

Ordenamiento e introducción a STL

❖ Cubetas.

Las cubetas son una estructura de datos basada en arreglos cuyo funcionamiento consiste en poner cada elemento en su lugar correspondiente.

Dicho comportamiento permite que sea usado como diccionario, para ordenar, etc.

<http://codeforces.com/contest/339/problem/A>

Cuando es utilizado para ordenar números, se obtiene un algoritmo conocido como bucket sort, consiste en acomodar cada número de la secuencia original en su 'cajita' correspondiente y después se recorren todas las cubetas sacando cada elemento en ellas.

```
#include <bits/stdc++.h>
using namespace std;
const int mN = 1e5;
int main(){
    int n, a[mN] = {0}, mxN = INT_MIN;
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> x;
        a[x]++;
        mxN = max(mxN, x);
    }
    for(int i = 0; i < 1 + mxN; i++)
        for(int j = 0; j < a[i]; j++)
            cout << i << " ";
    return 0;
}
```

Este es un algoritmo de ordenamiento especial, sirve para conjuntos de números que se ubican en un rango pequeño (con soporte para aproximadamente 10^5 cubetas).

Por eso no funciona para todos los casos, pero sí ofrece una complejidad mejor que los demás: $O(n)$ cuando los números están uniformemente distribuidos y $O(n + k)$ cuando no, siendo k el número más grande del conjunto.

❖ Burbuja.

También conocido como bubble sort, es un algoritmo intuitivo que se programa de manera sencilla y tiene una complejidad de $O(n^2)$.

[Problem - 230A - Codeforces](#)

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n;
    cin >> n;
    int a[n+3];
    for(int i = 0; i < n; cin >> a[i++]);
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (a[j] > a[j+1])
                swap(a[j], a[j+1]);
    for(int i = 0; i < n; i++)
        cout << a[i] << " ";
    return 0;
}
```

La idea principal de este algoritmo es intercambiar los números de todo par adyacente que esté en desorden.

Ejemplo. Ordenar la siguiente secuencia de números: (5, 1, 4, 2, 8).

(**5**, 1, 4, 2, 8) Se intercambian los primeros dos números porque $5 > 1$.

(1, **5**, 4, 2, 8) $5 > 4$.

(1, 4, **5**, 2, 8) $5 > 2$.

(1, 4, 2, **5**, 8) No se intercambian porque $5 < 8$.

El proceso se repite.

(**1**, 4, 2, 5, 8) No hay cambios.

(1, **4**, 2, 5, 8) $4 > 2$.

(1, 2, **4**, 5, 8) El arreglo ya quedó ordenado, pero el algoritmo no lo sabe, así que continúa hasta que termine el ciclo.

Nota. Una propiedad de este algoritmo es que el i - ésimo número más grande queda en su lugar en la i - ésima iteración. En este caso, el 8 es el número más grande del arreglo y quedó en la última posición desde la primera pasada.

❖ Pair

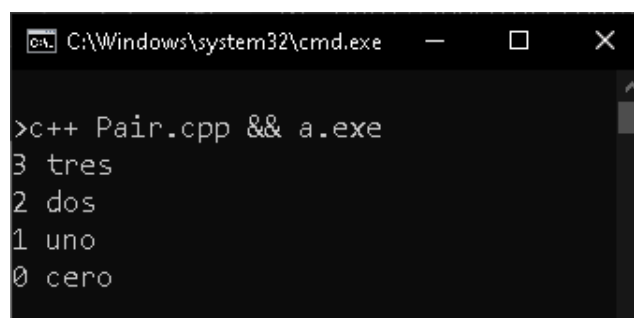
El par es un simple contenedor de dos elementos que pueden ser o no del mismo tipo. Los elementos de esta estructura se accesan por medio de las palabras first y second. Dichos atributos comúnmente son reemplazados por un token al utilizar:

```
#define f first  
#define s second
```

Declarar, inicializar y modificar un par se puede lograr de distintas maneras:

```
#include <iostream>  
#define f first  
#define s second  
using namespace std;  
int main(){  
    //pair<dato1, dato2> nombre;  
    pair<int, string> p(3, "tres");  
    cout << p.f << " " << p.s << "\n";  
    p = make_pair(2, "dos");  
    cout << p.f << " " << p.s << "\n";  
    p = {1, "uno"};  
    cout << p.f << " " << p.s << "\n";  
    p.f = 0, p.s = "cero";  
    cout << p.f << " " << p.s << "\n";  
    return 0;  
}
```

El código anterior da como salida:



```
C:\Windows\system32\cmd.exe  
>c++ Pair.cpp && a.exe  
3 tres  
2 dos  
1 uno  
0 cero
```

*Nota: Si no se utilizan los *#define*, es necesario hacer uso de la palabra completa.

