



Algoritmos y Programación II

Algoritmos de Ordenamiento

Semestre 2022-15 / Noviembre – 2021



There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

(C. A. R. Hoare)

CONTENIDO PROGRAMATICO:

1. Estructuras Dinámicas de Información
2. Recursión
- 3. Algoritmos de Búsqueda y Ordenamiento**
4. Tipos Abstractos de Datos
5. Tipos de Datos de Secuencia
6. Tipo de Dato Árbol

Algoritmos de Búsqueda y Ordenamiento

- ✓ Algoritmos de búsqueda: Búsqueda lineal, Búsqueda lineal con centinela, Búsqueda binaria.
- ✓ Consideraciones sobre los algoritmos de búsqueda.
- ✓ Algoritmos de ordenamiento: Ordenamiento por intercambio o método de la burbuja, ordenamiento por selección, ordenamiento por inserción., ordenamiento rápido (Quicksort), intercalación (Mergesort).
- ✓ Consideraciones sobre los algoritmos de ordenamiento.
- ✓ Orden de Complejidad de los algoritmos.

Ordenamiento

La ordenación es un proceso de organización de un conjunto de elementos similares de acuerdo a orden lineal, ya sea creciente o decreciente. En esta parte no centraremos en algoritmos que ordenan elementos con acceso directo como lo son los arreglos.



Los algoritmos de ordenación ha sido profundamente analizados, pero desgraciadamente debido a que se comprenden tan bien se considera algo obvio. Es importante conocer bien los diferentes enfoques de ordenación ya que tienen características diferentes, si bien es cierto que algunos algoritmos son mejores en promedio que otros, ninguno es perfecto para todos los casos. Es por eso que se deben estudiar una amplia variedad de algoritmos.

Ordenamiento

Clases de algoritmos

Algoritmos generales $O(N^2)$:

- **Intercambio** involucra repetidas comparaciones y si es necesario intercambio de elementos que no estén en orden, hasta que todos los elementos estén ordenados.
- **Selección** selecciona el elemento del menor valor y lo intercambia por el primero, luego entre los restante $n-1$ y lo intercambia con el segundo y así sucesivamente hasta llegar a los últimos elementos.
- **Insertión** inicialmente ordena los dos primeros elementos, después inserta el tercer en su posición correcta luego el cuarto y así sucesivamente hasta que no haya elementos por insertar.

Ordenamiento

Clases de algoritmos

Algoritmos mejorados:

Todos los algoritmos mencionados anteriormente tiene un tiempo de ejecución $O(n^2)$, lo que puede hacer que la ordenación para grandes volumen de datos se muy lenta.

La solución puede ser utilizar algoritmos de ordenación “mejor”, entre los que estudiaremos:

- Mergesort
- Shellsort
- Quicksort

Ordenamiento

Algoritmos por Intercambio

Método de Burbuja

Quizás el método de ordenamiento más simple, es el ordenamiento de burbuja (bubblesort). La idea básica consiste en imaginarse los elementos (o registros) como burbujas, los que tienen clave menores son más ligeros y suben, se recorre varias veces haciendo que los elementos adyacentes que no estén en orden se inviertan. El efecto producido por esta operación es que en el primer recorrido el elemento “más ligero de todos” es decir con la menor de las claves suba a la superficie, en el segundo recorrido el segundo elemento más ligero sube al segundo lugar, y así sucesivamente.

Ordenamiento

Algoritmos por Intercambio

Método de Burbuja

```
void burbuja(int array[], int size) {  
    int i, j;  
  
    for(i=1; i<size; i++)  
        for(j=size-1; j>=i; --j)  
            if(array[j-1]>array[j]) {  
                int t; // intercambio  
                t=array[j-1];  
                array[j-1]=array[j];  
                array[j]=t;  
            }  
}
```

Ordenamiento

Algoritmos por Intercambio

Método de Burbuja

Burbuja

Original: 25 35 5 50 20 15 40 10 30 45

Paso 1 : 5 25 35 10 50 20 15 40 30 45

Paso 2 : 5 10 25 35 15 50 20 30 40 45

Paso 3 : 5 10 15 25 35 20 50 30 40 45

Paso 4 : 5 10 15 20 25 35 30 50 40 45

Paso 5 : 5 10 15 20 25 30 35 40 50 45

Paso 6 : 5 10 15 20 25 30 35 40 45 50

Paso 7 : 5 10 15 20 25 30 35 40 45 50

Paso 8 : 5 10 15 20 25 30 35 40 45 50

Paso 9 : 5 10 15 20 25 30 35 40 45 50

Comparaciones:45 - Intercambios:20

Ordenamiento

Algoritmos por Intercambio

Método de Burbuja1

Los elementos más grandes burbujan hacia el final

```
void burbuja1(int array[], int size) {  
    int i, j;  
  
    for (i=0; i<size-1; i++)  
        for (j=0; j<size-i-1; j++)  
            if (array[j]>array[j+1]) {  
                int t; // intercambiar  
                t=array[j];  
                array[j]=array[j+1];  
                array[j+1]=t;  
            }  
}
```

