INGIENERIA INFORMATICA

MATERIA:
ESTRUCTURA DE DATOS

INVESTIGACION DEL TEMA 5
Ordenamiento

EQUIPO:
Castro Ramón David Alejandro 20010329
Martínez Ramos Rodrigo 20010347
Carrillo Ávila Juan Pablo 20010327
Guzmán Lino Roberto Rafael 20010339

GRUPO:
3 PM – 4 PM HRS CLAVE: 3a3A

Fecha de entrega: 31-05-2023

INTRODUCCION: En este trabajo que realizamos por equipo trabajamos lo que son los métodos de ordenamiento dentro de las estructuras de datos y llegamos al acuerdo de que el ordenamiento de datos se dedica a colocar los datos en cierto orden específico, como ascendente o descendente), el cual es uno de los temas más importantes dentro de este tema.

Un punto importante respecto a la ordenación interna es que el resultado final (el vector ordenado) será el mismo, sin importar qué algoritmo se utilice para ordenarlo o la elección del algoritmo sólo afecta al tiempo de ejecución y el uso que haga el programa de la memoria.

Los métodos directos más conocidos son:

Ordenación por intercambio.

Ordenación por inserción directa.

Ordenación por selección.

BURBUJA CON SEÑAL:

Descripción Breve: Consiste en utilizar una marca o señal para indicar que no se ha producido ningún intercambio en una pasada. Comprueba si el arreglo está totalmente ordenado después de cada pasada terminando su ejecución en caso afirmativo.

Análisis De Eficiencia: Este método es eficiente cuando los valores del vector se encuentran ordenados o semi ordenados.

Se basa en el mismo proceso del ordenamiento burbuja simple, pero si en una pasada no ocurren intercambios quiere decir que el vector esta ordenado, por lo tanto, hay que implementar un mecanismo para detener el proceso cuando ocurra una pasada sin intercambios.

Análisis De Los Casos:

Existen tres tipos de casos:

1. **El Mejor De Los Casos (Ordenado)**
    a. 124 750 comparaciones
    b. 0 movimientos

2. **El Caso Medio (Desordenado o Aleatorio)**
    a. 124 750 comparaciones
    b. 187 125 movimientos

3. **El Peor De Los Casos (Orden Inverso)**
    a. 124 750 comparaciones
    b. 374 250 comparaciones

Con el arreglo anterior, {40,21,4,9,10,35}:

Primera pasada:

{21,40,4,9,10,35} <-- Se cambia el 21 por el 40.

{21,4,40,9,10,35} <-- Se cambia el 40 por el 4.

{21,4,9,40,10,35} <-- Se cambia el 9 por el 40.

{21,4,9,10,40,35} <-- Se cambia el 40 por el 10.

{21,4,9,10,35,40} <-- Se cambia el 35 por el 40.

Segunda pasada:

{4,21,9,10,35,40} <-- Se cambia el 21 por el 4.

{4,9,21,10,35,40} <-- Se cambia el 9 por el 21.

{4,9,10,21,35,40} <-- Se cambia el 21 por el 10.

Ya están ordenados, pero para comprobarlo habría que acabar esta segunda

comprobación y hacer una tercera.

Prueba de escritorio burbuja con señal:

Lista=[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

Lista	N	Señal	i<N	j<N-i-1	tempo
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	false	0	0	23
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	10	true	0	1	
[-56, 23, 2, 67, 9, 44, 6, 18, 1, 7]	10	true	0	2	67
[-56, 23, 2, 9, 67, 44, 6, 18, 1, 7]	10	true	0	3	67
[-56, 23, 2, 9, 44, 67, 6, 18, 1, 7]	10	true	0	4	67
[-56, 23, 2, 9, 44, 6, 67, 18, 1, 7]	10	true	0	5	67
[-56, 23, 2, 9, 44, 6, 18, 67, 1, 7]	10	true	0	6	67
[-56, 23, 2, 9, 44, 6, 18, 1, 67, 7]	10	true	0	7	67
[-56, 23, 2, 9, 44, 6, 18, 1, 7, 67]	10	true	0	8	67
[-56, 23, 2, 9, 44, 6, 18, 1, 7, 67]	10	false	1	0	
[-56, 2, 23, 9, 44, 6, 18, 1, 7, 67]	10	true	1	1	23
[-56, 2, 9, 23, 44, 6, 18, 1, 7, 67]	10	true	1	2	23
[-56, 2, 9, 23, 44, 6, 18, 1, 7, 67]	10	true	1	3	

