

Data Structures and Algorithms Lab Sessions

C. Martínez

October 13, 2019



Briefing

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

- We use the *Jutge* (<https://jutge.org>), a virtual learning environment for computer programming
- Use your institutional email (`@est.fib.upc.edu`) and the *Racó* password to access your *Jutge* account; you should have received an email to enroll in the course EDA Curs 2019/2020 Q1
- There is a list of programming exercises associated to each lab session

- We use the *Jutge* (<https://jutge.org>), a virtual learning environment for computer programming
- Use your institutional email (`@est.fib.upc.edu`) and the *Racó* password to access your *Jutge* account; you should have received an email to enroll in the course **EDA Curs 2019/2020 Q1**
- There is a list of programming exercises associated to each lab session

- We use the *Jutge* (<https://jutge.org>), a virtual learning environment for computer programming
- Use your institutional email (`@est.fib.upc.edu`) and the *Racó* password to access your *Jutge* account; you should have received an email to enroll in the course **EDA Curs 2019/2020 Q1**
- There is a list of programming exercises associated to each lab session

- Data Structures and Algorithms course webpage: www.cs.upc.edu/eda-eng (English version)
- Access lecture notes, exams, problem sets, etc. from **Teaching Material**. Most of the documents and other materials are available in English; you might also want to check the Catalan version of the website for additional materials
- A brief **cheatsheet on STL** is available from the course webpage too
- We encourage you to take the **Self-Assessment Lab Test**. It has 11 questions, if you have problems to answer correctly too many of these or you do not understand the “solution” check with your lab prof
- A guide to programming style is available in Catalan at the website; an English translation is also available

The Standard Template Library

Divide and Conquer

- Data Structures and Algorithms course webpage: www.cs.upc.edu/eda-eng (English version)
- Access lecture notes, exams, problem sets, etc. from **Teaching Material**. Most of the documents and other materials are available in English; you might also want to check the Catalan version of the website for additional materials
- A brief [cheatsheet on STL](#) is available from the course webpage too
- We encourage you to take the [Self-Assessment Lab Test](#). It has 11 questions, if you have problems to answer correctly too many of these or you do not understand the “solution” check with your lab prof
- A guide to programming style is available in Catalan at the website; an English translation is also available

Briefing

- Data Structures and Algorithms course webpage: www.cs.upc.edu/eda-eng (English version)
- Access lecture notes, exams, problem sets, etc. from **Teaching Material**. Most of the documents and other materials are available in English; you might also want to check the Catalan version of the website for additional materials
- A brief [cheatsheet on STL](#) is available from the course webpage too
- We encourage you to take the [Self-Assessment Lab Test](#). It has 11 questions, if you have problems to answer correctly too many of these or you do not understand the “solution” check with your lab prof
- A guide to programming style is available in Catalan at the website; an [English translation](#) is also available

- Data Structures and Algorithms course webpage: www.cs.upc.edu/eda-eng (English version)
- Access lecture notes, exams, problem sets, etc. from **Teaching Material**. Most of the documents and other materials are available in English; you might also want to check the Catalan version of the website for additional materials
- A brief [cheatsheet on STL](#) is available from the course webpage too
- We encourage you to take the [Self-Assessment Lab Test](#). It has 11 questions, if you have problems to answer correctly too many of these or you do not understand the “solution” check with your lab prof
- A guide to programming style is available in Catalan at the website; an [English translation](#) is also available

- Data Structures and Algorithms course webpage: www.cs.upc.edu/eda-eng (English version)
- Access lecture notes, exams, problem sets, etc. from **Teaching Material**. Most of the documents and other materials are available in English; you might also want to check the Catalan version of the website for additional materials
- A brief [cheatsheet on STL](#) is available from the course webpage too
- We encourage you to take the [Self-Assessment Lab Test](#). It has 11 questions, if you have problems to answer correctly too many of these or you do not understand the “solution” check with your lab prof
- A guide to programming style is available in Catalan at the website; an [English translation](#) is also available

1 The Standard Template Library

2 Divide and Conquer

The Standard Template Library

- The Standard Template Library (**STL**, for short) is a fundamental component of the C++ Standard Library
- STL **is not** part of the C++ language, but virtually all, except the most trivial programs, use it
- The evolution of the STL has gone hand in hand with the evolution of C++; its design and use has shaped the language itself
- There are full specifications of the STL (functional and non-functional requirements); each compiler-vendor has freedom to implement it as they want as long as all requirements are met

The Standard Template Library

- The Standard Template Library (**STL**, for short) is a fundamental component of the C++ Standard Library
- STL **is not** part of the C++ language, but virtually all, except the most trivial programs, use it
- The evolution of the STL has gone hand in hand with the evolution of C++; its design and use has shaped the language itself
- There are full specifications of the STL (functional and non-functional requirements); each compiler-vendor has freedom to implement it as they want as long as all requirements are met