```
par.first;           par.second;
```

❖ Vector

Un vector es un contenedor secuencial que representa a un arreglo con posibilidad de cambiar en tamaño. De esta forma, justo como en los arreglos, se puede acceder a cualquier elemento (puede ser cualquier tipo) de una manera muy eficiente.

Podemos construir un vector de las siguientes maneras:

```
#include <vector>
using namespace std;
int main(){
    int n = 3, k = 10;
    //vector<dato> nombre;
    vector<int> u; //Declara un vector vacío.
    vector<int> d(n); //Vector de tamaño n, inicializado en 0.
    vector<int> t(n, k); //Tamaño n, con valores iniciales = k
    vector<int> v = {1, 2, 3}; //Vector a partir de un listado.
    return 0;
}
```

Y los métodos más utilizados son:

```
/* CAPACIDAD */
v.size(); /*size_type: Número de elementos. (unsigned int)
v.empty(); //bool: 1 si está vacío, 0 si no.

/* ACCESO */
//v[i]: Como en arreglos, accede a la posición i del vector.
int a = v[0];
v[1] = 10;

v.front(); //Referencia al primer elemento del vector.
v.back(); //Ref al último.

/* MODIFICADORES */
v.push_back(k); //Inserta 'k' al final del vector.
v.pop_back(); //Elimina el último elemento.

v.assign(n, k); //Asigna el valor k a los primeros n elementos.
v.clear(); //Borra todos los elementos.
```

❖ Sort

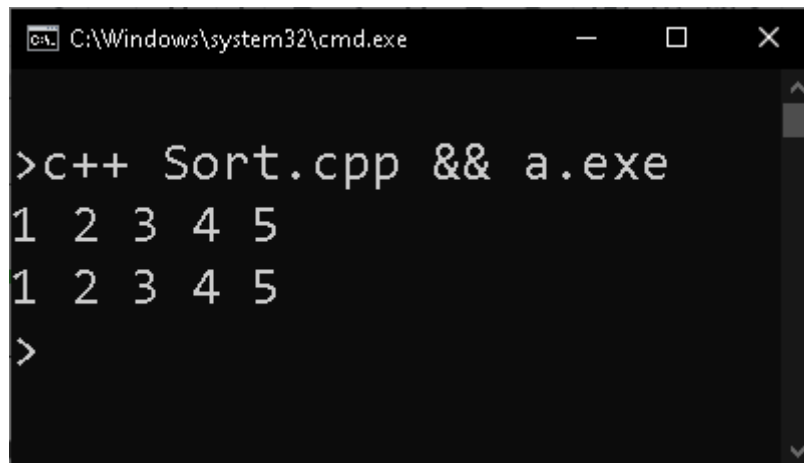
Complejidad: $O(n \log n)$

Función de la librería estándar que ordena los elementos de un contenedor de forma ascendente y necesita de dos parámetros.

Comúnmente se utiliza en arreglos estáticos o dinámicos.

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main(){
    int n = 5;
    int a[] = {3, 2, 1, 4, 5};
    sort(a, a + n);
    vector<int> b = {5, 4, 3, 1, 2};
    sort(b.begin(), b.end());
    for(int i = 0; i < n; cout << a[i++] << " ");
    cout << endl;
    for(auto x: b) cout << x << " ";
    return 0;
}
```

Y el código nos da como resultado:



```
C:\Windows\system32\cmd.exe

>c++ Sort.cpp && a.exe
1 2 3 4 5
1 2 3 4 5
>
```

Dicha función tiene un tercer parámetro opcional que recibe un comparador para ordenar de distintas maneras. Si se desea de forma descendente, la función se vería algo así:

```
sort(b.begin(), b.end(), greater<int>());
```