Universidad Autónoma del Estado de México

Centro Universitario UAEM Texcoco

Departamento de Ciencias Aplicadas.

Ingeniería en Computación.

Programación avanzada.

Unidad de competencia II: "Análisis de algoritmos de ordenamiento y búsqueda"

Presenta:

M. en C. C. J. Jair Vázquez Palma.

Programación avanzada

Objetivos de la Unidad de la Unidad de Aprendizaje

- Analizar y diseñar sistemas de información.
- Utilizar eficazmente los lenguajes de programación.
- Responder eficazmente a nuevas situaciones informáticas.
- Analizar soluciones del entorno y problemas propios de ser tratados mediante sistemas computacionales.
- Aplicar los conocimientos en la práctica.
- Desarrollar la habilidad análisis y síntesis de información.

Unidad II: Análisis de algoritmos de ordenamiento y búsqueda.

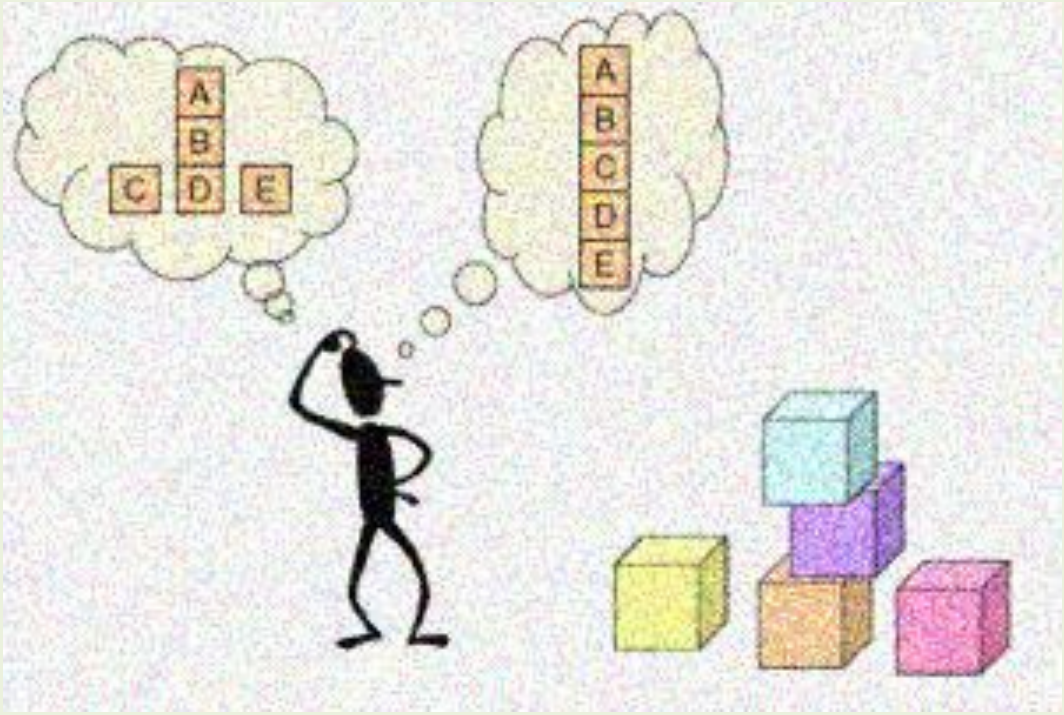
Contenido:

- Orden de complejidad de un algoritmo.
- Método de la burbuja.
- Método de selección.
- Método de inserción.
- Método de ordenamiento rápido.
- Método de mezclas.
- Búsqueda secuencial.
- Búsqueda binaria.

Objetivos de la Unidad de Competencia II

- Conocer el funcionamiento de un método de ordenamiento y búsqueda.
- Obtener el orden de complejidad de un algoritmo de ordenamiento y búsqueda.
- Comprender el funcionamiento de la complejidad de los métodos de ordenamiento y búsqueda.

ANÁLISIS DE ALGORITMOS DE ORDENAMIENTO Y BÚSQUEDA



Orden y complejidad algorítmica.

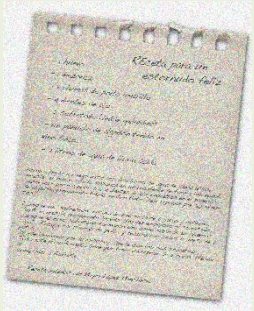
¿Qué es un algoritmo?

En cómputo se define como:

Un método preciso usado por una computadora para la solución de problemas.

Un algoritmo está compuesto de un conjunto finito de pasos, cada paso puede requerir una o más operaciones. Cada operación debe estar definida. Cada paso debe ser tal que deba, al menos, ser hecha por una persona usando lápiz y papel en un tiempo finito.

Orden y complejidad algorítmica.



Existe una gran diferencia entre receta y algoritmo:

- Receta: Colocar sal al gusto.
- Algoritmo: Debe de indicarse cada paso con exactitud.

