Universidad Nacional Autónoma de México

Estructuras de Datos y Análisis de Algoritmos 2.

Examen de Unidad 2.

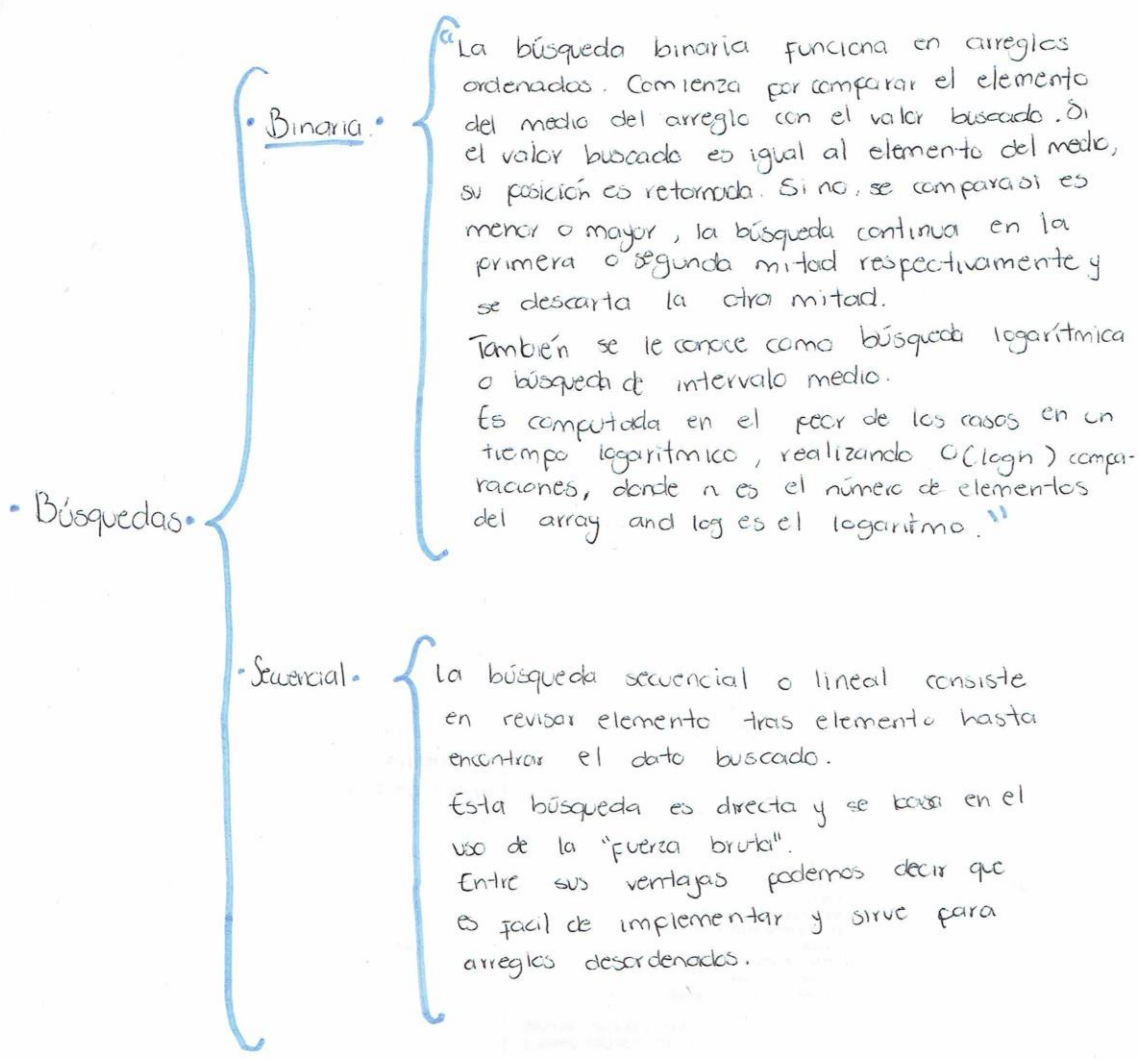
Prof. Gerardo Tovar Tapia.