The Standard Template Library

- The Standard Template Library (**STL**, for short) is a fundamental component of the C++ Standard Library
- STL **is not** part of the C++ language, but virtually all, except the most trivial programs, use it
- The evolution of the STL has gone hand in hand with the evolution of C++; its design and use has shaped the language itself
- There are full specifications of the STL (functional and non-functional requirements); each compiler-vendor has freedom to implement it as they want as long as all requirements are met

The Standard Template Library

- The Standard Template Library (**STL**, for short) is a fundamental component of the C++ Standard Library
- STL **is not** part of the C++ language, but virtually all, except the most trivial programs, use it
- The evolution of the STL has gone hand in hand with the evolution of C++; its design and use has shaped the language itself
- There are full specifications of the STL (functional and non-functional requirements); each compiler-vendor has freedom to implement it as they want as long as all requirements are met

The Standard Template Library

- The STL provides a rich number of general-purpose algorithms and data structures: vectors, lists, queues, sorting, . . .
- STL design and implementation heavily relies on **templates** (compile-time genericity) as opposed to inheritance (run-time genericity)
- STL offers highly-tuned, state-of-the-art implementations for common algorithms and data structures; check if it provides a solution to your problem, don't reinvent the wheel!

The Standard Template Library

- The STL provides a rich number of general-purpose algorithms and data structures: vectors, lists, queues, sorting, . . .
- STL design and implementation heavily relies on **templates** (compile-time genericity) as opposed to inheritance (run-time genericity)
- STL offers highly-tuned, state-of-the-art implementations for common algorithms and data structures; check if it provides a solution to your problem, don't reinvent the wheel!

The Standard Template Library

- The STL provides a rich number of general-purpose algorithms and data structures: vectors, lists, queues, sorting, . . .
- STL design and implementation heavily relies on **templates** (compile-time genericity) as opposed to inheritance (run-time genericity)
- STL offers highly-tuned, state-of-the-art implementations for common algorithms and data structures; check if it provides a solution to your problem, don't reinvent the wheel!

Example

```
// Pre: v.size() > 0, T has a total order <
// Post: min(v) returns the minimum element in v
template <typename T>
T minimum(const vector<T>& v) {
    T the_min = v[0];
    for (int i = 1; i < v.size(); ++i)
        if (v[i] < the_min) the_min = v[i];
    return the_min;
}
```

Templates

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

Example

```
vector<double> w;  
...  
// instantiates minimum with T = double  
double m = minimum(w);  
vector<string> words = read_words();  
// instantiates minimum with T = string  
string lexmin = minimum(words);
```

Example

```
template <typename T>
class BoundedStack {
private:
    vector<T> cont;
    int num_elems;
public:
    const int DEFAULT_MAX_ELEMS = 100;
    BoundedStack(int max_elems = DEFAULT_MAX_ELEMS) :
        cont(max_elems), num_elems(0) {};
    ...
    void push(const T& x) {
        if (num_elems < max_elems) {
            cont[num_elems] = x;
            ++num_elems;
        } else { ... }
    }
    T pop() {
        if (num_elems > 0) {
            --num_elems;
            return cont[num_elems];
        } else { // error, empty stack!
        }
    }
}
```

Example

```
BoundedStack<double> S1(100);
double x;
while (cin >> x) S1.push(x);
...
// creates a vector of 30 bounded stacks, each of size 10
vector< BoundedStack<int> > S2(30, 10);
...
for (int i = 0; i < 30; ++i) {
    cout << "Stack " << i << ":" << endl;
    while (not S2[i].empty())
        cout << S2[i].pop() << endl;
}
// creates a bounded stack of 30 vectors, each holding 10 ints
BoundedStack< vector<int> > S2(30, 10);
...
```

Containers, Iterators, Algorithms

The three fundamental concepts in the STL are:

- 1 **Containers**: a container is a collection or set of objects where we can perform different operations such as adding new objects, removing objects, examine the objects and perform some computation on each of them, etc.
- 2 **Iterators**: an iterator is an abstraction for a pointer to an object, they allow us to access, move around and *iterate* over the objects in a container
- 3 **Algorithms**: general-purpose algorithms to make some computation or update the contents of a container, e.g., copying, finding an element that satisfies a property, sorting, mapping a function on each object of a container, etc.

Containers, Iterators, Algorithms

The three fundamental concepts in the STL are:

- 1 **Containers**: a container is a collection or set of objects where we can perform different operations such as adding new objects, removing objects, examine the objects and perform some computation on each of them, etc.
- 2 **Iterators**: an iterator is an abstraction for a pointer to an object, they allow us to access, move around and *iterate* over the objects in a container
- 3 **Algorithms**: general-purpose algorithms to make some computation or update the contents of a container, e.g., copying, finding an element that satisfies a property, sorting, mapping a function on each object of a container, etc.