[-56, 2, 9, 23, 6, 44, 18, 1, 7, 67]	10	true	1	4	44
[-56, 2, 9, 23, 6, 18, 44, 1, 7, 67]	10	true	1	5	44
[-56, 2, 9, 23, 6, 18, 1, 44, 7, 67]	10	true	1	6	44
[-56, 2, 9, 23, 6, 18, 1, 7, 44, 67]	10	true	1	7	44
[-56, 2, 9, 23, 6, 18, 1, 7, 44, 67]	10	false	2	0	
[-56, 2, 9, 23, 6, 18, 1, 7, 44, 67]	10	false	2	1	
[-56, 2, 9, 23, 6, 18, 1, 7, 44, 67]	10	false	2	2	
[-56, 2, 9, 6, 23, 18, 1, 7, 44, 67]	10	true	2	3	23
[-56, 2, 9, 6, 18, 23, 1, 7, 44, 67]	10	true	2	4	23
[-56, 2, 9, 6, 18, 1, 23, 7, 44, 67]	10	true	2	5	23
[-56, 2, 9, 6, 18, 1, 7, 23, 44, 67]	10	true	2	6	23
[-56, 2, 9, 6, 18, 1, 7, 23, 44, 67]	10	true	2	7	23
[-56, 2, 9, 6, 18, 1, 7, 23, 44, 67]	10	false	3	0	
[-56, 2, 9, 6, 18, 1, 7, 23, 44, 67]	10	false	3	1	
[-56, 2, 6, 9, 18, 1, 7, 23, 44, 67]	10	true	3	2	9

[-56, 2, 6, 9, 18, 1, 7, 23, 44, 67]	10	true	3	3	
[-56, 2, 6, 9, 1, 18, 7, 23, 44, 67]	10	true	3	4	18
[-56, 2, 6, 9, 1, 7, 18, 23, 44, 67]	10	true	3	5	18
[-56, 2, 6, 9, 1, 7, 18, 23, 44, 67]	10	false	4	0	
[-56, 2, 6, 9, 1, 7, 18, 23, 44, 67]	10	false	4	1	
[-56, 2, 6, 9, 1, 7, 18, 23, 44, 67]	10	false	4	2	
[-56, 2, 6, 1, 9, 7, 18, 23, 44, 67]	10	true	4	3	9
[-56, 2, 6, 1, 7, 9, 18, 23, 44, 67]	10	true	4	4	9
[-56, 2, 6, 1, 7, 9, 18, 23, 44, 67]	10	false	5	0	
[-56, 2, 6, 1, 7, 9, 18, 23, 44, 67]	10	false	5	1	
[-56, 2, 1, 6, 7, 9, 18, 23, 44, 67]	10	true	5	2	6
[-56, 2, 1, 6, 7, 9, 18, 23, 44, 67]	10	true	5	3	
[-56, 2, 1, 6, 7, 9, 18, 23, 44, 67]	10	false	6	0	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10	true	6	1	2
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10	true	6	2	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10	false	7	0	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10	false	7	1	

DOBLE BURBUJA CON SEÑAL:

Descripción Breve: Funciona mediante la comparación repetida de elementos adyacentes y su intercambio si están en un orden incorrecto

Análisis De Eficiencia: Las comparaciones repetidas hacen que el elemento más pequeño/más grande se desplace hacia el final del array, por lo que este algoritmo recibe el nombre de ordenación burbuja. Aunque es ineficiente, sigue siendo la base de los algoritmos de ordenación.

Análisis De Los Casos:

Existen tres tipos de casos:

1. **El Mejor De Los Casos (Ordenado)**
   El mejor caso se da cuando la array ya está ordenada, y entonces sólo se necesitan N comparaciones.
   La complejidad temporal en el mejor caso es O(n).

2. **El Caso Medio (Desordenado o Aleatorio)**
   Por término medio, se realizan n-i comparaciones en la quinta pasada del ordenamiento de burbuja. Por tanto, si hay n pases, la complejidad temporal media puede ser la siguiente.
   (n-1) + (n-2) + (n-3) ..... + 1 = n*(n-1)/2
   Por tanto, la complejidad temporal es del orden de O(n2).

3. **El Peor De Los Casos (Orden Inverso)**

El peor caso se da cuando el array está ordenado de forma inversa, y hay que realizar el máximo número de comparaciones e intercambios.

La complejidad temporal en el peor caso es del orden de O(n2).

Complejidad espacial

La complejidad espacial de este algoritmo es O(n) debido a las llamadas recursivas almacenadas dentro de la pila.

Supongamos que tenemos el array (5,3,4,2,1). Vamos a ordenarlo utilizando el algoritmo de ordenamiento de burbuja.