Otra palabra que obedece a casi todo lo indicado es “proceso computacional”. Un ejemplo es el Sistema Operativo. Este proceso está diseñado para controlar la ejecución de trabajos, en teoría, el proceso nunca termina ya que se queda en estado de espera hasta que la solicitud de otro trabajo llegue.

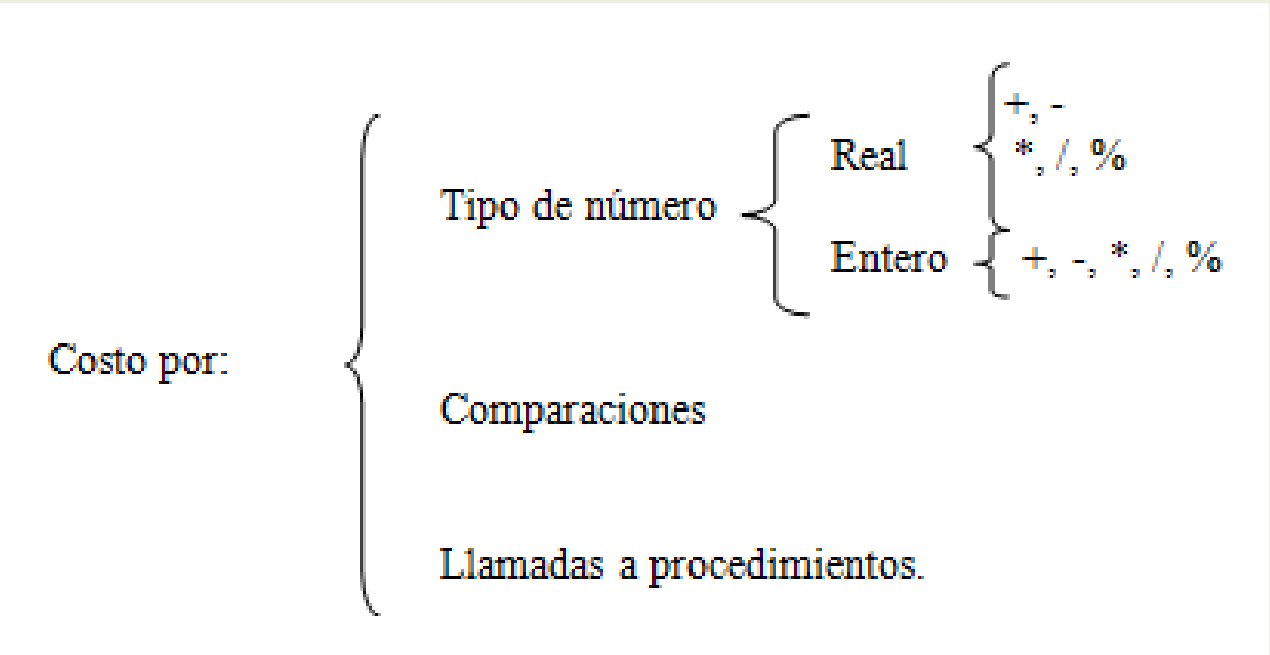
Orden y complejidad algorítmica.

El estudio de algoritmos incluye varias y activas áreas de la investigación:

1. Cómo elaborar algoritmos.
2. Cómo expresarlos.
3. Cómo validarlos.
4. Cómo analizarlos.
5. Cómo probarlos. El debugging sólo nos indica la presencia de errores, no la ausencia de ellos. También se debe de evaluar a un algoritmo en forma temporal como en forma espacial.

Orden y complejidad algorítmica.

Al analizar un algoritmo, lo primero es verificar cuales operaciones son empleadas y su costo.



Orden y complejidad algorítmica.

- Estas operaciones se pueden acotar en un tiempo constante. Ejemplo, la comparación entre caracteres se puede hacer en un tiempo fijo.
- La comparación de las cadenas depende del tamaño de las cadenas. (No se puede acotar el tiempo).
- Para ver los tiempos de un algoritmo se requiere un análisis a priori y no a posteriori.
- En un ***análisis a priori*** se obtiene una función que acota el tiempo del algoritmo.
- En un ***análisis a posteriori*** se colecta una estadística sobre el desarrollo del algoritmo en tiempo y espacio al momento de la ejecución de tal algoritmo.

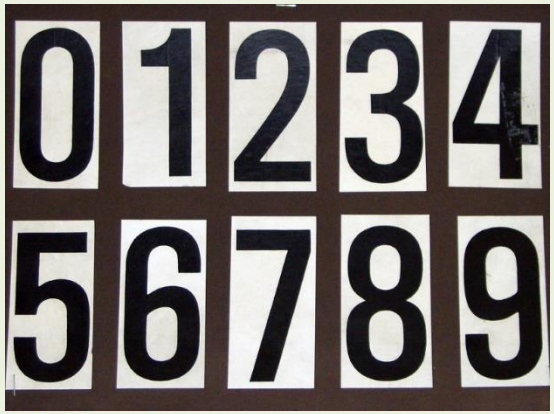
Orden y complejidad algorítmica.