Containers, Iterators, Algorithms

The three fundamental concepts in the STL are:

- 1 **Containers**: a container is a collection or set of objects where we can perform different operations such as adding new objects, removing objects, examine the objects and perform some computation on each of them, etc.
- 2 **Iterators**: an iterator is an abstraction for a pointer to an object, they allow us to access, move around and *iterate* over the objects in a container
- 3 **Algorithms**: general-purpose algorithms to make some computation or update the contents of a container, e.g., copying, finding an element that satisfies a property, sorting, mapping a function on each object of a container, etc.

Containers, Iterators, Algorithms

You are already familiar with several STL containers:

- `vector<T>`
- `list<T>` (doubly-linked lists)
- `stack<T>`
- `queue<T>`

Containers, Iterators, Algorithms

Typical operations included in (almost) all containers:

- `size`: returns the number of objects in the collection
- `push`/`push_back`: adds a new element at the end of the collection
- `pop`/`pop_back`: removes first/last element inserted
- `empty`: returns true iff the collection contains no object
- `begin`: returns an iterator to the first object in the container
- `end`: returns a fictitious iterator past-the-end of the container

Using Containers and Iterators

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

Example

```
#include <algorithm>
#include <list>
...
string w;
list<string> L;
while (cin >> w) L.push_back(w);
vector<string> v(L.size());
int k = 0;
for (list<string>::const_iterator it = L.begin();
     it != L.end(); ++it) {
    // ++it advances the iterator to the successor
    // *it accesses the string it points to
    v[k] = *it;
    ++k;
}
// copy(L.begin(), L.end(), v.begin()) does the same as loop above
sort(v.begin(), v.end());
L.sort(); // sort(L.begin(), L.end()) does not work
```

Generic Algorithms with Iterators

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

Example

```
// Pre: [beg, end) contains at least one element, comp
// Post: min(v) returns the minimum element in range
template <typename Iterator,
          typename T, typename Comparator = less<T>>
T minimum(Iterator beg, Iterator end, Comparator smaller) {
    T the_min = *beg;
    for (Iterator it = beg; it != end; ++it)
        if (smaller(*it, the_min))
            the_min = *it;
    return the_min;
}

struct Person {
    int age;
    ...
};

bool is_younger(const Person& a, const Person& b) {
    return a.age < b.age;
}

...
vector<Person> P;
Person benjamin = minimum(v.begin(), v.end(), is_younger);
...
```

Using Containers and Iterators

- The new standard C++11 introduces a new handy syntax to iterate through a container:

```
vector<string> words;
...
for (string w : words) {
    // w iterates through all the values in the
    // vector 'words'
    cout << ' ' << w;
}
```

- C++11 also introduces the keyword **auto** which will be substituted by a typename deduced by the compiler from the context

```
// auto => list<string>::const_iterator
for (auto it = L.begin(); it != L.end(); ++it) {
    v[k] = *it;
    ++k;
}
// auto => string
for (auto w : words) {
    ...
}
```

Using Containers and Iterators

- The new standard C++11 introduces a new handy syntax to iterate through a container:

```
vector<string> words;
...
for (string w : words) {
    // w iterates through all the values in the
    // vector 'words'
    cout << ' ' << w;
}
```

- C++11 also introduces the keyword **auto** which will be substituted by a typename deduced by the compiler from the context

```
// auto => list<string>::const_iterator
for (auto it = L.begin(); it != L.end(); ++it) {
    v[k] = *it;
    ++k;
}
// auto => string
for (auto w : words) {
    ...
}
```

Priority Queues

- A **priority queue** is a collection of objects which can be accessed in (descending) order of priority; each element is “identified” with its priority, so we access elements from largest to smallest.
- Technically speaking `priority_queue` is a **container adaptor** built on top of a container with random access iterators, e.g., `vector`
- Operations:

Method	Description	Method	Description
<code>empty</code>	returns true iff PQ is empty	<code>size</code>	returns number of elems
<code>pop</code>	remove elem of largest priority	<code>ctor()</code>	creates empty PQ
<code>push</code>	add new elem	<code>ctor(beg, end)</code>	creates PQ with elems in the range [beg,end)
<code>top</code>	return most priority elem		

Priority Queues

- A **priority queue** is a collection of objects which can be accessed in (descending) order of priority; each element is “identified” with its priority, so we access elements from largest to smallest.
- Technically speaking `priority_queue` is a **container adaptor** built on top of a container with random access iterators, e.g., `vector`
- Operations:

Method	Description	Method	Description
<code>empty</code>	returns true iff PQ is empty	<code>size</code>	returns number of elems
<code>pop</code>	remove elem of largest priority	<code>ctor()</code>	creates empty PQ
<code>push</code>	add new elem	<code>ctor(beg, end)</code>	creates PQ with elems in the range [beg,end)
<code>top</code>	return most priority elem		