Primer pase:

(5 3 4 2 1) → (3 5 4 2 1) (3 < 5 intercambiados)

(3 5 4 2 1) → (3 4 5 2 1) (4 < 5 intercambiados)

(3 4 5 2 1) → (3 4 2 5 1) (2 < 5 intercambiados)

(3 4 2 5 1) → (3 4 2 1 5) (1 < 5 intercambiado)

Segundo pase:

(3 4 2 1 5) → (3 4 2 1 5)

(3 4 2 1 5) → (3 2 4 1 5) (2 < 4 intercambiados)

(3 2 4 1 5) → (3 2 1 4 5) (1 < 4 intercambiado)

Tercer pase:

(3 2 1 4 5) → (2 3 1 4 5) (2 < 3 intercambiado)

(2 3 1 4 5) → (2 1 3 4 5) (1 < 3 intercambiado)

Cuarto pase:

(2 1 3 4 5) → (1 2 3 4 5) (1 < 2 intercambiado)

Obtenemos el array ordenado después de la cuarta pasada - (1 2 3 4 5)

Método de ordenamiento Doble burbuja con señal

Lista = [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

Lista	left	right	swapped	i	J	temp	k
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	9	false	0			
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]				0	1	23	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	9	true	1			
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	9	true	2			
[-56, 23, 2, 67, 9, 44, 6, 18, 1, 7]				2	3	67	
[-56, 23, 2, 67, 9, 44, 6, 18, 1, 7]	0	9	true	3			
[-56, 23, 2, 9, 67, 44, 6, 18, 1, 7]				3	4		67
[-56, 23, 2, 9, 67, 44, 6, 18, 1, 7]	0	9	true	4			
[-56, 23, 2, 9, 44, 67, 6, 18, 1, 7]				4	5		67
[-56, 23, 2, 9, 44, 67, 6, 18, 1, 7]	0	9	true	5			
[-56, 23, 2, 9, 44, 6, 67, 18, 1, 7]				5	6		67
[-56, 23, 2, 9, 44, 6, 67, 18, 1, 7]	0	9	true	6			
[-56, 23, 2, 9, 44, 6, 18, 67, 1, 7]				6	7		67
[-56, 23, 2, 9, 44, 6, 18, 67, 1, 7]	0	9	true	7			
[-56, 23, 2, 9, 44, 6, 18, 1, 67, 7]				7	8		67
[-56, 23, 2, 9, 44, 6, 18, 1, 67, 7]	0	9	true	8			
[-56, 23, 2, 9, 44, 6, 18, 1, 7, 67]				8	9		67
[-56, 23, 2, 9, 44, 6, 18, 1, 7, 67]	0	8	true	8			

[-56, 23, 2, 9, 44, 6, 18, 1, 7, 67]	0	8	true	7			
[-56, 23, 2, 9, 44, 6, 1, 18, 7, 67]				7	6		1
[-56, 23, 2, 9, 44, 6, 1, 18, 7, 67]	0	8	true	6			
[-56, 23, 2, 9, 44, 1, 6, 18, 7, 67]				6	5		1
[-56, 23, 2, 9, 44, 1, 6, 18, 7, 67]	0	8	true	5			
[-56, 23, 2, 9, 1, 44, 6, 18, 7, 67]				5	4		1
[-56, 23, 2, 9, 1, 44, 6, 18, 7, 67]	0	8	true	4			
[-56, 23, 2, 1, 9, 44, 6, 18, 7, 67]				4	3		1
[-56, 23, 2, 1, 9, 44, 6, 18, 7, 67]	0	8	true	3			
[-56, 23, 1, 2, 9, 44, 6, 18, 7, 67]				3	2		1
[-56, 23, 1, 2, 9, 44, 6, 18, 7, 67]	0	8	true	2			
[-56, 1, 23, 2, 9, 44, 6, 18, 7, 67]				2	1		1
[-56, 1, 23, 2, 9, 44, 6, 18, 7, 67]	0	8	true	1			
[-56, 1, 23, 2, 9, 44, 6, 18, 7, 67]	1	8	false	1			
[-56, 1, 23, 2, 9, 44, 6, 18, 7, 67]	1	8	false	2			
[-56, 1, 2, 23, 9, 44, 6, 18, 7, 67]				2	3		23
[-56, 1, 2, 23, 9, 44, 6, 18, 7, 67]	1	8	true	3			
[-56, 1, 2, 9, 23, 44, 6, 18, 7, 67]				3	2		23
[-56, 1, 2, 9, 23, 44, 6, 18, 7, 67]	1	8	true	4			