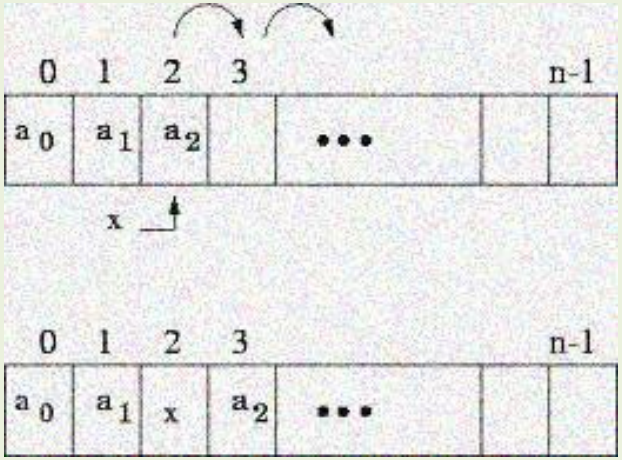
Un ejemplo de un análisis a priori es:

x ← x+y	para i ← 1 a n x ← x+y	para i ← 1 a n para j ←1 a n x ← x+y
1	**n**	**n^2**

El análisis a priori ignora qué tipo de máquina, el lenguaje, y sólo se concentra en determinar el orden de magnitud de la frecuencia de ejecución.

Ordenación y búsqueda

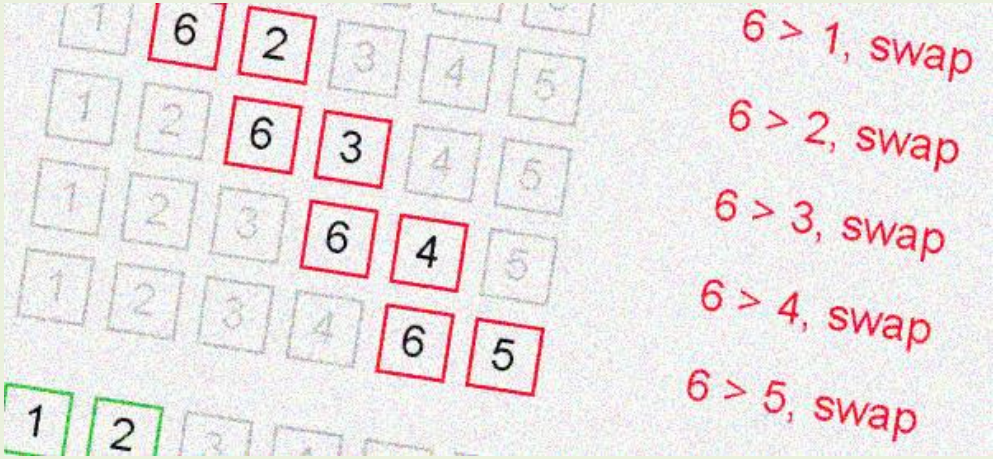




Ordenación y búsqueda.

Burbuja.

- Es el algoritmo más sencillo. Ideal para empezar. Consiste en ciclar repetidamente a través de una lista, comparando elementos adyacentes de dos en dos. Si un elemento es mayor que el que está en la siguiente posición se intercambian.



Algoritmo de la burbuja.

Algoritmo que ordena los elementos de un vector, donde ***A*** es el vector, ***N*** es el numero de elementos del arreglo.

```
{I,J,MENOR y TEMP son variables de tipo entero}
1. Repetir con I desde 1 hasta N-1
     1.1 Repetir con J desde 0 hasta N-1
        1.1.1 si A[J] > A[j+1] entonces
              hacer TEMP ← A[J]
              hacer A[J] ← A[J+1]
             hacer A[j+1] ← TEMP
        1.1.2 { fin de condición del paso 1.1.1}
     1.2 {fin del ciclo del paso 1.1}
2. {fin del ciclo del paso 1}
```

Burbuja.

- El algoritmo en pseudocódigo en C es:

```
for (i=1; i<TAM; i++){
      for (j=0; j<TAM-1; j++){
            if (lista[j]>lista[j+1]){
            temp = lista [j];
            lista[j]=lista[j+1];
            lista [j+1]=temp;}}}
```

Dónde:

lista: Es cualquier lista a ordenar

TAM: Es una constante que determina el tamaño de la lista.

i, j: Contadores

temp: Permite realizar los intercambios de la lista

Burbuja.

Ventajas:

- Fácil implementación.
- No requiere memoria adicional.

Desventajas:

- Muy lento.
- Realiza numerosas comparaciones.
- Realiza numerosos intercambios.

Ordenación por selección directa

La idea básica de éste algoritmo consiste en buscar el menor elemento del arreglo y colocarlo a la primera posición. Luego se busca el segundo elemento más pequeño del arreglo y se coloca en la segunda posición. El proceso continúa hasta que todos los elementos del arreglo hayan sido ordenados.