- A **priority queue** is a collection of objects which can be accessed in (descending) order of priority; each element is “identified” with its priority, so we access elements from largest to smallest.
- Technically speaking `priority_queue` is a **container adaptor** built on top of a container with random access iterators, e.g., `vector`
- Operations:

Method	Description	Method	Description
<code>empty</code>	returns true iff PQ is empty	<code>size</code>	returns number of elems
<code>pop</code>	remove elem of largest priority	<code>ctor()</code>	creates empty PQ
<code>push</code>	add new elem	<code>ctor(beg, end)</code>	creates PQ with elems in the range [beg,end)
<code>top</code>	return most priority elem		

```
#include <queue>
struct ChemElement {
    string symbol;
    double atomic_weight;
}

bool smaller_weight(...) { return X.atomic_weight < Y.atomic_weight; }

// new initialization syntax in C++11
vector<ChemElement> AllChemElements = { {"H", 1.008}, {"He", 4.003},
    ..., {"U", 238.03}};

priority_queue<ChemElement, vector<ChemElement>, smaller_weight> PT;
for (auto e : AllChemElements)
    PT.push(e);

while (not PT.empty()) { // print from highest atomic weight ("U")
    // to lowest ("H")
    ChemElement e = PT.top(); PT.pop();
    cout << e.symbol << " (" << e.atomic_weight << ")" << endl;
}

// it is significantly more efficient to fill PT with:
priority_queue<ChemElement, vector<ChemElement>,
    smaller_weight> PT(AllChemElements.begin(),
        AllChemElements.end());
```

Priority Queues

- Priority queues are usually implemented in C++ with **heaps** (to be explained in detail in theory lectures).
- Cost of the operations:

Method	Cost	Method	Cost
empty	$O(1)$	size	$O(1)$
pop	$O(\log n)$	ctor()	$O(1)$
push	$O(\log n)$	ctor(beg, end)	$O(n)$
top	$O(1)$		

- Notice that n insertions (with push) into an initially empty priority queue have cost $O(n \log n)$; inserting n elems with `priority_queue(beg, end)` has cost $O(n)$

Priority Queues

- Priority queues are usually implemented in C++ with **heaps** (to be explained in detail in theory lectures).
- Cost of the operations:

Method	Cost	Method	Cost
empty	$O(1)$	size	$O(1)$
pop	$O(\log n)$	ctor()	$O(1)$
push	$O(\log n)$	ctor(beg, end)	$O(n)$
top	$O(1)$		

- Notice that n insertions (with `push`) into an initially empty priority queue have cost $O(n \log n)$; inserting n elems with `priority_queue(beg, end)` has cost $O(n)$

Priority Queues

- Priority queues are usually implemented in C++ with **heaps** (to be explained in detail in theory lectures).
- Cost of the operations:

Method	Cost	Method	Cost
empty	$O(1)$	size	$O(1)$
pop	$O(\log n)$	ctor()	$O(1)$
push	$O(\log n)$	ctor(beg, end)	$O(n)$
top	$O(1)$		

- Notice that n insertions (with `push`) into an initially empty priority queue have cost $O(n \log n)$; inserting n elems with `priority_queue(beg, end)` has cost $O(n)$

- A `set<T>` is a finite set of objects in which we can efficiently add new elements, remove existing elements and search if a given element is or not in the set. Moreover, we can iterate through all elements of the set in ascending order.
- Operations:

Method	Description	Method	Description
<code>empty</code>	returns true iff set is empty	<code>size</code>	returns number of elems
<code>erase(it)</code>	remove element pointed to by <code>it</code>	<code>ctor()</code>	creates empty set
<code>insert</code>	add new elem	<code>ctor(beg, end)</code>	creates set with elems in the range <code>[beg,end)</code>
<code>begin</code>	returns iterator to smallest element	<code>end</code>	returns iterator past-the-end
<code>S.find(x)</code>	returns iterator to <code>x</code> if $x \in S$, end if $x \notin S$		

- A `set<T>` is a finite set of objects in which we can efficiently add new elements, remove existing elements and search if a given element is or not in the set. Moreover, we can iterate through all elements of the set in ascending order.
- Operations:

Method	Description	Method	Description
<code>empty</code>	returns true iff set is empty	<code>size</code>	returns number of elems
<code>erase(it)</code>	remove element pointed to by <code>it</code>	<code>ctor()</code>	creates empty set
<code>insert</code>	add new elem	<code>ctor(beg, end)</code>	creates set with elems in the range <code>[beg,end)</code>
<code>begin</code>	returns iterator to smallest element	<code>end</code>	returns iterator past-the-end
<code>S.find(x)</code>	returns iterator to x if $x \in S$, end if $x \notin S$		

