de 1977 vale a compra deste livro!
  - Apesar da data, o artigo se tornou atemporal

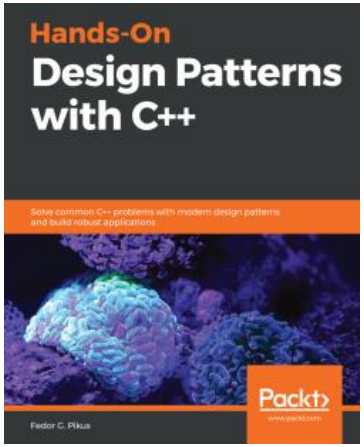
https://www-cs-faculty.stanford.edu/~knuth/cs.html

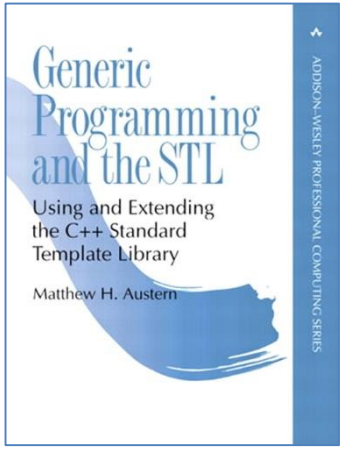
# Recomendações

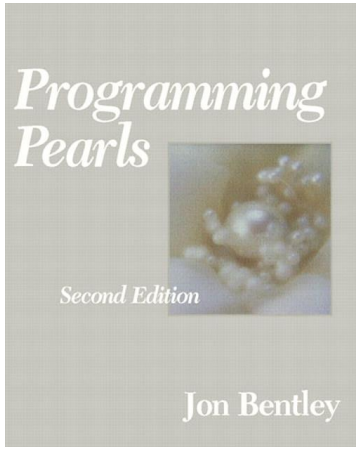
(livros)




Outros livros notáveis sobre C++, Programação Genérica, STL e/ou Algoritmos

https://www.informit.com/store/c-plus-plus-standard-library-a-tutorial-and-reference-9780321623218

https://www.informit.com/store/large-scale-c-plus-plus-volume-i-process-and-architecture-9780201717068

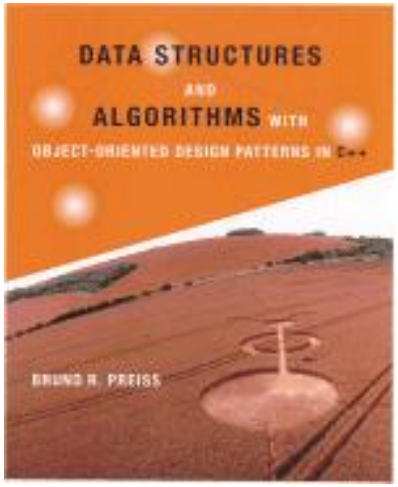
https://www.packtpub.com/product/hands-on-design-patterns-with-c/9781788832564

https://www.informit.com/store/generic-programming-and-the-stl-using-and-extending-9780201309560

https://www.informit.com/store/programming-pearls-9780201657883

# Recomendações

(livros gratuitos, *papers*)

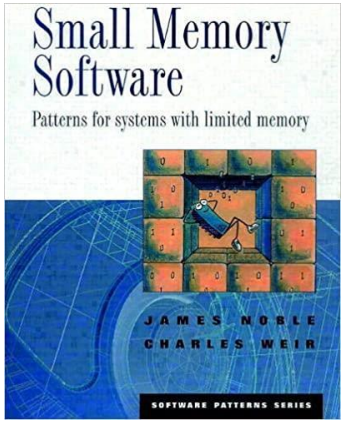



**Data Structures and Algorithms with Object-Oriented Design Patterns in C++**

https://web.archive.org/web/20161220060802/http://www.brpreiss.com/books/opus4/

**Small Memory Software**
**Patterns for systems with limited memory**

http://smallmemory.com/book.html




**Notes on Programming**

http://stepanovpapers.com/notes.pdf

**Thriving in a crowded and changing world: C++ 2006–2020**

https://dl.acm.org/doi/10.1145/3386320

# Recomendações

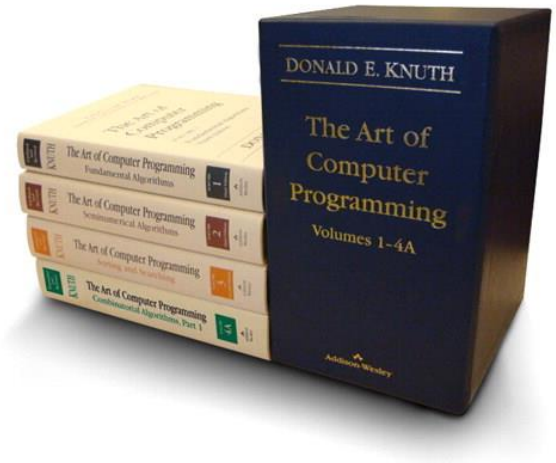
(vídeos, *lectures,* referências)

