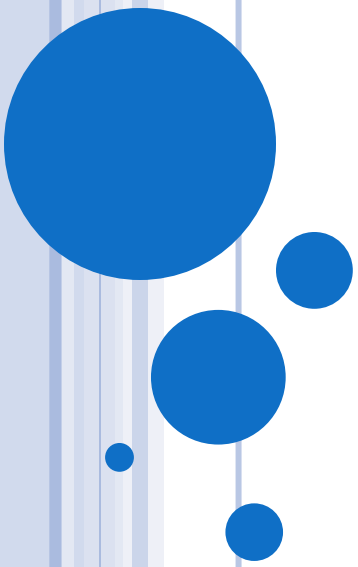
Estructura de Datos

Métodos de Ordenamiento



Métodos de ordenamiento

Muchas veces es necesario además de buscar elementos dentro de en un vector, ordenarlos. Este ordenamiento puede ser de mayor a menor si estamos manejando números y en orden alfabético si se trata de nombres o caracteres.

Existe una gran variedad de métodos de ordenamiento que nos permiten organizar con rapidez los elementos que se encuentran dentro de un vector o archivo. La elección de un determinado método de ordenamiento depende de el tamaño del vector que se desea ordenar.

METODOS ITERATIVOS

Estos métodos son simples de entender y de programar ya que son iterativos, simples ciclos y sentencias que hacen que el vector pueda ser ordenado.

Dentro de los Algoritmos iterativos encontramos:

- Burbuja
- Inserción
- Selección
- Shellsort

METODOS RECURSIVOS

Estos métodos son aún mas complejos, requieren de mayor atención y conocimiento para ser entendidos. Son rápidos y efectivos, utilizan generalmente la técnica Divide y vencerás, que consiste en dividir un problema grande en varios pequeños para que sea más fácil resolverlos.
Mediante llamadas recursivas a si mismos, es posible que el tiempo de ejecución y de ordenación sea más optimo.

Dentro de los algoritmos recursivos encontramos:

- Ordenamiento por Mezclas (merge)
- Ordenamiento Rápido (quick)

COMPLEJIDAD ALGORITMICA

•La complejidad algorítmica representa la cantidad de recursos(temporales) que necesita un algoritmo para resolver un problema y por tanto permite determinar la eficiencia de dicho algoritmo.

• Un algoritmo será mas eficiente comparado con otro, siempre que consuma menos recursos, como el tiempo y espacio de memoria necesarios para ejecutarlo.

•Los criterios que se van a emplear para evaluar la complejidad algorítmica no proporcionan medidas absolutas sino medidas relativas al tamaño del problema.

O(1): Complejidad constante. Cuando las instrucciones se ejecutan una vez.

O(log n): Complejidad logarítmica. Esta suele aparecer en determinados algoritmos con iteración o recursión no estructural, ejemplo la búsqueda binaria.

O(n): Complejidad lineal. Es una complejidad buena y también muy usual. Aparece en la evaluación de bucles simples siempre que la complejidad de las instrucciones interiores sea constante.

O(n log n): Complejidad cuasi-lineal. Se encuentra en algoritmos de tipo divide y vencerás como por ejemplo en el método de ordenación quicksort y se considera una buena complejidad. Si n se duplica, el tiempo de ejecución es ligeramente mayor del doble.

$O(n^2)$: Complejidad cuadrática. Aparece en bucles o ciclos doblemente anidados. Si n se duplica, el tiempo de ejecución aumenta cuatro veces.

$O(n^3)$: Complejidad cúbica. Suele darse en bucles con triple anidación. Si n se duplica, el tiempo de ejecución se multiplica por ocho. Para un valor grande de n empieza a crecer dramáticamente.

$O(n^a)$: Complejidad polinómica (a > 3). Si a crece, la complejidad del programa es bastante mala.

$O(2^n)$: Complejidad exponencial. No suelen ser muy útiles en la práctica por el elevadísimo tiempo de ejecución. Se dan en subprogramas recursivos que contengan dos o más llamadas internas.

$O(n!)$: Complejidad factorial. La peor, problemas generalmente intratables.