Ordenamiento

Algoritmos por Intercambio

Método de Burbuja1

Los elementos más grandes burbujan hacia el final

Burbuja1

Original: 25 35 5 50 20 15 40 10 30 45

Paso 1 : 25 5 35 20 15 40 10 30 45 50

Paso 2 : 5 25 20 15 35 10 30 40 45 50

Paso 3 : 5 20 15 25 10 30 35 40 45 50

Paso 4 : 5 15 20 10 25 30 35 40 45 50

Paso 5 : 5 15 10 20 25 30 35 40 45 50

Paso 6 : 5 10 15 20 25 30 35 40 45 50

Paso 7 : 5 10 15 20 25 30 35 40 45 50

Paso 8 : 5 10 15 20 25 30 35 40 45 50

Paso 9 : 5 10 15 20 25 30 35 40 45 50

Comparaciones:45 - Intercambios:20

Ordenamiento

Algoritmos por Intercambio

Método de Burbuja2

burbuja aprovechando el final ya ordenado

```
void burbuja2(int array[], int size) { //Burbuja mejorado
    int i, j, f=1;

    for(i=0; i<size-1 && f; i++){
        f=0;
        for(j=0; j<size-i-1; j++){
            if(array[j]>array[j+1]){
                int t;
                t=array[j];
                array[j]=array[j+1];
                array[j+1]=t;
                f=1;
            }
        }
    }
}
```

Ordenamiento

Algoritmos por Intercambio

Método de Burbuja2

burbuja aprovechando el final ya ordenado

Burbuja2

Original: 25 35 5 50 20 15 40 10 30 45

Paso 1 : 25 5 35 20 15 40 10 30 45 50

Paso 2 : 5 25 20 15 35 10 30 40 45 50

Paso 3 : 5 20 15 25 10 30 35 40 45 50

Paso 4 : 5 15 20 10 25 30 35 40 45 50

Paso 5 : 5 15 10 20 25 30 35 40 45 50

Paso 6 : 5 10 15 20 25 30 35 40 45 50

Paso 7 : 5 10 15 20 25 30 35 40 45 50



Comparaciones:42 - Intercambios:20



Ordenamiento

Algoritmos por Intercambio

Método de Intercambio

Se comparan elementos que no son adyacentes

```
void intercambio(int array[], int size) {  
    int i, j;  
  
    for(i=0; i<size-1; i++)  
        for(j=i+1; j<size; j++)  
            if(array[i]>array[j]){  
                int t; // intercambiar  
                t=array[i];  
                array[i]=array[j];  
                array[j]=t;  
            }  
}
```

Ordenamiento

Algoritmos por Intercambio

Método de Intercambio

Se comparan elementos que no son adyacentes

Intercambio

Original: 25 35 5 50 20 15 40 10 30 45



Paso 1 : 5 35 25 50 20 15 40 10 30 45

Paso 2 : 5 10 35 50 25 20 40 15 30 45

Paso 3 : 5 10 15 50 35 25 40 20 30 45

Paso 4 : 5 10 15 20 50 35 40 25 30 45

Paso 5 : 5 10 15 20 25 50 40 35 30 45

Paso 6 : 5 10 15 20 25 30 50 40 35 45

Paso 7 : 5 10 15 20 25 30 35 50 40 45

Paso 8 : 5 10 15 20 25 30 35 40 50 45

Paso 9 : 5 10 15 20 25 30 35 40 45 50

Comparaciones:45 - Intercambios:20



Ordenamiento

Método por Selección



En la i -ésima posición se coloca el menor entre los $n-i$ restantes.

```
void seleccion(int array[], int size) {  
    int e, i, j;  
  
    for(i=0; i<size-1; i++) {  
        e=i;  
        for(j=i+1; j<size; j++)  
            if(array[e]>array[j])  
                e=j;  
        if(i!=e) {  
            int t; // intercambiar  
            t=array[i];  
            array[i]=array[e];  
            array[e]=t;  
        }  
    }  
}
```

Ordenamiento

Método por Selección

En la i -ésima posición se coloca el menor entre los $n-i$ restantes.