Alumna. Monroy Velázquez Alejandra Sarahí

Sección 1: Investigar y argumentar las siguientes cuestiones.

Nota: Esta sección fue resuelta en hojas blancas y escaneadas.

1) Realice un cuadro sinóptico en donde exponga información relevante y compare búsqueda binaria y secuencial



2) Explicar los métodos (por lo menos 4), para evitar colisiones cuando se usan funciones hash.

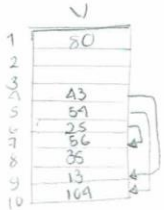
Reasignación

Se analizan tres de ellos:

→ Prueba lineal

Consiste en que una vez detectada la colisión se debe recorrer el arreglo secuencialmente a partir del punto de colisión, buscando al elemento. El proceso de búsqueda concluye cuando el elemento es hallado, o bien cuando se encuentra una posición vacía.

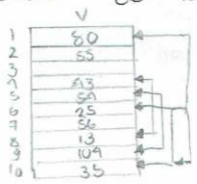
La principal desventaja es que puede haber un fuerte agrupamiento alrededor de unas claves, mientras que otras zonas del arreglo estarían vacías.



K	H(K)
25	6
43	4
56	7
35	6
69	5
13	4
80	1
109	5

→ Prueba cuadrática

Este método es similar al anterior. La diferencia es que el cuadratico las direcciones alternativas se generan como $D+1, D+4, D+9, \ldots D+i^2$ en vez de $D+1, D+2, \ldots, D+i$. Esta variación permite una mejor distribución de las claves colisionadas.



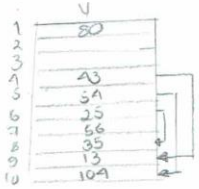
K	H(K)
25	6
43	4
56	7
35	6
69	5
13	4
80	1
109	5
55	6

La principal desventaja es que pueden quedar casillas sin visitar.

→ Doble dirección hash

Consiste en que una vez detectada la colisión debe generar otra dirección aplicando la función hash a la dirección previamente obtenida. El proceso se detiene cuando el elemento es hallado, o bien se encuentra una posición vacía.

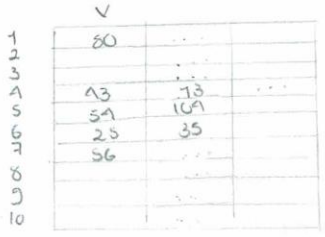
$D = H(K)$
$D' = H(D)$
$D'' = H(D')$



K	H(K)	H'(D)	H'(D')	H'(D'')
25	6	—	—	—
13	4	—	—	—
56	7	—	—	—
35	6	8	—	—
54	5	—	—	—
13	4	6	8	10
80	1	—	—	—
104	5	7	9	—

Arreglos anidados

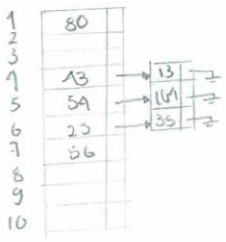
Este método consiste en que cada elemento tenga otro arreglo en el cual se almacena los elementos colisionados. Si bien la solución parece ser sencilla, es claro que resulta ineficiente.



