**Data Structure:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre | Tamanio | Elementos caracteristicos |  |  |
| vector |  |  |  |  |
| map |  |  |  |  |
| array |  |  |  |  |
| list |  |  |  |  |
| set |  |  |  |  |
| stack |  |  |  |  |
| queue |  |  |  |  |
| deque |  |  |  |  |
| auto |  |  |  |  |
| matrix |  |  |  |  |

**Arrays: (por ej p[n])**

Accessing time: O(1)

Syntax: p[i]=?

Search time: O(n) para busqueda secuencial, O(log n) para busqueda binaria

Busqueda secuencial:

find(p, p+n, elemento)

Busqueda binaria:

(Hay que definir una funcion aparte. Generalmente luce mas o menos asi, si quiero buscar a elemento)

int binary(int p[], int n, int elemento){

int left=-1; // left tiene que ser un elemento que definitivamente no cumpla con la condicion de lo que quiero buscar

int right= n+1; // right tiene que ser un elemento que definitivamente cumpla con la condicion

while(left <= right){

int m = (left+right)/2;

if(p[m] == k)

return true;

if(p[m] <k)

left = m+1;

else

right = m-1;

}

return false;

}

Insertar y eliminar elementos no tiene sentido en un array porque es de tamanio inmutable.

**Vectores:**

Accessing time: O(n)

Syntax: v[i]

Buscar un elemento: O(n)

find(v.begin(), v.end(), val)

Insertar un elemento: O(1)

v.push\_back(elemento)

Remover un elemento (o varios): O(n)

v.erase(elemento) // si es uno solo

v.erase(elemento inicial, elemento inicial+ cantidad de elementos hasta donde queres borrar)

Limpiar el vector: O(n)

v.clear() // no recibe nada

**Set:**

Sets are containers that store unique elements following a specific order.

Sets are typically implemented as *binary search trees*.

Elements in associative containers are referenced by their *key* and not by their absolute position in the container.

No se puede acceder a un elemento de un set a partir de su index. Se hace unicamente moviendo un iterador!!

Devolver lo q hay en un set:

forn(i, sz(myset)){

int x = \*std::next(myset.begin(), i);

cout<< x <<endl;

}

Chequear si un set es vacio:

myset.empty() // devuelve un booleano

Agregar un elemento: O(log n)

//forma normal:

myset.insert(elemento)

// forma optimizada:

set<int> myset;

set<int>::iterator it;

myset.insert(it, elemento)

Eliminar un elemento: O(1) si eliminas un solo elemento, O(log n) para subintervalos, O(n) para clear

Removes from the [set](http://www.cplusplus.com/set) container either a single element or a range of elements ([first,last)).

myset.erase()

Encontrar un elemento O(log n)

myset.find(elemento)

Contar cuantas veces aparece un elemento en un set:

((Searches the container for elements equivalent to *val* and returns the number of matches.))

myset.count(elemento)

lower\_bound O(log n)

upper\_bound

equal\_range

**Map:**

Maps are associative containers that store elements formed by a combination of a *key value* and a *mapped value*, following a specific order.

Accessing time:

Stack:

Queue;

Deque:

**Linked lists:**