Selección											
Original:		25	35	5	50	20	15	40	10	30	45
											
Paso	1 :	5	35	25	50	20	15	40	10	30	45
Paso	2 :	5	10	25	50	20	15	40	35	30	45
Paso	3 :	5	10	15	50	20	25	40	35	30	45
Paso	4 :	5	10	15	20	50	25	40	35	30	45
Paso	5 :	5	10	15	20	25	50	40	35	30	45
Paso	6 :	5	10	15	20	25	30	40	35	50	45
Paso	7 :	5	10	15	20	25	30	35	40	50	45
Paso	8 :	5	10	15	20	25	30	35	40	50	45
Paso	9 :	5	10	15	20	25	30	35	40	45	50

Comparaciones:45 - Intercambios:8

Ordenamiento

Método por Inserción Directa

Se inserta el i -ésimo elemento en la posición correcta en los $i-1$ anteriores

```
void insercionDirecta(int array[], int size) {  
    int i, j;  
  
    for (i=1; i<size; i++) {  
        for (j=i; j>0 && array[j]<array[j-1]; j--) {  
            int t; // Intercambiar  
            t=array[j];  
            array[j]=array[j-1];  
            array[j-1]=t;  
        }  
    }  
}
```

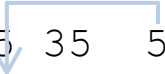
Ordenamiento

Método por Inserción Directa

Se inserta el i -ésimo elemento en la posición correcta en los $i-1$ anteriores

Insercion Directa

Original: 25 35 5 50 20 15 40 10 30 45

Paso 1 :  25 35 5 50 20 15 40 10 30 45

Paso 2 : 5 25 35 50 20 15 40 10 30 45

Paso 3 :  5 25 35 50 20 15 40 10 30 45

Paso 4 : 5 20 25 35 50 15 40 10 30 45

Paso 5 : 5 15 20 25 35 50 40 10 30 45

Paso 6 : 5 15 20 25 35 40 50 10 30 45

Paso 7 : 5 10 15 20 25 35 40 50 30 45

Paso 8 : 5 10 15 20 25 30 35 40 50 45

Paso 9 : 5 10 15 20 25 30 35 40 45 50

Comparaciones:29 - Intercambios:20

Ordenamiento

Método por Inserción

Se inserta el i -ésimo elemento en la posición correcta en los $i-1$ anteriores

```
void insercion(int array[], int size) {  
    int t, i, j;  
  
    for(i=1; i<size; i++) {  
        t=array[i];  
        for(j=i; j>0 && t<array[j-1]; j--)  
            array[j]=array[j-1];  
        array[j]=t;  
    }  
}
```

Ordenamiento

Método por Inserción

Se inserta el i-ésimo elemento en la posición correcta en los i-1 anteriores

Insercion

Original: 25 35 5 50 20 15 40 10 30 45

Paso	1	:	25	35	5	50	20	15	40	10	30	45
Paso	2	:	5	25	35	50	20	15	40	10	30	45
Paso	3	:	5	25	35	50	20	15	40	10	30	45
Paso	4	:	5	20	25	35	50	15	40	10	30	45
Paso	5	:	5	15	20	25	35	50	40	10	30	45
Paso	6	:	5	15	20	25	35	40	50	10	30	45
Paso	7	:	5	10	15	20	25	35	40	50	30	45
Paso	8	:	5	10	15	20	25	30	35	40	50	45
Paso	9	:	5	10	15	20	25	30	35	40	45	50