El método se basa en los siguientes principios:

- Seleccionar el menor elemento del arreglo
- Intercambiar dicho elemento con el primero
- Repetir los pasos anteriores con los (n-1), (n-2) elementos, y así sucesivamente hasta que sólo quede el elemento mayor.

Algoritmo selección directa.

Algoritmo que ordena los elementos de un vector, donde ***A*** es el vector, ***N*** es el numero de elementos del arreglo.

```
{I, MENOR, K y J son variables de tipo entero}
1. Repetir con I desde 1 hasta N-1
     Hacer MENOR ← A[I] Y K←I
     1.1 Repetir con J desde I+1 hasta N
        1.1.1 si A[J] < MENOR entonces
              hacer MENOR ← A[J] y K←J
        1.1.2 { fin de condición del paso 1.1.1}
     1.2 {fin del ciclo del paso 1.1}
     1.3 si MENOR < A[I] entonces
              hacer A[K] ← A[I] y A[I]← MENOR
     1.4 {fin del paso 1.3}
2. {fin del ciclo del paso 1}
```

Ordenación por selección directa.

El algoritmo en código en C es:

```
#include <stdio.h>
main(){
        int MENOR, i, a[7]={10,7,6,4,9,8,1},j , k , m;
        for (i=0;i<6;i++){
                MENOR = a[i];
                for (j=i+1; j<7;j++){
                        if (a[j] <MENOR){
                        MENOR=a[j];
                        k=j;
                }
        }
if (MENOR<a[i]){
        a[k] =a[i];
        a[i] =MENOR;
        }
}
for (i=0; i<7;i++)
        printf ("%d...", a[i]);
getchar ();
}
```

Ordenación por selección directa.

- El análisis del método de selección directa es relativamente simple.
- Se debe considerar el número de comparaciones entre elementos es independiente de la disposición inicial de éstos elementos en el arreglo.
- En la primera pasada se realizan (n-1) comparaciones, en la segunda pasada (n-2) comparaciones y así sucesivamente hasta 2 y 1 comparaciones en la penúltima y última pasadas, respectivamente.

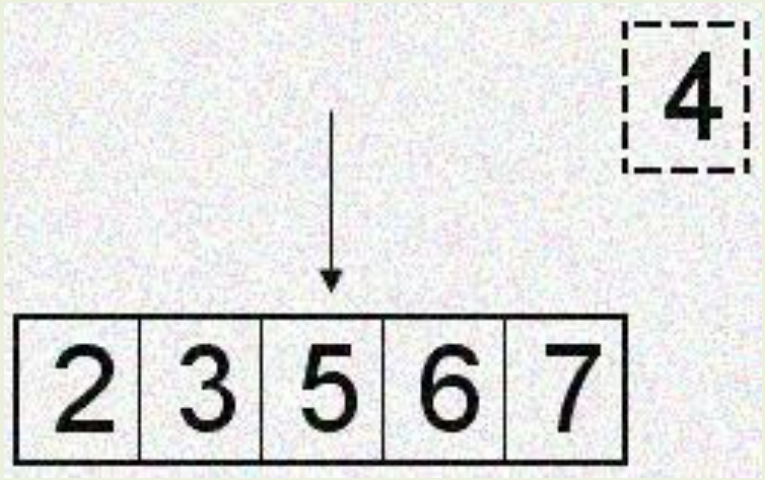
Método de inserción binaria.

El **método por inserción binaria** es una mejora del método de **inserción directa**. La mejora consiste en realizar una búsqueda binaria en lugar de una búsqueda secuencial, para insertar un elemento en la parte izquierda del arreglo, que ya se encuentra ordenado.

El proceso al igual que el de inserción directa, se repite desde el 2do hasta el n-ésimo elemento. Toma su nombre debido a la similitud de ordenamiento de los arboles binarios.

Método de inserción binaria

El método de ordenación por inserción binaria realiza una búsqueda binara en lugar de una búsqueda secuencial, para insertar un elemento en la parte izquierda del arreglo, que ya se encuentra ordenado. El proceso se repite desde el segundo hasta el ***n-ésimo*** elemento.



Método de inserción binaria.

Tengamos en cuenta que:

- La secuencia donde se inserta el nuevo elemento ya esta ordenada.
- Búsqueda binaria para localizar el lugar de inserción.
- Desplazar elementos.
- Insertar.
- Toma su nombre debido a la similitud de ordenamiento de los arboles binarios.

Algoritmo por inserción binaria.