Example

```
#include <set>
#include <cstdlib>
...
set<int> generate_random_subset(int n, int k) {
    set<int> C;
    while (C.size() != k) {
        int r = rand() \% n + 1; // r = random number in 1..n
        C.insert(r); // does nothing if r already in C
    }
}
...
set<int> lotto = generate_random_subset(49, 6);
cout << "Winning numbers:" << endl;
// prints the k selected integers in ascending order
for (int x : lotto) {
    cout << ' ' << x;
    cout << endl;
}
...
```


- Sets are usually implemented in C++ with some variant of **balanced search trees**, for instance, red-black trees (they'll be explained in theory lectures).
- Cost of the operations:

Method	Cost	Method	Cost
<code>empty</code>	$O(1)$	<code>size</code>	$O(1)$
<code>erase(it)</code>	$O(\log n)$	<code>ctor()</code>	$O(1)$
<code>insert</code>	$O(\log n)$	<code>ctor(beg, end)</code>	$O(n \log n)$
<code>begin</code>	$O(\log n)$	<code>end</code>	$O(1)$
<code>find</code>	$O(\log n)$		

- Incrementing an iterator has worst-case $O(\log n)$, but n increments have total cost $O(n) \Rightarrow$ amortized cost of `++it` is $O(1)$

- Sets are usually implemented in C++ with some variant of **balanced search trees**, for instance, red-black trees (they'll be explained in theory lectures).
- Cost of the operations:

Method	Cost	Method	Cost
<code>empty</code>	$O(1)$	<code>size</code>	$O(1)$
<code>erase(it)</code>	$O(\log n)$	<code>ctor()</code>	$O(1)$
<code>insert</code>	$O(\log n)$	<code>ctor(beg, end)</code>	$O(n \log n)$
<code>begin</code>	$O(\log n)$	<code>end</code>	$O(1)$
<code>find</code>	$O(\log n)$		

- Incrementing an iterator has worst-case $O(\log n)$, but n increments have total cost $O(n) \Rightarrow$ amortized cost of `++it` is $O(1)$

- Sets are usually implemented in C++ with some variant of **balanced search trees**, for instance, red-black trees (they'll be explained in theory lectures).
- Cost of the operations:

Method	Cost	Method	Cost
<code>empty</code>	$O(1)$	<code>size</code>	$O(1)$
<code>erase(it)</code>	$O(\log n)$	<code>ctor()</code>	$O(1)$
<code>insert</code>	$O(\log n)$	<code>ctor(beg, end)</code>	$O(n \log n)$
<code>begin</code>	$O(\log n)$	<code>end</code>	$O(1)$
<code>find</code>	$O(\log n)$		

- Incrementing an iterator has worst-case $O(\log n)$, but n increments have total cost $O(n) \Rightarrow$ amortized cost of `++it` is $O(1)$

The STL provides the convenient class `pair`, which is used by several other classes and functions in order to input or output information.

Example

```
template <typename T1, typename T2>
struct pair {
    T1 first;
    T2 second;
};

pair<int, string> p = {3, "hello"};
pair<double, int> q = make_pair(3.14, 7);
cout << "(" << p.first << ", " << p.second << " )";
```

- A `map<K, V>` is a finite set of pairs `< key,value >` such that no two pairs have the same key, in which we can efficiently add new pairs, remove existing pairs, update the value associated to a key and search for a key and retrieve its associated value if present. Moreover, we can iterate through all pairs in the map in ascending order of keys.
- Operations:

Method	Description	Method	Description
<code>empty</code>	returns true iff set is empty	<code>size</code>	returns number of elems
<code>erase(it)</code>	remove pair pointed to by <code>it</code>	<code>ctor()</code>	creates empty set
<code>insert(p)</code>	add new pair <code>p=<k,v></code>	<code>ctor(beg, end)</code>	creates set with elems in the range <code>[beg,end)</code>
<code>begin</code>	returns iterator to pair with smallest key	<code>end</code>	returns iterator past-the-end
<code>S.find(k)</code>	returns iterator to pair with key <code>k</code>	<code>operator[](k)</code>	returns reference to value associated to key <code>k</code> , adding a pair if necessary

- A `map<K, V>` is a finite set of pairs `< key,value >` such that no two pairs have the same key, in which we can efficiently add new pairs, remove existing pairs, update the value associated to a key and search for a key and retrieve its associated value if present. Moreover, we can iterate through all pairs in the map in ascending order of keys.
- Operations:

Method	Description	Method	Description
<code>empty</code>	returns true iff set is empty	<code>size</code>	returns number of elems
<code>erase(it)</code>	remove pair pointed to by <code>it</code>	<code>ctor()</code>	creates empty set
<code>insert(p)</code>	add new pair <code>p=<k,v></code>	<code>ctor(beg, end)</code>	creates set with elems in the range <code>[beg,end)</code>
<code>begin</code>	returns iterator to pair with smallest key	<code>end</code>	returns iterator past-the-end
<code>S.find(k)</code>	returns iterator to pair with key <code>k</code>	<code>operator[] (k)</code>	returns reference to value associated to key <code>k</code> , adding a pair if necessary

