Niñas PRO(grama doras)

Niñas PRO(grama doras)

Funciones de ordenamiento en C++

Niñas PRO(grama doras)

Temario

- ★ Algoritmos de ordenamiento
- ★ Análisis de costos
- ★ Sort en C++

Niñas PRO(grama)doras

Algoritmos Eficientes de Ordenamiento

Niñas PRO(grama doras)

Quicksort

Arreglos Ordenados

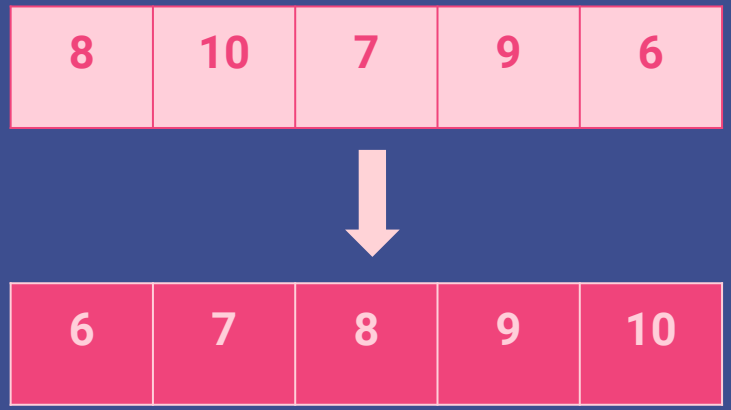
Un arreglo ordenado tiene todos sus elementos en un **orden particular**. Por ejemplo:

arreglo en orden alfabético

arreglo de números enteros en orden **descendente**

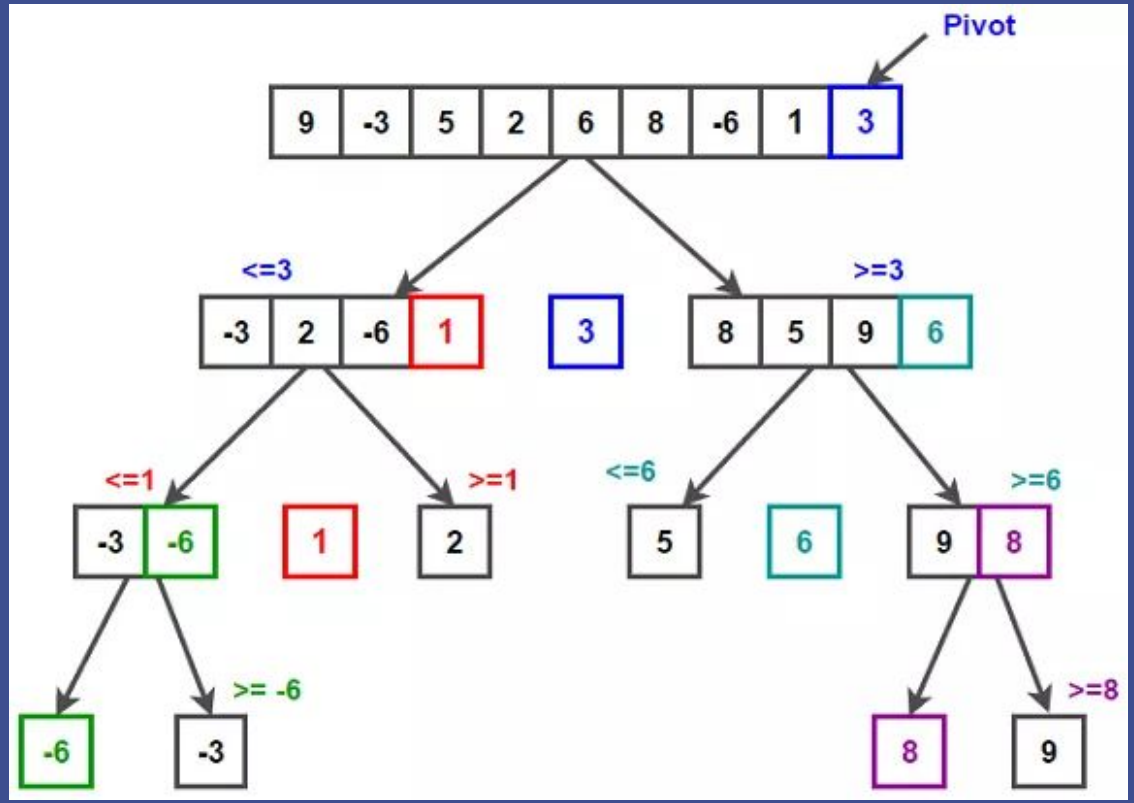
Algoritmos de Ordenamiento

Los algoritmos de ordenamiento reciben un arreglo y retornan otro con los elementos ordenados, en base a algún **criterio de ordenamiento**.