Comparaciones:29 - Asignaciones:29

Ordenamiento

Comparación de los Métodos Simples

Ordenar un arreglo ordenado de forma decreciente

Original: 50 45 40 35 30 25 20 15 10 5

Burbuja

Comparaciones:45 - Intercambios:45

Burbuja1

Comparaciones:45 - Intercambios:45

Burbuja2

Comparaciones:45 - Intercambios:45

Intercambio

Comparaciones:45 - Intercambios:45

Selección

Comparaciones:45 - Intercambios: 5

Inserción Directa

Comparaciones:54 - Intercambios:45

Inserción

Comparaciones:54 - Asignaciones:54

Ordenamiento

Comparación de los Métodos Simples

Ordenar un arreglo ordenado de forma creciente

Original: 5 10 15 20 25 30 35 40 45 50

Burbuja

Comparaciones: 45 - Intercambios: 0

Burbuja1

Comparaciones: 45 - Intercambios: 0

Burbuja2

Comparaciones: 9 - Intercambios: 0

Intercambio

Comparaciones: 45 - Intercambios: 0

Selección

Comparaciones: 45 - Intercambios: 0

Inserción Directa

Comparaciones: 9 - Intercambios: 0

Inserción

Comparaciones: 9 - Asignaciones: 9

Ordenamiento

Comparación de los Métodos Simples

Ordenar un arreglo con el mismo elemento

Original: 10 10 10 10 10 10 10 10 10 10 10

Burbuja

Comparaciones:45 - Intercambios: 0

Burbuja1

Comparaciones:45 - Intercambios: 0

Burbuja2

Comparaciones: 9 - Intercambios: 0

Intercambio

Comparaciones:45 - Intercambios: 0

Selección

Comparaciones:45 - Intercambios: 0

Inserción Directa

Comparaciones: 9 - Intercambios: 0

Inserción

Comparaciones: 9 - Asignaciones: 9

Ordenamiento

Ordenación Shell

Este método adquiere de su inventor D. L. Shell, es un método derivado de la ordenación por inserción, y se basa en incrementos decrecientes. Por ejemplo se puede ordenar todos los elementos separados por tres posiciones, luego los separados por dos y finalmente los adyacentes.

Cada paso de la ordenación involucra relativamente a pocos elementos o elementos que están en un orden razonable, por lo que cada paso aumenta el orden de los elementos.

La secuencia de paso puede cambiar y la única regla es que el ultimo paso sea 1.

Ordenamiento

Ordenación Shell

Disminución de incrementos, con intercambios

```
void shellsort(int array[], int size) {  
    int gap, i, j;  
  
    for (gap = size / 2; gap > 0; gap /= 2)  
        for (i = gap; i < size; i++)  
            for (j = i - gap; j >= 0 && array[j + gap] < array[j]; j -= gap) {  
                int t; // Intercambiar  
  
                t = array[j];  
                array[j] = array[j + gap];  
                array[j + gap] = t;  
            }  
}
```

Ordenamiento

Ordenación Shell

ShellSort											
Original:											
Gap 5 Paso 5:											
Gap 5 Paso 6:											
Gap 5 Paso 7:											
Gap 5 Paso 8:											
Gap 5 Paso 9:											
Gap 2 Paso 2:											
Gap 2 Paso 3:											
Gap 2 Paso 4:											
Gap 2 Paso 5:											
Gap 2 Paso 6:											
Gap 2 Paso 7:											
Gap 2 Paso 8:											
Gap 2 Paso 9:											
Gap 1 Paso 1:											
Gap 1 Paso 2:											
Gap 1 Paso 3:											
Gap 1 Paso 4:											
Gap 1 Paso 5:											
Gap 1 Paso 6:											
Gap 1 Paso 7:											
Gap 1 Paso 8:											
Gap 1 Paso 9:											
Ordenado:											
Comparaciones:28											
Intercambios:12											

Ordenamiento

Ordenación Shell

Disminución de incrementos, con corrimientos

```
void shellSort(int array[], int size) {  
    int gap, i, j, t;  
  
    for (gap=size/2; gap>0; gap/=2)  
        for (i=gap; i<size; i++) {  
            t=array[i];  
            for (j=i-gap; j>=0 && t<array[j]; j-=gap)  
                array[j+gap]=array[j];  
            array[j+gap]=t;  
        }  
}
```

Ordenamiento

Ordenación Shell

Disminución de incrementos,
con corrimientos

ShellSort (M)