Example

```
#include <map>
...
map<string,int> word_freqs;
string w;

while (cin >> w)
    ++word_freqs[w];

// print the list of words in the input
// in alphabetical order and their frequencies
for (auto p : word_freqs)
    cout << p.first << ": " << p.second << endl;

auto it = word_freqs.find("abracadabra");
if (it != word_freqs.end()) {
    cout << "this was a magic text!" << endl;
    word_freqs.remove(it);
}
```

- Maps, like sets, are usually implemented in C++ with some variant of **balanced search trees**.
- Cost of the operations:

Method	Cost	Method	Cost
empty	$O(1)$	size	$O(1)$
erase(it)	$O(\log n)$	ctor()	$O(1)$
insert	$O(\log n)$	ctor(beg, end)	$O(n \log n)$
begin	$O(\log n)$	end	$O(1)$
find	$O(\log n)$	operator[]	$O(\log n)$

- As in sets, n iterator increments have total cost $O(n)$, amortized cost of `++it` is $O(1)$

- Maps, like sets, are usually implemented in C++ with some variant of **balanced search trees**.
- Cost of the operations:

Method	Cost	Method	Cost
empty	$O(1)$	size	$O(1)$
erase(it)	$O(\log n)$	ctor()	$O(1)$
insert	$O(\log n)$	ctor(beg, end)	$O(n \log n)$
begin	$O(\log n)$	end	$O(1)$
find	$O(\log n)$	operator[]	$O(\log n)$

- As in sets, n iterator increments have total cost $O(n)$, amortized cost of `++it` is $O(1)$

- Maps, like sets, are usually implemented in C++ with some variant of **balanced search trees**.
- Cost of the operations:

Method	Cost	Method	Cost
empty	$O(1)$	size	$O(1)$
erase(it)	$O(\log n)$	ctor()	$O(1)$
insert	$O(\log n)$	ctor(beg, end)	$O(n \log n)$
begin	$O(\log n)$	end	$O(1)$
find	$O(\log n)$	operator[]	$O(\log n)$

- As in sets, n iterator increments have total cost $O(n)$, amortized cost of `++it` is $O(1)$

More on Containers

- The containers discussed here offer a very rich set of methods, we have described here just a few
- `multiset`, `multimap`: like sets and maps, but duplicities (of elements, of keys) are allowed
- The order used in sets and maps can be changed by supplying a specific parameter when instantiating the container

More on Containers

- The containers discussed here offer a very rich set of methods, we have described here just a few
- `multiset`, `multimap`: like sets and maps, but duplicities (of elements, of keys) are allowed
- The order used in sets and maps can be changed by supplying a specific parameter when instantiating the container

The Standard Template Library

Divide and Conquer

Example

```
set< double, greater<double> > S;  
double x;  
while (cin >> x) S.insert(x);  
for (auto it = S.begin(); it != S.end(); ++it)  
    cout << *it << " ";  
if (next != S.end())  
    S.insert(-x);  
    cout << " " << x << " ";  
    it = next;  
}
```

More on Containers

- The containers discussed here offer a very rich set of methods, we have described here just a few
- `multiset`, `multimap`: like sets and maps, but duplicities (of elements, of keys) are allowed
- The order used in sets and maps can be changed by supplying a specific parameter when instantiating the container

Example

```
set< double, greater<double> > S;  
double x;  
while (cin >> x) S.insert(x);  
for (auto it = S.begin(); it != S.end(); ) {  
    auto next = ++it;  
    if (next != S.end())  
        if (*next - *it > 0.01)  
            cout << ' ' << *it;  
    it = next;  
}
```

More on Containers

- The containers discussed here offer a very rich set of methods, we have described here just a few
- `multiset`, `multimap`: like sets and maps, but duplicities (of elements, of keys) are allowed
- The order used in sets and maps can be changed by supplying a specific parameter when instantiating the container

Example

```
set< double, greater<double> > S;  
double x;  
while (cin >> x) S.insert(x);  
for (auto it = S.begin(); it != S.end(); ) {  
    auto next = ++it;  
    if (next != S.end())  
        if (*next - *it > 0.01)  
            cout << ' ' << *it;  
    it = next;  
}
```

More on Containers

- The newest standards (2011, 2014) introduce `unordered_set` and `unordered_map` (and the multi-*** variants). These offer almost the same functionality as `sets` and `maps`, except that iteration in order of elements/keys is not possible, but the average cost of insertions, deletions and searches is $O(1)$.
- For instance, our example of counting word frequencies has cost $O(N \log n)$, where n is the number of distinct words in the text and N the length of the text; if we replace `map` by `unordered_map` the average cost drops down to $O(N)$, but the output comes in no particular order
- Unordered sets and maps are implemented with hash tables; basic hashing schemes will be explained in theory.