K	H(K)
25	6
13	4
46	7
35	6
54	5
13	4
80	1
104	5

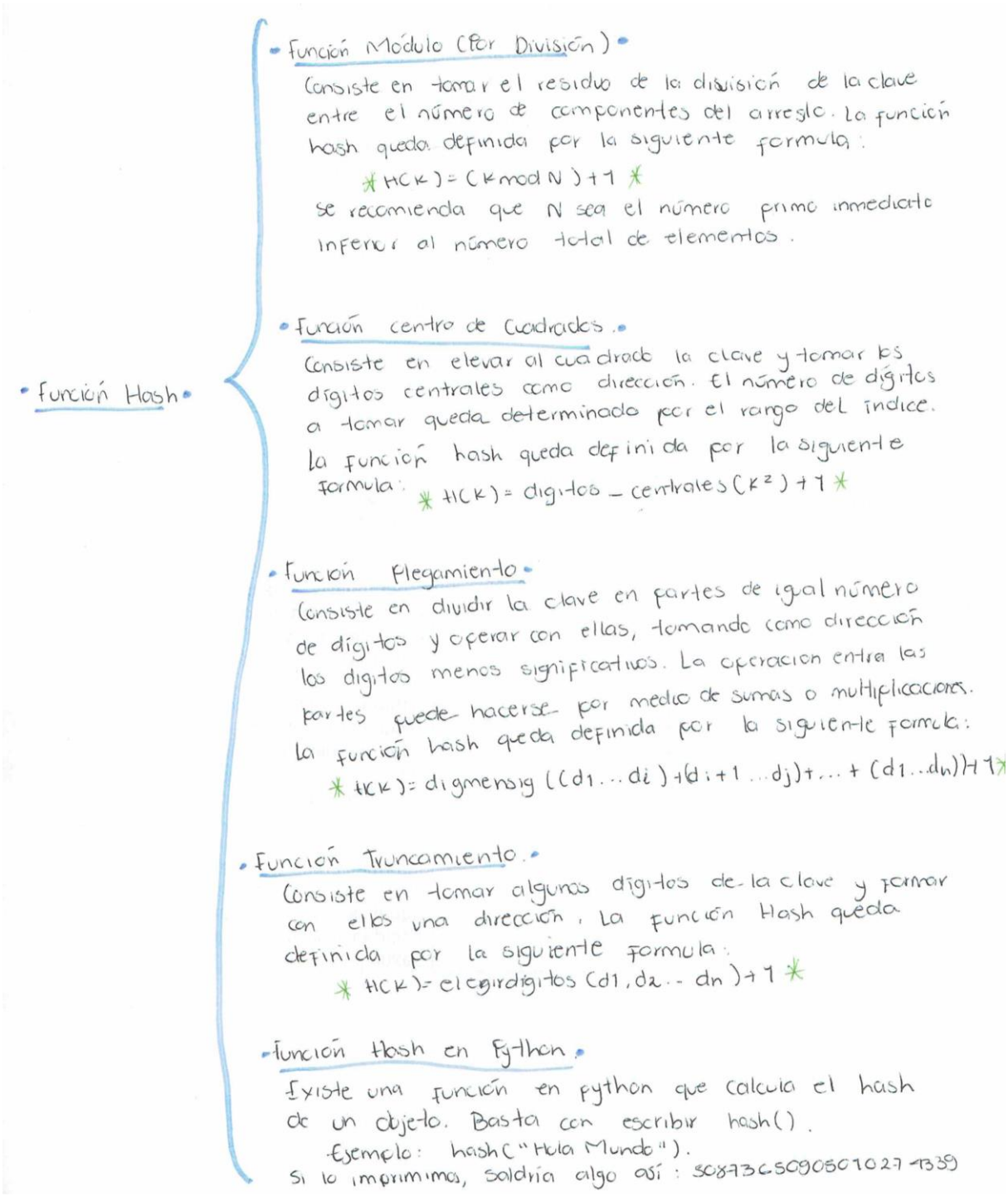
Encadenamiento

Consiste en que cada elemento del arreglo tenga un apuntador a una lista ligada, la cual se irá generando e irá almacenando los valores colisionado a medida que se requiera.



K	H(K)
25	6
13	4
56	7
35	6
54	5
13	4
80	1
104	5

3) Hacer un cuadro sinóptico en donde se expongan las diferentes formas para hacer funciones hash (por lo menos 5), explicar los algoritmos.



4) ¿En cuál de los siguientes algoritmos de búsqueda es necesario que la lista donde se realiza la búsqueda esté ordenada? (Argumentar Respuesta)

a) Búsqueda binaria
b) Tablas de dispersión con direccionamiento abierto
c) Búsqueda lineal

5) ¿En cuál de los siguientes algoritmos de búsqueda no es necesario que la lista donde se realiza la búsqueda este ordenada? (Argumentar Respuesta)

a) Búsqueda Secuencial
b) Tablas de dispersión con direccionamiento abierto
c) Búsqueda Binaria.