[-56, 1, 2, 9, 23, 44, 6, 18, 7, 67]	1	8	true	5			
[-56, 1, 2, 9, 23, 6, 44, 18, 7, 67]				5	6		44
[-56, 1, 2, 9, 23, 6, 44, 18, 7, 67]	1	8	true	6			
[-56, 1, 2, 9, 23, 6, 18, 44, 7, 67]				6	7		44
[-56, 1, 2, 9, 23, 6, 18, 44, 7, 67]	1	8	true	7			
[-56, 1, 2, 9, 23, 6, 18, 7, 44, 67]				7	8		44
[-56, 1, 2, 9, 23, 6, 18, 7, 44, 67]	1	7	true	7			
[-56, 1, 2, 9, 23, 6, 7, 18, 44, 67]				7	6		7
[-56, 1, 2, 9, 23, 6, 7, 18, 44, 67]	1	7	true	6			
[-56, 1, 2, 9, 23, 6, 7, 18, 44, 67]	1	7	true	5			
[-56, 1, 2, 9, 6, 23, 7, 18, 44, 67]				5	4		6
[-56, 1, 2, 9, 6, 23, 7, 18, 44, 67]	1	7	true	4			
[-56, 1, 2, 6, 9, 23, 7, 18, 44, 67]				4	3		6
[-56, 1, 2, 6, 9, 23, 7, 18, 44, 67]	1	7	true	4			
[-56, 1, 2, 6, 9, 23, 7, 18, 44, 67]	1	7	true	3			
[-56, 1, 2, 6, 9, 23, 7, 18, 44, 67]	1	7	true	2			
[-56, 1, 2, 6, 9, 23, 7, 18, 44, 67]	2	7	false				
[-56, 1, 2, 6, 9, 23, 7, 18, 44, 67]	2	7	false	2			
[-56, 1, 2, 6, 9, 23, 7, 18, 44, 67]	2	7	false	3			

[-56, 1, 2, 6, 9, 23, 7, 18, 44, 67]	2	7	false	4			
[-56, 1, 2, 6, 9, 23, 7, 18, 44, 67]	2	7	false	5			
[-56, 1, 2, 6, 9, 7, 23, 18, 44, 67]				5	6		23
[-56, 1, 2, 6, 9, 7, 23, 18, 44, 67]	2	7	true	6			
[-56, 1, 2, 6, 9, 7, 18, 23, 44, 67]				6	7		23
[-56, 1, 2, 6, 9, 7, 18, 23, 44, 67]	2	7	true	6			
[-56, 1, 2, 6, 9, 7, 18, 23, 44, 67]	2	6	true				
[-56, 1, 2, 6, 9, 7, 18, 23, 44, 67]	2	6	true	6			
[-56, 1, 2, 6, 9, 7, 18, 23, 44, 67]	2	6	true	5			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]				5	4		7
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	2	6	true	5			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	2	6	true	4			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	2	6	true	3			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	3	6	false				
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	3	6	false	3			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	3	6	false	4			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	3	6	false	5			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	3	5	false				
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	3	5	false	5			

[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	3	5	false	4			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	4	5	false				

SHELL (INCREMENTOS - DECREMENTOS):

Descripción Breve: El ordenamiento de Shell, a veces llamado “ordenamiento de incremento decreciente”, mejora el ordenamiento por inserción al romper la lista original en varias sablistas más pequeñas, cada una de las cuales se ordena mediante un ordenamiento por inserción.

ANALISIS DE EFICIENCIA:

La manera única en que se eligen estas sublistas es la clave del ordenamiento de Shell. En lugar de dividir la lista en sublistas de ítems contiguos, el ordenamiento de Shell usa un incremento i, a veces denominado brecha, para crear una sublista eligiendo todos los ítems que están separados por i ítems.

El método Shell resulta más útil que otros, ya que a través de él se puede dividir en subtareas el trabajo, haciendo ordenamientos independientes para después unirlos.

En las siguientes sentencias de programación, elige la respuesta correcta a cada reactivo que se te presenta. Al finalizar podrás conocer tu desempeño.

ANALISIS DE LOS CASOS:

- Existen tres tipos de casos:

  Aleatorio: Tiempo de ejecución para 50 tiempos

generados de manera aleatoria, con

incrementos de 2000 en el tamaño del vector.

• Ordenado: Tiempo de ejecución para 50

arreglos ordenados anticipadamente.

Incrementos de 2000.

• Desordenado: Tiempo de ejecución para 50

arreglos que contienen datos ordenados de

Mayor a Menor. Incrementos de 2000.

EJEMPLO: Por ejemplo, lo pasos para ordenar el arreglo {40,21,4,9,10,35} mediante el método de Shell serían:

Salto=3:

Primera pasada:

{9,21,4,40,10,35} <-- se intercambian el 40 y el 9.

{9,10,4,40,21,35} <-- se intercambian el 21 y el 10.

Salto=1:

Primera pasada:

{9,4,10,40,21,35} <-- se intercambian el 10 y el 4.

{9,4,10,21,40,35} <-- se intercambian el 40 y el 21.

{9,4,10,21,35,40} <-- se intercambian el 35 y el 40.

Segunda pasada:

{4,9,10,21,35,40} <-- se intercambian el 4 y el 9.

Con sólo 6 intercambios se ha ordenado el arreglo, cuando por inserción se necesitaban muchos más.