Algoritmo que ordena los elementos de un vector, donde ***A*** es el vector y ***N*** el tamaño del arreglo.

```
{I, AUX, DER,IZQ, M y J son variables de tipo entero}
1. Repetir con I desde 1 hasta N-1
      Hacer AUX ← A[I], IZQ←0 Y DER←I-1
      1.1 Mientras IZQ <= DER
               M ← (IZQ+DER)/2
         1.1.1 si AUX <=A[M] MENOR entonces
               hacer DER ← M-1
         1.1.2 si NO
               hacer IZQ ← M+1
      1.2 {fin del ciclo del paso 1.1}
          1.2.1 hacer J←I-1
      1.3 Mientras J>=IZQ entonces
               hacer A[J+1] ← A[J]
               hacer J--
      1.4 {fin del paso 1.3}
      hacer A[IZQ]←|AUX
2. {fin del ciclo del paso 1}
```

Método de inserción binaria. El algoritmo en código en C es:

```
#include <stdio.h>
main(){
        int a[]={10,8,7,2,1,3,5,4,6,9},i,aux,der,izq,m,j;
        for (i=1;i<10;i++){
                aux=a[i];
                izq=0;
                der=i-1;
        while(izq<=der){
                m=(izq+der)/2;
                if (aux<=a[m])
                        der=m-1;
                else
                        izq=m+1;
                }
                j=i-1;
        while(j>=izq){
                a[j+1]=a[j];
                j--;
                }
        a[izq]=aux;
        }
        for (i=0;i<10;i++)
                printf("%d..",a[i]);
        putchar('\n');
getchar();
}
```

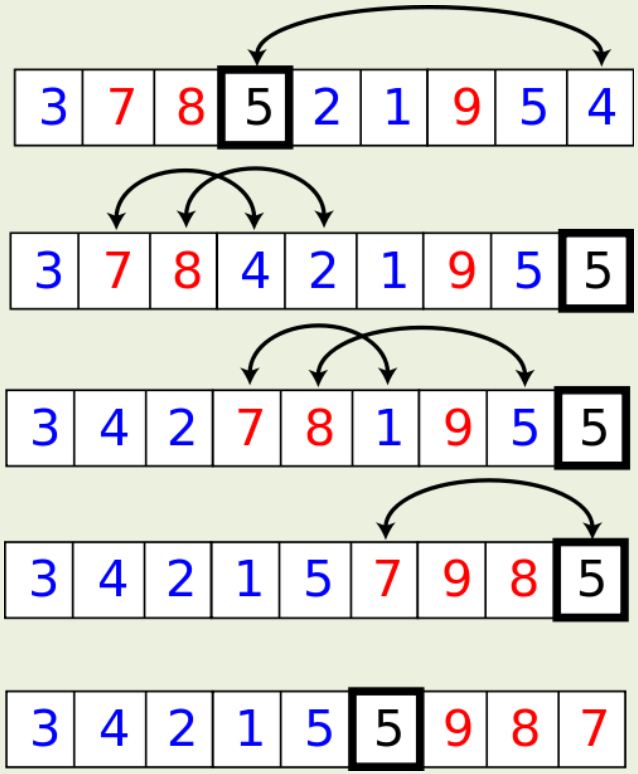
Método de inserción binaria.

- Al analizar el método de ordenación por inserción binaria se advierte la presencia de un caso antinatural. El método efectúa el menor número de comparaciones cuando el arreglo está totalmente desordenado y el máximo cuando se encuentra ordenado.

- Es posible suponer que mientras en una búsqueda secuencial se necesitan K comparaciones para insertar un elemento, en una binaria se necesita la mitad de las K comparaciones.

Método de ordenación rápida (quicksort)

- El método de ordenación ***quicksort*** es actualmente el más eficiente y veloz de los métodos de ordenación interna. Este método es una mejora sustancial por la velocidad con que ordena los elementos del arreglo.



Método de ordenación rápida (quicksort)

La idea central de este algoritmo consiste en lo siguiente:

1. Se toma un elemento X de una posición cualquiera del arreglo.
2. Se trata de ubicar a X en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentren a su izquierda sean menores o iguales a X y todos los que se encuentran a su derecha sean mayores o iguales a X.
3. Se repiten los pasos anteriores, pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición X en el arreglo.
4. El proceso termina cuando todos los elementos se encuentran en su posición correcta en el arreglo.

Algoritmo QuickSort.

RAPIDORRECURSIVO(A,N)

```
{El algoritmo ordena los elementos del arreglo utilizando el
método rápido, de manera recursiva. A es un arreglo de N
elementos}

1. Llamar al algoritmo REDUCERRECURSIVO con 1 y N
```

El algoritmo ***rapidorecursivo*** requiere para su funcionamientos de otro algoritmo, se presenta a continuación.

Algoritmo QuickSort.