More on Containers

- The newest standards (2011, 2014) introduce `unordered_set` and `unordered_map` (and the multi-*** variants). These offer almost the same functionality as `sets` and `maps`, except that iteration in order of elements/keys is not possible, but the average cost of insertions, deletions and searches is $O(1)$.
- For instance, our example of counting word frequencies has cost $O(N \log n)$, where n is the number of distinct words in the text and N the length of the text; if we replace `map` by `unordered_map` the average cost drops down to $O(N)$, but the output comes in no particular order
- Unordered sets and maps are implemented with hash tables; basic hashing schemes will be explained in theory.

More on Containers

- The newest standards (2011, 2014) introduce `unordered_set` and `unordered_map` (and the multi-*** variants). These offer almost the same functionality as `sets` and `maps`, except that iteration in order of elements/keys is not possible, but the average cost of insertions, deletions and searches is $O(1)$.
- For instance, our example of counting word frequencies has cost $O(N \log n)$, where n is the number of distinct words in the text and N the length of the text; if we replace `map` by `unordered_map` the average cost drops down to $O(N)$, but the output comes in no particular order
- Unordered sets and maps are implemented with hash tables; basic hashing schemes will be explained in theory.

Remarks on the Programming Assignments

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

- Use `g++ -std=c++11` to compile C++11 programs
- Read carefully each exercise statement
- Identify which is the most suitable container to efficiently solve the problem
- Write a draft of your solution, do not care about the syntax at this point

Remarks on the Programming Assignments

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

- Use `g++ -std=c++11` to compile C++11 programs
- Read carefully each exercise statement
- Identify which is the most suitable container to efficiently solve the problem
- Write a draft of your solution, do not care about the syntax at this point

Remarks on the Programming Assignments

- Use `g++ -std=c++11` to compile C++11 programs
- Read carefully each exercise statement
- Identify which is the most suitable container to efficiently solve the problem
- Write a draft of your solution, do not care about the syntax at this point

Remarks on the Programming Assignments

- Use `g++ -std=c++11` to compile C++11 programs
- Read carefully each exercise statement
- Identify which is the most suitable container to efficiently solve the problem
- Write a draft of your solution, do not care about the syntax at this point

Remarks on the Programming Assignments

- Fill-in the details checking the cheatsheet, these slides, some source from in the internet, ...
- Inefficient solutions will not be likely accepted by the *Judge*, the private tests consist of huge inputs which will break down inefficient solutions, e.g., an $O(n^2)$ solution for a problem where a reasonable algorithm should have cost $O(n \log n)$
- Ask your lab prof for help!

Remarks on the Programming Assignments

- Fill-in the details checking the cheatsheet, these slides, some source from in the internet, . . .
- Inefficient solutions will not be likely accepted by the *Jutge*, the private tests consist of huge inputs which will break down inefficient solutions, e.g., an $O(n^2)$ solution for a problem where a reasonable algorithm should have cost $O(n \log n)$
- Ask your lab prof for help!

Remarks on the Programming Assignments

- Fill-in the details checking the cheatsheet, these slides, some source from in the internet, . . .
- Inefficient solutions will not be likely accepted by the *Jutge*, the private tests consist of huge inputs which will break down inefficient solutions, e.g., an $O(n^2)$ solution for a problem where a reasonable algorithm should have cost $O(n \log n)$
- Ask your lab prof for help!

- A very handy and convenient on-line reference manual for C++ in general (and the STL in particular)
<http://www.cplusplus.com/reference>
Other documents, tutorials, etc. can also be found at www.cplusplus.com
- The most authoritative reference on the STL is the book “*The C++ Standard Library 2nd ed.*” by Nicolai M. Josuttis
- Another important reference for the C++ is “*The C++ Programming Language 3rd ed.*” by Bjarne Stroustrup (creator of C++)

1 The Standard Template Library

2 Divide and Conquer

Divide and Conquer

- **Divide and Conquer** is an algorithm design technique which allows us to develop very efficient solutions to some algorithmic fundamental problems
- It is also an important principle behind the design of some efficient data structures
- Examples: binary search, quicksort, quickselect, mergesort, fast multiplication, fast matrix multiplication, FFT, ...

Divide and Conquer

```
procedure DIVIDE_AND_CONQUER( $x$ )  
  if  $x$  is simple then  
    return DIRECT_SOLUTION( $x$ )  
  else  
     $\langle x_1, x_2, \dots, x_k \rangle :=$  DIVIDE( $x$ )  
    for  $i := 1$  to  $k$  do  
       $y_i :=$  DIVIDE_AND_CONQUER( $x_i$ )  
    end for  
    return COMBINE( $y_1, y_2, \dots, y_k$ )  
  end if  
end procedure
```

Sometimes, when $k = 1$, the algorithm is called a *reduction algorithm*.