- Programming Conversations
  - https://www.youtube.com/watch?v=k-meLQaYP5Y&list=PLHxtyCq_WDLXFAEA-lYoRNQIezL_vaSX-
- Efficient Programming with Components
  - https://www.youtube.com/watch?v=aIHAEYyoTUc&list=PLHxtyCq_WDLXryyw91lahwdtpZsmo4BGD
- Talks e Keynotes do Sean Parent
  - https://sean-parent.stlab.cc/papers-and-presentations/#presentations
- Introduction to Algorithms (MIT Opencourseware)
  - https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/
- Algorithms Princeton (Coursera)
  - https://www.coursera.org/learn/algorithms-part1
  - https://www.coursera.org/learn/algorithms-part2
- Awesome Algorithms
  - https://github.com/tayllan/awesome-algorithms

# Recomendação Especial

## (The Art of Computer Programming)

**The Art of Computer Programming (TAOCP)**



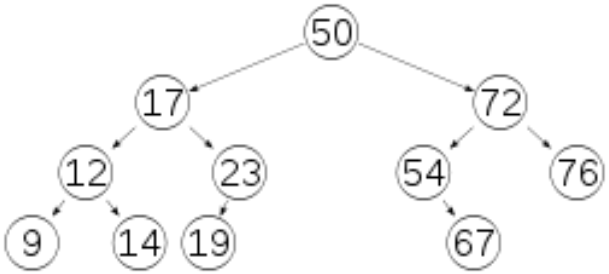

- Obra-prima do lendário Donald Knuth conhecido, entre outras coisas, por ter “criado” a disciplina Análise de Algoritmos
- Os livros abordam algoritmos, estrutura de dados e análise de algoritmos de uma forma rigorosa com muita “Matemática Concreta”. A bíblia do assunto! Obra ainda incompleta.
- Compre pelo menos para deixar na estante e impressionar seus amigos e familiares ☺

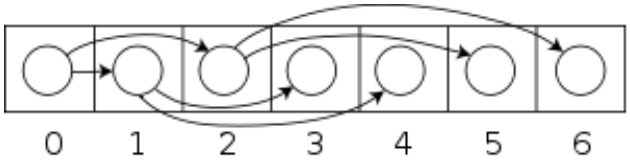
**I find that I don't understand things unless I try to program them.**

**Donald E. Knuth**
Professor Emeritus at Stanford University

https://www.informit.com/store/art-of-computer-programming-volumes-1-4a-boxed-set-9780321751041

https://www-cs-faculty.stanford.edu/~knuth/taocp.html




# Algoritmos com C++

# Recomendações

Fabio Galuppo, M.Sc.

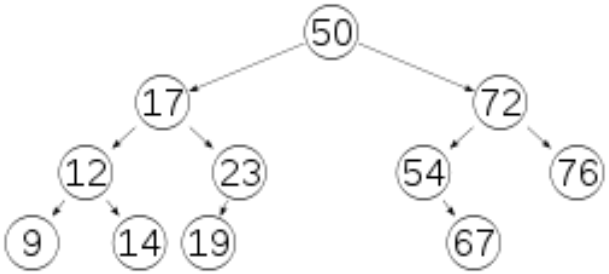
http://fabiogaluppo.com

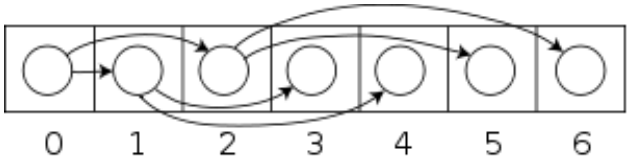
http://simplycpp.com/

http://github.com/fabiogaluppo

fabiogaluppo@acm.org

@FabioGaluppo




# Algoritmos com C++

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com

http://simplycpp.com/

http://github.com/fabiogaluppo

fabiogaluppo@acm.org

@FabioGaluppo

www.agit.com.br

Curso

# Algoritmos com C++

## Fundamentos e Prática para Soluções de Problemas

*por Fabio Galuppo*