Prueba de escritorio del método Shell:

Lista = [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

Lista	n	h	i	Key	j
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1			
[23, -56, 67, 2, 23, 44, 6, 18, 1, 7]	10	4	4	9	4

[9, -56, 67, 2, 23, 44, 6, 18, 1, 7]	10	4	4	9	0
[9, -56, 67, 2, 23, 44, 6, 18, 1, 7]	10	4	5	44	5
[9, -56, 67, 2, 23, 44, 67, 18, 1, 7]	10	4	6	6	6
[9, -56, 67, 2, 23, 44, 67, 18, 1, 7]	10	4	6	6	2
[9, -56, 6, 2, 23, 44, 67, 18, 1, 7]	10	4	6		
[9, -56, 6, 2, 23, 44, 67, 18, 1, 7]	10	4	7	18	7
[9, -56, 6, 2, 23, 44, 67, 18, 23, 7]	10	4	8	1	8
[9, -56, 6, 2, 9, 44, 67, 18, 23, 7]	10	4	8	1	4
[1, -56, 6, 2, 9, 44, 67, 18, 23, 7]	10	4	8	1	0
[1, -56, 6, 2, 9, 44, 67, 18, 23, 44]	10	4	9	7	9
[1, -56, 6, 2, 9, 7, 67, 18, 23, 44]	10	4	9	7	5
[1, 1, 6, 2, 9, 7, 67, 18, 23, 44]	10	1	1	-56	1
[-56, 1, 6, 2, 9, 7, 67, 18, 23, 44]	10	1	1	-56	0

[-56, 1, 6, 2, 9, 7, 67, 18, 23, 44]	10	1	1	-56	0
[-56, 1, 6, 2, 9, 7, 67, 18, 23, 44]	10	1	2	6	2
[-56, 1, 6, 6, 9, 7, 67, 18, 23, 44]	10	1	3	2	3
[-56, 1, 2, 6, 9, 7, 67, 18, 23, 44]	10	1	3	2	2
[-56, 1, 2, 6, 9, 7, 67, 18, 23, 44]	10	1	4	9	4
[-56, 1, 2, 6, 9, 9, 67, 18, 23, 44]	10	1	5	7	5
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	10	1	5	7	4
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	10	1	6	67	6
[-56, 1, 2, 6, 7, 9, 67, 67, 23, 44]	10	1	7	18	7
[-56, 1, 2, 6, 7, 9, 18, 67, 23, 44]	10	1	7	18	6
[-56, 1, 2, 6, 7, 9, 18, 67, 67, 44]	10	1	8	23	8
[-56, 1, 2, 6, 7, 9, 18, 23, 67, 44]	10	1	8	23	7
[-56, 1, 2, 6, 7, 9, 18, 23, 67, 67]	10	1	9	44	9

[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10	1	9	44	8
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10	0			

SELECCIÓN DIRECTA:

Descripción Breve: Consiste en buscar el menor elemento del arreglo y colocarlo en la primera posición. Luego se busca el segundo elemento más pequeño del arreglo y se coloca en la segunda posición. El proceso continuo hasta que todos los elementos del arreglo han sido ordenados.

ANALISIS DE EFICIENCIA:

Se basa en realizar varias pasadas, intentando encontrar en cada una de ellas el elemento que según el criterio de ordenación es mínimo y colocándolo posteriormente en su sitio.

Es relativamente simple. Se debe considerar que el número de comparaciones entre los elementos es independiente de la disposición inicial de estos arreglos. En la primera pasada se realizan (n-1) comparaciones, en la segunda pasada (n- 2) comparaciones y así sucesivamente hasta 2 y 1 comparaciones.

Es igual:

$C= n-1 + n-2 +\ldots+2+1= n* n-1\ 2 = n\ 2 -n\ 2$

Respecto al numero de intercambios, siempre será n- 1. por lo tanto: M=n-1

ANALISIS DE LOS CASOS:

- Existen tres tipos de casos: .

- La función ordSeleccion() ordena una lista o vector de números reales de n elementos. En la pasada i, el proceso de selección explora la sublista A[i] a A[n-1] y fija el índice del elemento más pequeño. Después de terminar

la exploración, los elementos A[i] y A[indiceMenor] intercambian las posiciones.

```
/*
ordenar un array de n elementos de tipo double
utilizando el algoritmo de ordenación por selección
*/
void ordSeleccion (double a[], int n)
{
int indiceMenor, i, j;
/* ordenar a[0]..a[n-2] y a[n-1] en cada pasada */
for (i = 0; i < n-1; i++)
{
/* comienzo de la exploración en índice i */
indiceMenor = i;
/* j explora la sublista a[i+1]..a[n-1] */
for (j = i+1; j < n; j++)
if (a[j] < a[indiceMenor])
indiceMenor = j;
/* situa el elemento mas pequeño en a[i] */
if (i != indiceMenor)
{
```


21 36 39 80 51

pasada 3

Pasada 3: Seleccionar 51

Intercambiar 51 y A[3]

21 36 39 51 80 Lista ordenada

```
double aux = a[i];
a[i] = a[indiceMenor];
a[indiceMenor] = aux ;
}
}
}
```

El análisis del algoritmo de selección es sencillo y claro, ya que requiere un número fijo de comparaciones que sólo dependen del tamaño de la lista o array y no de la distribución inicial de los datos.