Divide and Conquer

- Most D&C algorithms are recursive: reason about their correctness by induction
- Generalize the problem using **embeddings** (cat: *immersions*)
- Obtain the specification of the embedded function either by precondition strengthening or postcondition weakening (check your Programming 2 lecture notes, or [my PRO2 lecture \(in Catalan\)](#))

Divide and Conquer

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

Example

Write an efficient recursive function that returns the position of a value x in the sorted vector v , or the position after which x should be inserted to keep v in ascending order, if x is not in v .

```
// Pre: v is sorted in ascending order
template <typename T>
int position(const vector<T>& v, const T& x);
// Post: i = position(v, x) => -1 <= i < n and v[i] <= x < v[i+1]
```

Example

Embedding: solve the problem in the subvector
 $v[\textit{left}..\textit{right}]$

```
// Pre: v[left..right] is sorted in ascending order
// -1 <= left <= right-1, right <= n = v.size()
// v[left] <= x < v[right]
template <typename T>
int position_rec(const vector<T>& v, const T& x, int left, int right);
// Post: i = position(v, x, left, right) =>
//       (left <= i < right and v[i] <= x < v[i+1])
```

N.B. Use the convention that $v[-1] = -\infty$ and
 $v[n] = +\infty$ to define the meaning of the logical
expressions above

Divide and Conquer

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

Example

```
template <typename T>
int position_rec(const vector<T>& v, const T& x, int left, int right) {
    if (left == right-1) return left;
    // left < right-1
    int mid = (left + right) / 2;
    // left < mid < right
    if (v[mid] <= x)
        return position_rec(v, x, mid, right);
    else
        return position_rec(v, x, left, mid);
}
```


Some Useful C++ Features

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

- Use **function overloading** to advantage

Example

```
template <typename T>
int position(const vector<T>& v, const T& x,
            int left, int right) {
    ...
}

template <typename T>
int position(const vector<T>& v, const T& x) {
    return position(v, x, -1, v.size());
}
```

Some Useful C++ Features

EDA Lab

C. Martínez

- Use **operator overloading** for cleaner and user-friendly code

The Standard Template Library

Divide and Conquer

Example

```
typedef vector<double> Row;
typedef vector<Row> Matrix;
Matrix operator*(const Matrix& A, const Matrix& B) {
    ...
}
ostream& operator<<(ostream& os, const Matrix& A) {
    for (int i = 0; i < A.size(); ++i) {
        for (int j = 0; j < A[i].size(); ++j)
            os << (j > 0) ? " " : "" << A[i][j];
        os << endl;
    }
    return os;
}

int main() {
    Matrix A, B;
    ...
    cout << "Result = " << A * B << endl;
}
```

- Check, for instance, en.cppreference.com/w/cpp/language/operators for additional info

Divide and Conquer: Jutge

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

P81966	Dichotomic search	Binary search
P84219	First occurrence	
P33412	Resistant dichotomic search	
P34682	Fixed points	
X82938	Search in an unimodal vector	

Divide and Conquer: Jutge

EDA Lab

C. Martínez

The Standard Template
Library

Divide and Conquer

P29212	Modular exponentiation	Fast exponentiation
P61833	Powers of a matrix	
P74219	Fibonacci numbers (2)	
X39049	Powers of permutations	
P58512	Interest rates	Bisection
P80595	How many inversions	Mergesort-like

Divide and Conquer: Judge

- 1 Make sure to solve at least one problem from each block
- 2 Subdivide the “Search in an unimodal vector” (X82938) into two subproblems: finding the *peak*, then searching for x
- 3 Avoid trial & error: think carefully about your pre- and postconditions and reason about correctness; this is valid in all cases, but will be specially useful in the “Binary search” block

Divide and Conquer: Jutge

- 4 The efficient computation of “Fibonacci numbers” (P74219) needs fast matrix exponentiation (P61833) as a previous step. Indeed, the key to solve the problem is to find 2×2 matrix A such that

$$A^n = \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix}$$

Divide and Conquer: Judge

- 5 In “Interest rates” (P58512) compute (1) the amount of money $B(d)$ that must be returned to the bank if the money is lent for d days; (2) the amount of money $R(d)$ that must be returned to Prof. Oak if the money is lent for d days. Both functions are increasing, but $R(d) - B(d)$ is decreasing and so what we want is to find the smallest $d \geq 1$ such that $R(d) - B(d) < 0$. Try $d = 1, d = 2, d = 4, d = 8, \dots$, until a value $d = 2^k$ such that $R(d) - B(d) < 0$ is found; find the solution by bisection in the range $[2^{k-1}, 2^k]$

Divide and Conquer: Judge

- Some of the problems in the “Divide and Conquer” list have been used in past Computer Exams: “Fibonacci numbers (2)”, “Powers of permutations”, “Search in a unimodal vector”
- Other D&C problems in past Computer Exams:
 - Rightmost position of insertion ([P54070](#))
 - Bi-increasing vector ([P99753](#))