Quicksort

- Elegir un elemento del arreglo de elementos a ordenar, al que llamaremos pivote.
- Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores.
- La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.


Pivot
9 -3 5 2 6 8 -6 1 3
<=3
-3 2 -6 1
3
8 5 9 6
>=3
<=1
-3 -6
1
2
>=1
<=6
5
6
9 8
>=6
-6
-3
>= -6
8
9
>=8

Quicksort

Veamos cómo funciona el algoritmo de quicksort en el siguiente video de Geeksforgeeks:

https://youtu.be/PgBzjlCcFvc?t=26

Actividad

Indica cómo se irá ordenando el arreglo en la siguiente iteración.

Niñas PRO(grama doras)

Código Quicksort

En primer lugar, es necesaria una función que intercambie dos elementos de un arreglo.

```
// Swap two elements
void swap(int arr[], int i, int j)
{
    int t = arr[i];
    arr[i] = arr[j];
    arr[j] = t;
}
```

Código Quicksort

Luego, es necesaria una función que tome como pivote el último elemento del arreglo, y lo ubique correctamente en el arreglo ordenado.

```
// Particionar
int partition (int arr[], int low, int high)
{
     int pivot = arr[high]; // pivot
     int i = (low - 1); // Index of smaller element

     for (int j = low; j <= high- 1; j++)
     {
          if (arr[j] <= pivot)
          {
               i++;
               swap(arr, i, j);
          }
     }
     swap(arr, i + 1, high);
     return (i + 1);
}
```

Código Quicksort

Finalmente, tenemos la función que implementa el algoritmo de **quicksort**.

```
// Quicksort
void quickSort(int arr[], int low, int high)
{
      if (low < high)
      {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
      }
}

// Main
int main()
{
      int arr[] = {10, 7, 8, 9, 1, 5};
      int n = sizeof(arr)/sizeof(arr[0]);
      quickSort(arr, 0, n-1);
      return 0;
}
```

Niñas PRO(grama doras)

Mergesort

Merge Sort

- Se basa en Recursión
- **Caso Base :**
  - **Una lista vacía o de un elemento ya está ordenada**
- Caso Recursivo:
  - Dividir la lista en 2 partes.
  - Ordenar recursivamente ambas partes.
  - Mezclar las dos sublistas en una sola lista ordenada.

Pseudocódigo Merge Sort

```
MergeSort(arr[], l,  r)

If r > l:

 1.   Encuentra el punto medio para dividir el arreglo en
      dos partes:

           middle m = (l+r)/2

 2.   Llame a mergeSort para la primera mitad:

           Call mergeSort(arr, l, m)

 3.   Llame a mergeSort para la segunda mitad:

           Call mergeSort(arr, m+1, r)

 4.   Mezcle las dos mitades ordenadas en los pasos 2 y 3:

           Call merge(arr, l, m, r)
```

6 5 3 1 8 7 2 4

Merge Sort

Veamos cómo funciona el algoritmo de MergeSort en el siguiente video de Geeksforgeeks:

https://www.youtube.com/watch?v=JSceec-wEyw

Código Merge Sort


```
#include<stdlib.h>
#include<stdio.h>
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}
```

Función Merge

```
void merge(int arr[], int l, int m, int r) {

    int i, j, k;

    int n1 = m - l + 1;

    int n2 =  r - m;

* create temp arrays */

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)

        R[j] = arr[m + 1+ j];
```

Función Merge

```
i = 0 j = 0 k = l;

while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++; }

        else{

            arr[k] = R[j];

            j++;

        }

        k++;

    }
```

Función Merge

```
    while (i < n1) {

        arr[k] = L[i];

        i++;

        k++;

    }

    while (j < n2) {

        arr[k] = R[j];

        j++;

        k++; }

}
```

Actividad

Indica cómo se irá ordenando el arreglo con el algoritmo Merge Sort.