• Comparaciones:

C=(n-1)+(n-2)+…+2+1=(n*(n-1))/2= (n

2-n)/2

• Intercambios o Movimientos:

M= n-1

Si se quisiera ordenar un arreglo de 500 elementos:

Se efectuarán 124,750 comparaciones.

Se efectuarán 499 movimientos.

Por ejemplo, si tenemos el arreglo {40,21,4,9,10,35}, los pasos a seguir

son:

{4,21,40,9,10,35} <-- Se coloca el 4, el más pequeño, en primera posición

: se cambia el 4 por el 40.

{4,9,40,21,10,35} <-- Se coloca el 9, en segunda posición: se cambia el 9

por el 21.

{4,9,10,21,40,35} <-- Se coloca el 10, en tercera posición: se cambia el 10 por el 40.

{4,9,10,21,40,35} <-- Se coloca el 21, en tercera posición: ya está colocado.

{4,9,10,21,35,40} <-- Se coloca el 35, en tercera posición: se cambia el 35 por el 40.

Prueba de escritorio de Selección directa:

Lista	n	minIndex	i	j	temp
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	0	0	1	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0	2	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0	3	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0	3	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0	4	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0	5	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0	6	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0	7	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0	8	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0	9	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	0		-56

[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	10	1	1	2	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	10	3	1	4	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	10	3	1	5	

Lista = [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

……A partir de aquí lo único que cambia es la j hasta la posición 8

[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	10	8	1	8	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	10	8	1	9	
[-56, 1, 67, 2, 9, 44, 6, 18, 23, 7]	10	8	1		1
[-56, 1, 67, 2, 9, 44, 6, 18, 23, 7]	10	2	2	3	
[-56, 1, 67, 2, 9, 44, 6, 18, 23, 7]	10	3	2	3	
[-56, 1, 67, 2, 9, 44, 6, 18, 23, 7]	10	3	2	4	

……A partir de aquí lo único que cambia es la j hasta la posición 9

[-56, 1, 67, 2, 9, 44, 6, 18, 23, 7]	10	3	2	9	
[-56, 1, 2, 67, 9, 44, 6, 18, 23, 7]	10	3	2		2
[-56, 1, 2, 67, 9, 44, 6, 18, 23, 7]	10	3	3	4	
[-56, 1, 2, 67, 9, 44, 6, 18, 23, 7]	10	4	3	4	
[-56, 1, 2, 67, 9, 44, 6, 18, 23, 7]	10	4	3	5	
[-56, 1, 2, 67, 9, 44, 6, 18, 23, 7]	10	6	3	6	

[-56, 1, 2, 67, 9, 44, 6, 18, 23, 7]	10	6	3	7	
[-56, 1, 2, 67, 9, 44, 6, 18, 23, 7]	10	6	3	8	
[-56, 1, 2, 67, 9, 44, 6, 18, 23, 7]	10	6	3	9	
[-56, 1, 2, 6, 9, 44, 67, 18, 23, 7]	10	6	3		6
[-56, 1, 2, 6, 9, 44, 67, 18, 23, 7]	10	4	4	5	
[-56, 1, 2, 6, 9, 44, 67, 18, 23, 7]	10	4	4	6	

……A partir de aquí lo único que cambia es la j hasta la posición 9

[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	10	9	4		7
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	10	5	5	6	
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	10	7	5	7	
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	10	7	5	8	
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	10	9	5	9	
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	10	9	5		9
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	10	6	6	7	
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	10	7	6	7	
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	10	7	6	8	
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	10	7	6	9	
[-56, 1, 2, 6, 7, 9, 18, 67, 23, 44]	10	7	6		18
[-56, 1, 2, 6, 7, 9, 18, 67, 23, 44]	10	8	7	8	
[-56, 1, 2, 6, 7, 9, 18, 67, 23, 44]	10	8	7	9	

[-56, 1, 2, 6, 7, 9, 18, 23, 67, 44]	10	8	7		23
[-56, 1, 2, 6, 7, 9, 18, 23, 67, 44]	10	9	8	9	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 66]	10	9	8		44
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 66]	10				

INSERCION DIRECTA:

Descripción Breve: En este método lo que se hace es tener una sablista ordenada de

elementos del arreglo e ir insertando el resto en el lugar adecuado para que la sablista no pierda el orden.