4.

a) Búsqueda binaria.

Se necesita que la lista este ordenada ya que el algoritmo va haciendo comparaciones con el valor que se encuentra a la mitad de la lista y pregunta si es igual, menor o mayor, si es igual retorna la posición pero si no dependiendo de las otras dos opciones, descarta la mitad correspondiente. Por ejemplo si se elige que es menor, descartara toda la mitad mayor a ese numero. Y así en cada iteración.

5.

a) Búsqueda Secuencial

En esta búsqueda la lista puede no estar ordenada, ya que el algoritmo funciona en ir recorriendo elemento por elemento y comparando si es igual al que se busca.

6) Investigar y argumentar los tipos de complejidad en un algoritmo, y dar un ejemplo de cada tipo.

Órden	Nombre	
$O(1)$	constante	Todos aquellos algoritmos que responden en un tiempo constante, sea cual sea la talla del problema. Por ejemplo, hallar el máximo de 2 valores.
$O(\log n)$	logarítmico	Los que el tiempo crece con un criterio logarítmico, independientemente de cuál sea la base mientras ésta sea mayor que 1. Por ejemplo, la búsqueda dicotómica den un vector ordenado.
$O(n)$	lineal	El tiempo crece linealmente con respecto a la talla. Por ejemplo, encontrar el máximo de un vector de talla n.
$O(n \cdot \log(n))$	enelogarítmico loglineal, linearítmico, n por logaritmo de n.	Tiene muchos nombres, es un orden relativamente bueno, porque la mayoría parte de los algoritmos tienen un orden superior. En éste caben, esta, por ejemplo el algoritmo de ordenación Quicksort.
$O(n^c)$ con $c>1$	polinómico	Aquí están muchos de los logaritmos más comunes. Cuando c es 2 es cuadrático, cuando es 3 es cúbico.... Por ejemplo el algoritmo de Dijkstra.
$O(c^n)$ con $c>1$	exponencial	Este tipo de algoritmos son peores que los anteriores, ya que crecen muchísimo más rapidamente. Por ejemplo, calcular la raiz cuadrada de un número.
$O(n!)$	factorial	Es un algoritmo que para un problema complejo prueban todas las combinaciones posibles.
$O(n^n)$	combinatorio.	Este tipo de algoritmos son igual de intratables que el anterior. A menudo no se hacen distinciones entre ellos.

7) Si se tiene una tabla hash de tamaño 5 y las llaves 10,14,16,13, 03, ¿cuáles son las direcciones de la tabla que se obtienen para cada llave utilizando la siguiente función hash? Argumentar respuesta y hacer análisis.

```
def H(llave):
return llave%5
```

a) 2,3,2,0,2
b) 0,4,1,3,3
c) 0,1,4,2,0
d) 2,2,3,2,3
e) 0,1,2,3,4

8) Utilizando alguno de los algoritmos que implementaste en las prácticas 1, 2 y 3, se desea ordenar una lista de 10000 palabras diferentes tomadas en forma aleatoria de un diccionario. Por ejemplo: ['casa','brincar','foco','verde',...,'gritar','elefante']. ¿Cuál sería el más conveniente? Argumentar tu respuesta y tu análisis.

7.

b) 0, 4, 1, 3, 3

De acuerdo con la función que se nos da, devolvemos los modulos de 5:

$5\overline{)10}\ 2$ r 0 $\quad 5\overline{)14}\ 2$ r 4 $\quad 5\overline{)16}\ 3$ r 1 $\quad 5\overline{)13}\ 2$ r 3 $\quad 5\overline{)3}\ 0$ r 3

8.

Basandome en los resultados que obtuve comparando los 6 algoritmos, el que mejor y menor tiempo de ejecución fue el counting y el radix sort. Ya que los otros algoritmos son menos eficientes, el de la burbuja es el menos eficiente de todos, seguido por el merge que va casi a la par del quick sort y heap sort.
Para tratar listas de talla grande es mejor implementar estos algoritmos. Además de que en dicha comparación no solo se trataron cadenas, si no también, caracteres, números, y flotantes, y el resultado fue el mismo.