REDUCERRECURSIVO(A[],INI,FIN)

```
{INI y FIN representan las posiciones del extremos izquierdo y
derecho respectivamente del conjunto de elementos a ordenar}

{IZQ, DER, POS y AUX son variables de tipo entero. BAND es una
variable de tipo entero que contiene valores 1 y 0; 1 =
verdadero y 0 = falso}

1. Hacer IZQ ← INI, DER ← FIN, POS ← INI, BAND ← 1
2. Repetir mientras (BAND = 1)
      Hacer BAND ← 0
      2.1 Repetir mientras (A[POS]<= A[DER] y POS != DER)
            Hacer DER ← DER-1
      2.2 {Fin del ciclo del paso 2.1}
      2.3 Si POS != DER entonces
            Hacer AUX ← A[POS],A[POS]←A[DER]
                  A[DER]← AUX y POS← DER
       2.3.1 Repetir mientras (A[POS]>=A[IZQ] y POS !=IZQ)
```

```
            Hacer IZQ ← IZQ+1
      2.3.2 {Fin del ciclo del paso 2.3.1}
      2.3.3 Si POS != IZQ entonces
            Hacer BAND←1,AUX←A[POS],A[POS]←A[IZQ],
                  A[IZQ]←AUX y POS ← IZQ
      2.3.4 {Fin del condicional del paso 2.3.3}
    2.4 {Fin del condicional del paso 2.3}
3. {Fin del ciclo del paso 1}
4. Si (POS-1)>INI entonces
      Regresar a REDUCERRECURSIVO con (A,N,POS-1)
5. {Fin del condicional del paso 4}
6. Si FIN > (POS+1) entonces
      Regresar a REDUCERRECURSIVO con (A,POS+1,FIN)
7. {Fin del condicional del paso 6}
```

Método de ordenación rápida (quicksort) . Algoritmo en C:

```
#include <stdio.h>
#define N 10
void quicksort(int [], int, int);

main(){
      int a[]={10,8,7,2,1,3,5,4,6,9},i;
      quicksort(a,0,N-1);
      for (i=0;i<10;i++)
            printf("%d..",a[i]);
      putchar('\n');
      getchar();
}
```

Método de ordenación rápida (quicksort) . Algoritmo en C:

```
void quicksort(int a[],int
ini,int fin){
int izq=ini, der=fin,
pos=ini,band=1,aux;
while(band==1){
band=0;
while(a[pos]<=a[der] &&
pos!=der)
der--;
if (pos!=der){
aux=a[pos];
a[pos]=a[der];
a[der]=aux;
pos=der;
while(a[pos]>=a[izq] &&
pos!=izq)
izq++;
if (pos!=izq){
band=1;
aux=a[pos];
a[pos]=a[izq];
a[izq]=aux;
pos=izq;
}
}
}
if (pos-1>ini)
quicksort(a, ini, pos-1);
if (fin>pos+1)
quicksort(a,pos+1,fin);
}
```

Método de ordenación rápida (quicksort).

- El método quicksort es el más rápido de ordenación interna que existe en la actualidad. Esto es sorprendente, porque el método tiene su origen en el método de intercambio directo, el peor de todos los métodos directos.

Método de mezcla (merge sort).

- El método MergeSort es un algoritmo de ordenación recursivo con un número de comparaciones entre elementos del array mínimo.
- Su funcionamiento es similar al Quicksort, y está basado en la técnica divide y vencerás.

De forma resumida el funcionamiento del método MergeSort es el siguiente:

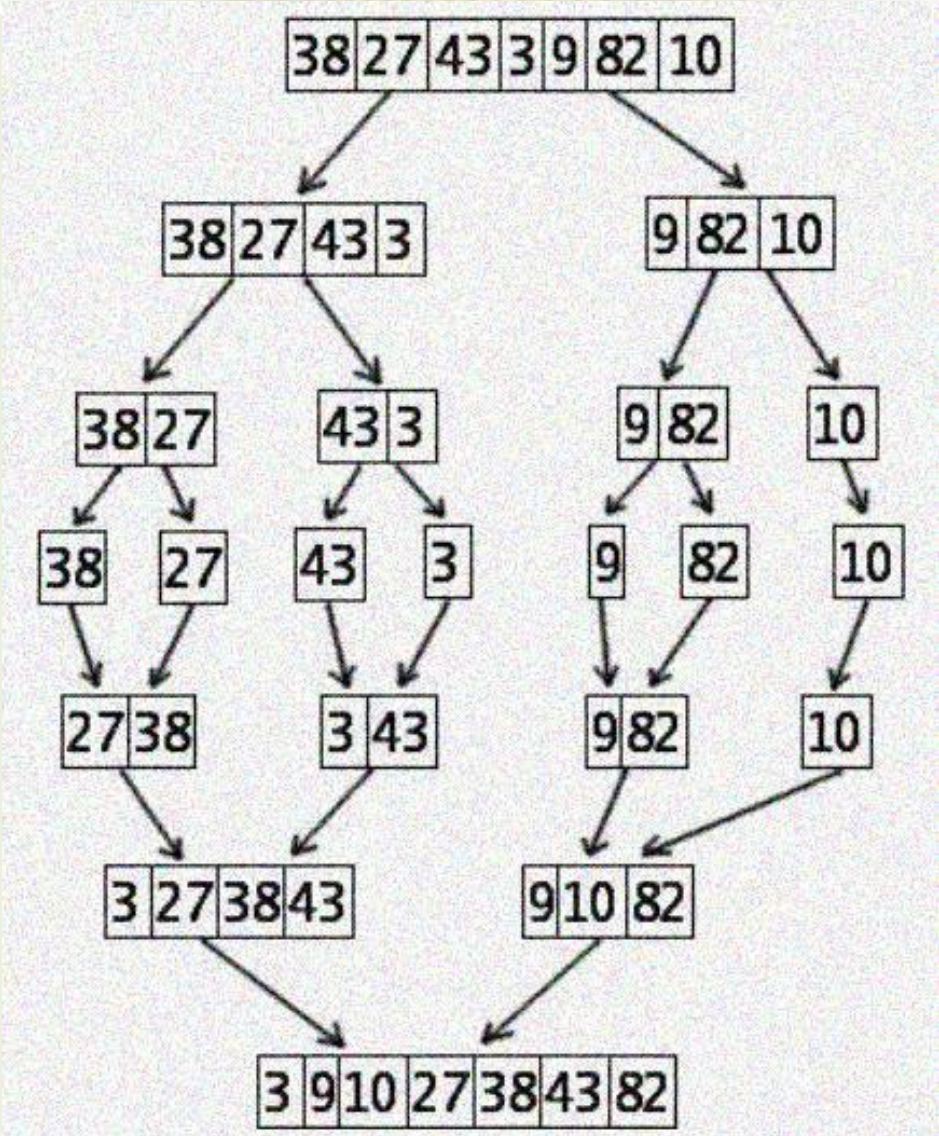
- Si la longitud del array es menor o igual a 1 entonces ya está ordenado.
- El array a ordenar se divide en dos mitades de tamaño similar.
- Cada mitad se ordena de forma recursiva aplicando el método MergeSort.
- A continuación las dos mitades ya ordenadas se mezclan formando una secuencia ordenada.

Método de mezcla (merge sort).

- Los elementos van a ser ordenados en forma creciente. Dados ***n*** elementos, éstos se dividirán en 2 subconjuntos. Cada subconjunto será ordenado y el resultado será unido para producir una secuencia de elementos ordenados.

- Si se piensa en este algoritmo recursivamente, podemos imaginar que dividirá la lista hasta tener un elemento en cada lista, luego lo compara con el que está a su lado y según corresponda, lo sitúa donde corresponde.

Merge Sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Algoritmo recursivo MergeSort.

MERGESORT(A[],INI,FIN)

```
{INI y FIN representan las posiciones del extremos izquierdo
y derecho respectivamente del conjunto de elementos a
ordenar}

{MID es una variable local de tipo entero.}

1. Si INI<FIN entonces
      1. MID←(INI+FIN)/2
      2. MERGESORT(A,INI,FIN)
      3. MERGESORT(A,INI+1,FIN)
      4. MERGE(A,INI,MID,FIN)
2. Fin del procedimiento 1.
3. Fin del modulo MERGESORT
```

Algoritmo QuickSort.

MERGE(A[],INI, MID,FIN)

```
{INI y FIN representan las posiciones del extremos izquierdo y
derecho respectivamente del conjunto de elementos a ordenar, MID
representa la posición a la mitad donde se particionará el vector}

{Se crea un vector tipo entero B[N], donde N es el tamaño del arreglo
a ordenar, H,I,J y K son variables enteras}

1. Hacer H←INI, I←INI, J←MID+1
2. Repetir mientras (H<=MID && J<=FIN)
        2.1 Si A[H]<=A[J] entonces
                B[I]←A[H]
                H++
        2.2 Fin del ciclo de paso 2.1
        2.3 entonces
                B[i]←A[J]
                J++
        2.4 Fin del ciclo de paso 2.3
        I++
3.Fin del ciclo de paso 2
```

Algoritmo QuickSort.

```
4. Si H>MID entonces
5. Repetir con K desde J hasta K<=FIN
       B[I]←A[K]
       I++
6. Fin del ciclo de paso 5
7. Entonces
       7.1 Repetir con K desde H hasta K<=MID
              B[I]←A[K]
              I++
       7.2 Fin del ciclo del paso 7.1
8. Repetir con K desde INI hasta K<=FIN
       A[K]←B[K]
9. Fin del procedimiento MERGE
```

Método de mezcla (merge sort). El algoritmo en código en C:

```
#include <stdio.h>
#define N 10
void mergesort(int [],int,int);
void merge(int [],int,int,int);

main(){
        int
        i,a[N]={9,7,10,8,2,4,6,5,
                1,3};
        mergesort(a,0,9);
        for (i=0;i<10;i++)
printf("%d..",a[i]);
getchar();
}
```

```
void mergesort(int a[],int low, int
high){
        int mid;
        if (low<high){
                mid=(low+high)/2;
        mergesort(a,low,mid);
        mergesort(a,mid+1,high);
        merge(a,low,mid,high);
        }
}
```