3	9	1	7	5

Niñas PRO(grama doras)

Análisis de Costo

En el cuadro tenemos una comparación de los costos de los algoritmos que vimos hoy. La cantidad de veces que se ejecuta cada algoritmo de ordenamiento es variable, por esta razón se comparan el caso promedio, el mejor y el peor caso.

Algoritmo	Mejor caso	Peor caso	Caso promedio
`Quick sort`	$O(nlogn)$	$O(n^2)$	$O(nlogn)$
`Merge sort`	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$

Niñas PRO(grama)doras

Ordenamiento en C++

Niñas PRO(grama doras)

¿Por qué ocupar sort?

Casi nunca es una buena idea usar un algoritmo hecho por uno mismo en una competencia, porque hay buenas implementaciones disponibles en los lenguajes de programación. (...) La librería estándar de C++ contiene la función sort, que puede ser usada fácilmente para ordenar arreglos y otras estructuras de datos..

Antti Laaksonen
Competitive Programmer's Handbook

Función Sort

Forma parte de la librería algorithm, por lo que debemos incluir en nuestro código:

```
#include <algorithm>
```

Sirve para ordenar **vectores** y tiene la siguiente estructura:

Ejemplo 1: `sort(mivector.begin(), mivector.end())`

Esta función no retorna nada porque el mismo vector de entrada se ordena.

Sort con vectores

El siguiente código ordena un **vector.**

Después de ordenarlo el vector queda como [2,3,3,4,5,5,8] en el caso **creciente** y [8,5,5,4,3,3,2] en el caso **decreciente.**

```
vector<int> v = {4,2,5,3,5,8,3};
// En orden creciente
sort(v.begin(),v.end());
// En orden decreciente
sort(v.rbegin(),v.rend());
```

Sort con arreglos

El siguiente código ordena un **arreglo** de tamaño 7.

```
int n = 7; // tamaño del arreglo
int t[] = {4,2,5,3,5,8,3};
sort(t,t+n);
```

Sort con strings

El siguiente código ordena el **string** "monkey".

En este caso, el string s se modifica y queda como "**ekmnoy**".

```
string s = "monkey";
sort(s.begin(), s.end());
```

Función Sort

Forma parte de la librería algorithm, por lo que debemos incluir en nuestro código:

```
#include <algorithm>
```

Sirve para ordenar **vectores** y tiene la siguiente estructura:

Ejemplo 2: `sort(mivector.begin(), mivector.end(), miFuncionComparacion)`

Esta función no retorna nada porque el mismo vector de entrada se ordena. **El vector se ordena según la función de comparación entregada.**

Sort con vectores

El siguiente código ordena un **vector.**

Después de ordenarlo el vector queda como [2,3,3,4,5,5,8] en el caso **creciente** y [8,5,5,4,3,3,2] en el caso **decreciente.**

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

// Función de comparación
bool miOrden(int i, int j) { return i > j; }

int main()
{
    vector<int> mivec = {56, 32, -43, 23, 12, 93, -132};

    // Ordenar con la función especial
    sort(mivec.begin(), mivec.end(), miOrden);

    return 0;
}
```

Actividad

Indica cómo sería el código para ordenar un **arreglo** usando la función **sort** con una función de comparación que ordene de mayor a menor.

Niñas PRO(grama doras)

Actividad

Indica cómo sería el código para ordenar un **arreglo** usando la función **sort** con una función de comparación que ordene de mayor a menor.

```
#include <iostream>
#include <algorithm>

using namespace std;

// Función de comparación
bool miOrden(int i, int j) { return i > j; }

int main()
{
      int n = 5; // tamaño del arreglo
      int t[] = {3,9,1,7,5};

      // Ordenar con la función especial
      sort(t,t+n, miOrden);

      return 0;
}
```

Niñas PRO(grama doras)

¿Qué veremos la próxima clase?

- ★ Semana de Desafío

Línea Gráfica

Lorena Gonzalez - Diseñadora
@soygonzaleztambien