9) ¿Cuáles de las siguientes proposiciones es verdadera en la búsqueda por transformación de llaves? Argumentar tu respuesta y tu análisis.

a) Las llaves utilizadas en una función hash deben ser caracteres o cadenas.

b) Las colisiones se presentan cuando la tabla hash es más grande que el número de llaves.

c) El manejo de colisiones se puede realizar si los elementos son mapeados a la misma dirección de la tabla hash en una lista ligada.

10) Para el siguiente pseudocódigo

```
Sort(A,p,r)
Inicio
   Si p<r entonces
      q=⌊(p+r)/2⌋ Sort(A,
      p , q)
       Sort(A, q+1 , r)
      combina(A,p,q,r)
Fin
```

a) Quick Sort

b) MergeSort

c) Heap Sort

d) Radix Sort

Indicar a que algoritmo de ordenamiento corresponde Argumentar tu respuesta y tu análisis.

9.

c) El manejo de las colisiones se puede realizar si los elementos son mapeados a la misma direccion de la tabla hash en una lista ligada.

En eso consiste el encadenamiento separado o Hasing Abierto, donde para resolver la colisión, se construye una lista enlazada de registros cuyas claves caigan en esa dirección, para cada localización de la tabla.

10.

b) MergeSort

El pseudocodigo corresponde a MergeSort ya que este algoritmo trabaja con divide y venceras. p es el primer indice de la lista y r el ultimo. Si p es menor a r se divide en dos y el indice es q. Esta es la parte de divide. Se llama asi misma la función pasandole a p y q y se va dividiendo en 2 cada vez. Posteriormente estas divisiones se van combinando pero ya arregladas.

Sección 2: Programar los siguientes ejercicios, documentar y entregar evidencia de cada programa.

11) Programa que quite valores duplicados de un vector.

```
1  listao =[1,2,3,3,3,4,5,5,5,6,6,6,7,8,9,9,9]
2  listan=[]
3  for i in listao:
4      if i not in listan:
5          listan.append(i)
6  print("-->Ejercicio 11<--\n")
7  print("Lista original: ",listao)
8  print("Nueva lista: ",listan)
9
```

```
Lista original:  [1, 2, 3, 3, 3, 4, 5, 5, 5, 6, 6, 6, 7, 8, 9, 9, 9]
Nueva lista:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

El programa elimina los valores repetidos de una lista. Se declara una lista con valores repetidos y una nueva lista vacía donde almacenaremos los valores sin los repetidos, con un ciclo for recorremos la lista y si el número actual no está en la lista nueva se agrega. Después se imprimen ambas listas.

12) Buscar si existe el número x leído en el teclado en una lista dada, la búsqueda debe ser binaria.

```python
import math

#Algoritmo de ordenamiento
def intercambia(A,x,y):
  temp = A[x]
  A[x] = A[y]
  A[y] = temp

def particionar(A,p,r):
  x = A[r]
  i = p - 1
  for j in range(p,r):
    if(A[j] <= x):
      i = i + 1
      intercambia(A,i,j)
  intercambia(A,i+1,r)
  return i + 1

def QuickSort(A,p,r):
  if(p < r):
    q = particionar(A,p,r)
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)

#Algoritmo de Busqueda Binaria
def BusquedaBinaria(A,x,izquierda,derecha):
  if izquierda > derecha:
    return -1
  medio = math.floor((izquierda + derecha)/2)

  if A[medio] == x:
    print("Se encontro en el indice: ",medio)

  elif A[medio] < x:
    return BusquedaBinaria(A,x,medio +1, derecha)
  else:
    return BusquedaBinaria(A,x,izquierda, medio-1)