La sablista ordenada se va haciendo cada vez mayor, de modo que al final la lista entera queda ordenada.

ANALISIS DE EFICIENCIA:

Este método es eficiente cuando los valores del vector se encuentran ordenados o semi ordenados.

Se basa en el mismo proceso del ordenamiento burbuja simple, pero si en una pasada no ocurren intercambios quiere decir que el vector esta ordenado, por lo tanto, hay que implementar un mecanismo para detener el proceso cuando ocurra una pasada sin intercambios.

ANALISIS DE LOS CASOS:

- Existen tres tipos de casos: .

- Esto ocurre cuando los datos de entrada ya están ordenados.

Peor caso: El peor caso ocurre cuando los datos de entrada están ordenados a la inversa. En este caso,

d2 =1, d3=2, ..., dn=n-1. De este modo,

Caso promedio: Cuando se está considerando x¡, ya se han ordenado (i — 1) datos. Si x¡ es el más grande de todos los i números, entonces el ciclo interior no se ejecuta y dentro de este ciclo interior no hay en absoluto ningún movimiento de datos. Si x¡ es el segundo más grande de todos los i números, habrá un intercambio de datos, y así sucesivamente.

La probabilidad de que x¡ sea el más grande es 1/i. Esta también es la probabilidad de que x¡ sea el j-ésimo más grande para 1 <= j<= i. En consecuencia, el promedio (2 + d¡) es:

La complejidad temporal media para el ordenamiento por inserción directa es:

En resumen, la complejidad temporal del ordenamiento por inserción directa para cada caso es:

Mejor caso: 2(n — 1) = O(n).

Caso promedio: 1/4 (n + 8)(n - 1) = O(n^2).

Peor caso: 1/2 (n - 1)(n + 4) = O(n^2).

Para el ejemplo {40,21,4,9,10,35}, se tiene:

{40,21,4,9,10,35} <-- La primera sublista ordenada es {40}.

Insertamos el 21:

{40,40,4,9,10,35} <-- aux=21;

{21,40,4,9,10,35} <-- Ahora la sublista ordenada es {21,40}.

Insertamos el 4:

{21,40,40,9,10,35} <-- aux=4;

{21,21,40,9,10,35} <-- aux=4;

{4,21,40,9,10,35} <-- Ahora la sublista ordenada es {4,21,40}.

Insertamos el 9:

{4,21,40,40,10,35} <-- aux=9;

{4,21,21,40,10,35} <-- aux=9;

{4,9,21,40,10,35} <-- Ahora la sublista ordenada es {4,9,21,40}.

Insertamos el 10:

{4,9,21,40,40,35} <-- aux=10;

{4,9,21,21,40,35} <-- aux=10;

{4,9,10,21,40,35} <-- Ahora la sublista ordenada es {4,9,10,21,40}.

Y por último insertamos el 35:

{4,9,10,21,40,40} <-- aux=35;

{4,9,10,21,35,40} <-- El arreglo está ordenado.

Prueba de escritorio de Inserción directa:

Lista = [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

Lista	n	Key	i	j	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]					
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	10	-56	1	-1	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	10	67	2	1	
[-56, 23, 67, 67, 9, 44, 6, 18, 1, 7]	10	2	3	2	
[-56, 23, 23, 67, 9, 44, 6, 18, 1, 7]	10	2	3	1	
[-56, 2, 23, 67, 9, 44, 6, 18, 1, 7]	10	2	3	0	
[-56, 2, 23, 67, 9, 44, 6, 18, 1, 7]	10	9	4	3	
[-56, 2, 23, 67, 67, 44, 6, 18, 1, 7]	10	9	4	2	
[-56, 2, 23, 23, 67, 44, 6, 18, 1, 7]	10	9	4	1	
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	10		4		

[-56, 2, 9, 23, 67, 67, 6, 18, 1, 7]	10	44	5	4	
[-56, 2, 9, 23, 44, 67, 6, 18, 1, 7]	10	44	5	3	
[-56, 2, 9, 23, 44, 67, 67, 18, 1, 7]	10	6	6	5	
[-56, 2, 9, 23, 44, 44, 67, 18, 1, 7]	10	6	6	4	
[-56, 2, 9, 23, 23, 44, 67, 18, 1, 7]	10	6	6	3	
[-56, 2, 9, 9, 23, 44, 67, 18, 1, 7]	10	6	6	2	
[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	10	6	6	1	
[-56, 2, 6, 9, 23, 44, 67, 67, 1, 7]	10	18	7	6	
[-56, 2, 6, 9, 23, 44, 44, 67, 1, 7]	10	18	7	5	
[-56, 2, 6, 9, 23, 23, 44, 67, 1, 7]	10	18	7	4	
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	10	18	7	3	
[-56, 2, 6, 9, 18, 23, 44, 67, 67, 7]	10	1	8	7	
[-56, 2, 6, 9, 18, 23, 44, 44, 67, 7]	10	1	8	6	
[-56, 2, 6, 9, 18, 23, 23, 44, 67, 7]	10	1	8	5	
[-56, 2, 6, 9, 18, 18, 23, 44, 67, 7]	10	1	8	4	
[-56, 2, 6, 9, 9, 18, 23, 44, 67, 7]	10	1	8	3	
[-56, 2, 6, 6, 9, 18, 23, 44, 67, 7]	10	1	8	2	
[-56, 2, 2, 6, 9, 18, 23, 44, 67, 7]	10	1	8	1	
[-56, 1, 2, 6, 9, 18, 23, 44, 67, 7]	10	1	8	0	