Método de mezcla (merge sort). El algoritmo en código en C:

```
void merge(int a[],int low, int
mid, int high){
int b[N],h,i,j,k;
h=low;
i=low;
j=mid+1;
while(h<=mid && j<=high){
        if (a[h]<=a[j]){
        b[i]=a[h];
        h++;
}
else{
        b[i]=a[j];
        j++;
}
i++;
}
```

```
if (h>mid)
for (k=j;k<=high;k++){
b[i]=a[k];
i++;
}
else
for (k=h;k<=mid;k++){
b[i]=a[k];
i++;
}
for (k=low;k<=high;k++)
a[k]=b[k];
}
```

Búsqueda Secuencial

- La búsqueda secuencial consiste en revisar elemento tras elemento hasta encontrar el dato buscado, o llegar al final del conjunto de datos disponible.

- Normalmente cuando una función de búsqueda concluye con éxito, interesa conocer en qué posición fue hallado el elemento que se estaba buscando. Esta idea se puede generalizar para todos los métodos de búsqueda.

Búsqueda secuencial iterativo

A continuación se presenta el algoritmo de búsqueda secuencial en arreglos desordenados en código C.

```
#include <stdio.h>
#define N 10
main(){
        int x,i=0,a[N]={9,7,10,8,2,4,6,5,1,3};
        scanf("%d",&x);
        while (i<N && a[i]!=x)
                i++;
        if (i>N-1)
                printf("dato no encontrado");
        else
        printf("El dato se encuentra en la posicion[%d]\n",i);
        getchar();
getchar();
}
```

Búsqueda Secuencial

A continuación se presenta una variante de este algoritmo, pero utilizando recursividad.

```
#include <stdio.h>
#define N 10
void secuencial(int [],
int,int,int);

main(){
        int
x,i=0,a[N]={9,7,10,8,2,4,6,5,1,3
};
        scanf("%d",&x);
        secuencial(a,N,x,0);
        getchar();
        getchar();
}
```

```
void secuencial(int a[],int n, int
x, int i){
if (i>n-1)
printf("Dato no localizado\n");
else if (a[i]==x)
printf("Dato localizado en la
posicion %d\n",i);
else
secuencial(a,n,x,i+1);
}
```

Búsqueda Secuencial

- El número de comparaciones es uno de los factores más importantes para determinar la complejidad de los métodos de búsqueda secuencial, se deben establecer los casos más favorables o desfavorables que se presenten.

- Al buscar un elemento en el arreglo unidimensional desordenado de N componentes, puede suceder que ese valor no se encuentre; por lo tanto, se harán N comparaciones al recorrer el arreglo.

- Por otra parte, si el elemento se encuentra en el arreglo, éste puede estar en la primer posición o en la última o en alguna intermedia.

Búsqueda binaria (Binary Search).

- La búsqueda binaria consiste en dividir el intervalo de búsqueda en dos partes, comparando el elemento buscado con el que ocupa la posición central en el arreglo.

- Para el caso de que no fueran iguales se redefinen los extremos del intervalo, según el elemento central sea mayor o menor que el elemento buscado, disminuyendo de esta forma el espacio de búsqueda.

- El proceso concluye cuando el elemento es encontrado, o cuando el intervalo de búsqueda se anula, es vacío.

Búsqueda binaria (Binary Search).

- El método de búsqueda binaria funciona exclusivamente con arreglos ordenados. No se puede utilizar con listas simplemente ligadas ni con arreglos desordenados.

- Con cada iteración del método el espacio de búsqueda se reduce a la mitad; por lo tanto, el número de comparaciones a realizarse disminuye notablemente. Esta disminución resulta significativa cuanto más grande sea el tamaño del arreglo.

Búsqueda binaria (Binary Search). Algoritmo en C:

```
#include <stdio.h>
#define N 10
main(){
int
x,low,high,mid,j,n,a[]={1,2,3,5,6,
7,8,9,10,13};
low=j=0;
high=N-1;
scanf("%d",&x);
while(low<=high){
mid=(low+high)/2;
if (x<a[mid])
high=mid-1;
else if (x>a[mid])
low=mid+1;
else{
j=mid;
break;
} }
if (j==0)
printf("Elemento no encontrado");
else
printf("Elemento localizado en el
lugar %d.\n",j);
getchar();
getchar();
}
```

BIBLIOGRAFÍA

- Cairó, Osvaldo y Guardati, Silvia. (2002). Estructuras de datos (2a. Edición). McGraw-Hill.
- Drozdeck, Adam. (2007). Estructuras de datos y algoritmos en Java (2ª Edición). Thomson.
- Joyanes Aguilar L., Fundamentos de programación, algoritmos, estructuras de datos y objetos, 4ed. McGrawHill.
- Loomis Mary E.S., Estructura de datos y organización de archivos. Prentice Hall.