Original: 25 35 5 50 20 15 40 10 30 45

Gap 5 Paso 5: 25 35 5 50 20 15 40 10 30 45

Gap 5 Paso 6: 15 35 5 50 20 25 40 10 30 45

Gap 5 Paso 7: 15 35 5 50 20 25 40 10 30 45

Gap 5 Paso 8: 15 35 5 50 20 25 40 10 30 45

Gap 5 Paso 9: 15 35 5 30 20 25 40 10 50 45

Gap 2 Paso 2: 15 35 5 30 20 25 40 10 50 45

Gap 2 Paso 3: 5 35 15 30 20 25 40 10 50 45

Gap 2 Paso 4: 5 30 15 35 20 25 40 10 50 45

Gap 2 Paso 5: 5 30 15 35 20 25 40 10 50 45

Gap 2 Paso 6: 5 25 15 30 20 35 40 10 50 45

Gap 2 Paso 7: 5 25 15 30 20 35 40 10 50 45

Gap 2 Paso 8: 5 10 15 25 20 30 40 35 50 45

Gap 2 Paso 9: 5 10 15 25 20 30 40 35 50 45

Gap 1 Paso 1: 5 10 15 25 20 30 40 35 50 45

Gap 1 Paso 2: 5 10 15 25 20 30 40 35 50 45

Gap 1 Paso 3: 5 10 15 25 20 30 40 35 50 45

Gap 1 Paso 4: 5 10 15 25 20 30 40 35 50 45

Gap 1 Paso 5: 5 10 15 20 25 30 40 35 50 45

Gap 1 Paso 6: 5 10 15 20 25 30 40 35 50 45

Gap 1 Paso 7: 5 10 15 20 25 30 40 35 50 45

Gap 1 Paso 8: 5 10 15 20 25 30 35 40 50 45

Gap 1 Paso 9: 5 10 15 20 25 30 35 40 50 45

Ordenado: 5 10 15 20 25 30 35 40 45 50

Comparaciones:28 - Asignaciones:34

21 +13

Ordenamiento

Ordenación por mezcla

Emplea la estrategia de divide y vencerás, se divide el arreglo en dos partes y estos son ordenados (utilizando el algoritmo recursivamente) y luego ambas partes son mezclados para producir uno ordenado.

```
void mezclar(int array[], int inicio, int medio, int fin) {  
    int i=inicio, j=medio+1, k=0, temp[fin-inicio];  
  
    while (i<=medio && j<=fin)  
        temp[k++]=(array[i]<array[j]?array[i++]:array[j++]);  
    while (i<=medio)  
        temp[k++]=array[i++];  
    while (j<=fin)  
        temp[k++]=array[j++];  
    for (i=inicio, k=0; i<=fin; i++, k++)  
        array[i]=temp[k];  
}
```

Ordenamiento

Ordenación por mezcla

```
void ordenarMezcla(int array[], int inicio, int fin) {  
    int central;  
  
    if(inicio >= fin)  
        return;  
    central = (inicio + fin) / 2;  
    ordenarMezcla(array, inicio, central);  
    ordenarMezcla(array, central + 1, fin);  
    mezclar(array, inicio, central, fin);  
}
```


Ordenamiento

Ordenación por mezcla

	Original:	25	35	5	50	20	15	40	10	30	45
inicio:0 medio:0 fin:1	-	25	35	5	50	20	15	40	10	30	45
inicio:0 medio:1 fin:2	-	5	25	35	50	20	15	40	10	30	45
inicio:3 medio:3 fin:4	-	5	25	35	20	50	15	40	10	30	45
inicio:0 medio:2 fin:4	-	5	20	25	35	50	15	40	10	30	45
inicio:5 medio:5 fin:6	-	5	20	25	35	50	15	40	10	30	45
inicio:5 medio:6 fin:7	-	5	20	25	35	50	10	15	40	30	45
inicio:8 medio:8 fin:9	-	5	20	25	35	50	10	15	40	30	45
inicio:5 medio:7 fin:9	-	5	20	25	35	50	10	15	30	40	45
inicio:0 medio:4 fin:9	-	5	10	15	20	25	30	35	40	45	50

Comparaciones: 23

Ordenamiento

Ordenación rápida (quicksort)

Inventado por C. A. R. Hoare, basado en la ordenación por intercambio, es considerado el mejor algoritmo de ordenación disponible.

Se basa en la idea de particiones, para lo cual se selecciona un elemento v (pivote) alrededor del cual se organizan los elementos restantes del arreglo, se permutan los elementos del tal forma que los elementos menores o iguales que v estén en $A[0], \dots, A[i]$ y todos aquellos con clave mayor estén en $A[i+1], \dots, A[n]$. Luego se aplica recursivamente el algoritmo para clasificar ambos grupos, esto se repite hasta que el arreglo queda ordenado.