print("-->Ejercicio 12<--\n")
lista = [5,4,3,2,6,9,8,1,7]
print("Lista desordenada: ",lista)
QuickSort(lista,0,8)
print("Lista ordenada: ",lista)
x = int(input("\nIngresa el valor a buscar en la lista: "
  ))
BusquedaBinaria(lista,x,0,8)
```

El programa implementa una búsqueda binaria con un algoritmo recursivo, revisando si el punto medio es igual al número buscado o si es mayor o menor y descartando la mitad correspondiente. Antes de ello utiliza un algoritmo de ordenamiento, el quick sort, ya que para hacer una búsqueda binaria, necesariamente debe de estar la lista ordenada. Así que, se tiene una lista cualquiera, se le pasa a la función QuickSort que la ordena, la devuelve y pregunta que numero se desea buscar, se ingresa y se le pasa a la función BusquedaBinaria y esta devuelve el índice en el que se encuentra dicho número.

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
-->Ejercicio 12<--

Lista desordenada:  [5, 4, 3, 2, 6, 9, 8, 1, 7]
Lista ordenada:  [1, 2, 3, 4, 5, 6, 7, 8, 9]

Ingresa el valor a buscar en la lista:  7
Se encontro en el indice:  6
>
```

13) Investigar qué tipo de Algoritmo de ordenamiento esta implementado en los siguientes Manejadores de Bases de Datos (MYSQL, SQL SERVER).

La cláusula *ORDER BY* es la instrucción que nos permite especificar el orden en el que serán devueltos los datos. Podemos especificar la ordenación ascendente o descendente a través de las palabras clave **ASC** y **DESC**. La ordenación depende del tipo de datos que esté definido en la columna, de forma que un campo numérico será ordenado como tal, y un alfanumérico se ordenará de la A a la Z, aunque su contenido sea numérico. El valor predeterminado es ASC si no se especifica al hacer la consulta.

Ejemplo:

'***SELECT'*** matricula,
marca,
modelo,
color,
numero_kilometros,
num_plazas
'***FROM'*** coches
'***ORDER BY'*** marca '***ASC'***, modelo '***DESC'***;

Este ejemplo, selecciona todos los campos matricula, marca, modelo, color, numero_kilometros y num_plazas de la tabla coches, ordenándolos por los campos marca y modelo, marca en forma ascendente y modelo en forma descendente.

14) Del documento que se anexa junto a este examen, leer el capítulo 7, y hacer un cuadro comparativo con los distintos métodos.

Algoritmo de ordenamiento	**Descripción**
Por Inserción	Consta de tomar uno por uno los elementos de un arreglo y recorrerlo hacia su posición con respecto a los anteriormente ordenados. Así empieza con el segundo elemento y lo ordena con respecto al primero. Luego sigue con el tercero y lo coloca en su posición ordenada con respecto a los dos anteriores, así sucesivamente hasta recorrer todas las posiciones del arreglo.
Por Selección	Consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo.
De la Burbuja (Bubblesort)	Se recorre el arreglo intercambiando los elementos adyacentes que estén desordenados. Se recorre el arreglo tantas veces hasta que ya no haya cambios que realizar. Prácticamente lo que hace es tomar el elemento mayor y lo va recorriendo de posición en posición hasta ponerlo en su lugar.

Rápido (Quicksort)	La idea del algoritmo es elegir un elemento llamado pivote, y ejecutar una secuencia de intercambios tal que todos los elementos menores que el pivote queden a la izquierda y todos los mayores a la derecha.
Por Montículo (Heapsort)	Consiste en almacenar todos los elementos del vector a ordenar en un montículo (heap), y luego extraer el nodo que queda como nodo raíz del montículo (cima) en sucesivas iteraciones obteniendo el conjunto ordenado.
Por Incrementos Decrecientes	La idea es reorganizar la secuencia de datos para que cumpla con la propiedad siguiente: si se toman todos los elementos separados a una distancia h, se obtiene una secuencia ordenada.
Por Mezclas Sucesivas (merge sort)	Se aplica la técnica divide-y-vencerás, dividiendo la secuencia de datos en dos subsecuencias hasta que las subsecuencias tengan un único elemento, luego se ordenan mezclando dos subsecuencias ordenadas en una secuencia ordenada, en forma sucesiva hasta obtener una secuencia única ya ordenada.