Ordenamiento

Ordenación rápida (quicksort)

Función para intercambiar dos elementos del arreglo.

```
void swap(int array[], int a, int b) {  
    int t;  
  
    t=array[b];  
    array[b]=array[a];  
    array[a]=t;  
}
```

Ordenamiento

Ordenación rápida (quicksort)

Función para dividir el arreglo.

```
int particion(int array[], int inicio, int fin){
    int pivote, inferior=inicio+1, superior=fin;
    pivote=array[inicio];
    do{
        while(array[inferior]<=pivote && inferior<=superior)
            inferior++;
        while(array[superior]>pivote && inferior<=superior)
            superior--;
        if(inferior <= superior){
            swap(array,inferior,superior);
            inferior++;
            superior--;
        }
    }while(inferior<=superior);
    swap(array,inicio,superior);
    return superior;
}
```

Ordenamiento

Ordenación rápida (quicksort)

```
void quicksort(int array[], int inicio, int fin) {  
    int p;  
  
    if (inicio < fin) {  
        p = partition(array, inicio, fin);  
        quicksort(array, inicio, p - 1);  
        quicksort(array, p + 1, fin);  
    }  
}
```

Ordenamiento

Ordenación rápida (quicksort)

```

Original: 25 35 5 50 20 15 40 10 30 45
i:0 f:9 p:4 - 20 10 5 15 25 50 40 35 30 45
i:0 f:3 p:3 - 15 10 5 20 25 50 40 35 30 45
i:0 f:2 p:2 - 5 10 15 20 25 50 40 35 30 45
i:0 f:1 p:0 - 5 10 15 20 25 50 40 35 30 45
i:5 f:9 p:9 - 5 10 15 20 25 45 40 35 30 50
i:5 f:8 p:8 - 5 10 15 20 25 30 40 35 45 50
i:5 f:7 p:5 - 5 10 15 20 25 30 40 35 45 50
i:6 f:7 p:7 - 5 10 15 20 25 30 35 40 45 50
Ordenado: 5 10 15 20 25 30 35 40 45 50

```

Comparaciones:41 - Intercambios:10

Ordenamiento

Ordenación rápida (quicksort)

Otra implementación con pivote elemento central

```
void quickSort(int array[], int left, int right) {  
    int i, last;  
  
    if (left >= right)  
        return;  
    swap(array, left, (left + right) / 2);  
    last = left;  
    for (i = left + 1; i <= right; i++)  
        if (array[i] < array[left])  
            swap(array, ++last, i);  
    swap(array, left, last);  
    quickSort(array, left, last - 1);  
    quickSort(array, last + 1, right);  
}
```

Ordenamiento

Ordenación rápida (quicksort)

Original:	25	35	5	50	20	15	40	10	30	45
l:0 r:9 p:3 -	10	5	15	20	25	35	40	50	30	45
l:0 r:2 p:0 -	5	10	15	20	25	35	40	50	30	45
l:1 r:2 p:1 -	5	10	15	20	25	35	40	50	30	45
l:4 r:9 p:7 -	5	10	15	20	30	35	25	40	50	45
l:4 r:6 p:6 -	5	10	15	20	25	30	35	40	50	45
l:4 r:5 p:4 -	5	10	15	20	25	30	35	40	50	45
l:8 r:9 p:9 -	5	10	15	20	25	30	35	40	45	50
Ordenado:	5	10	15	20	25	30	35	40	45	50

Comparaciones:21 - Intercambios:23

Consideraciones sobre los algoritmos de ordenamiento

Al evaluar un algoritmo de ordenamiento hay que tener en cuenta :

- ¿Cómo es su comportamiento en el caso medio?
- ¿Cómo es su comportamiento en el mejor y peor caso?
- ¿Cómo es su comportamiento si la lista está ordenada?
- ¿Cómo es su comportamiento si la lista está en el orden inverso?
- ¿Intercambia o reorganiza elementos iguales?

Por lo anterior se debe tener a mano una selección de algoritmos de ordenamiento, aunque la ordenación rápida (Quicksort) es la mejor para el caso medio, no lo es para todos los casos, por ejemplo hay que tomar en cuenta la forma como están ordenados los elementos y costo de la recursión, o en casos de otros métodos si requieren espacio temporal.

Consideraciones sobre los algoritmos de ordenamiento

Se ha revisados diferentes métodos, algunos de los cuales tiene mejoras u optimizaciones. El orden, nos puede indicar su complejidad y sirve de base para comparar su eficiencia. Los métodos generales (Intercambio, selección e Inserción) tienen complejidad de $O(n^2)$ mientras el de mezcla tiene $O(n \log n)$, el Shell tiene un orden de $O(n^{3/2})$ y la ordenación rápida en promedio $O(n \log n)$ y en peor caso es $O(n^2)$.

N	N^2	$N^{3/2}$	$N \log N$
10	100	32	33
100	10.000	1.000	664
1000	1.000.000	31.623	9.966
10000	100.000.000	1.000.000	132.877
100000	10.000.000.000	31.622.777	1.660.964
1000000	1.000.000.000.000	1.000.000.000	19.931.569