[-56, 1, 2, 6, 9, 18, 23, 44, 67, 67]	10	7	9	8	
[-56, 1, 2, 6, 9, 18, 23, 44, 44, 67]	10	7	9	7	
[-56, 1, 2, 6, 9, 18, 23, 23, 44, 67]	10	7	9	6	
[-56, 1, 2, 6, 9, 18, 18, 23, 44, 67]	10	7	9	5	
[-56, 1, 2, 6, 9, 9, 18, 23, 44, 67]	10	7	9	4	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10	7	9	3	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10				

BINARIA:

Descripción Breve: El ordenamiento binario es un algoritmo de ordenación de tipo comparación. Es una modificación del algoritmo de ordenamiento por inserción.

ANALISIS DE EFICIENCIA:

En este algoritmo, también mantenemos una submatriz ordenada y otra sin ordenar. La única diferencia es que encontramos la posición correcta de un elemento utilizando la búsqueda binaria en lugar de la búsqueda lineal. Esto ayuda a acelerar el algoritmo de ordenación reduciendo el número de comparaciones necesarias.

ANALISIS DE LOS CASOS:

- Existen tres tipos de casos: .

Caso medio

La búsqueda binaria tiene una complejidad logarítmica logn comparada con la complejidad lineal n de la búsqueda lineal utilizada en la ordenación por inserción. Utilizamos la ordenación binaria para n elementos lo que nos da la complejidad temporal nlogn. Por tanto, la complejidad temporal es del orden de [Big Theta]: O(nlogn).

Lista	i	key	left	Right	Mid	j
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	1	-56	0	0	0	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	1	-56	0	-1		0
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	1	-56	0	-1		
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	1					
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	2	67	0	1	0	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	2	67	1	1	1	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	2	67	2	1		
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	2					
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	2	0	2	1	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	2	0	0	0	
[-56, 23, 2, 67, 9, 44, 6, 18, 1, 7]	3	2	1	0		2
[-56, 2, 23, 67, 9, 44, 6, 18, 1, 7]	3	2	1	0		1
[-56, 2, 23, 67, 9, 44, 6, 18, 1, 7]	3					
[-56, 2, 23, 67, 9, 44, 6, 18, 1, 7]	4	9	0	3	1	
[-56, 2, 23, 67, 9, 44, 6, 18, 1, 7]	4	9	2	3	2	
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	4	9	2	1		
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	4					
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	44	0	4	2	
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	44	3	4		
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	44	3	4	3	
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	44	4	4		

[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	44	4	4	4	
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	44	4	3	4	
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	44	4	3		
[-56, 2, 9, 23, 67, 67, 6, 18, 1, 7]	5	44	4	3		4
[-56, 2, 9, 23, 44, 67, 6, 18, 1, 7]	5					
[-56, 2, 9, 23, 44, 67, 6, 18, 1, 7]	6	6	0	5	2	
[-56, 2, 9, 23, 44, 67, 6, 18, 1, 7]	6	6	0	1	2	
[-56, 2, 9, 23, 44, 67, 6, 18, 1, 7]	6	6	0	1		
[-56, 2, 9, 23, 44, 67, 6, 18, 1, 7]	6	6	0	1	0	
[-56, 2, 9, 23, 44, 67, 6, 18, 1, 7]	6	6	1	1		
[-56, 2, 9, 23, 44, 67, 6, 18, 1, 7]	6	6	1	1	1	
[-56, 2, 9, 23, 44, 67, 6, 18, 1, 7]	6	6	2	1		
[-56, 2, 9, 23, 44, 67, 67, 18, 1, 7]	6	6	2	1		5
[-56, 2, 9, 23, 44, 44, 67, 18, 1, 7]	6	6	2	1		4
[-56, 2, 9, 23, 23, 44, 67, 18, 1, 7]	6	6	2	1		3
[-56, 2, 9, 9, 23, 44, 67, 18, 1, 7]	6	6	2	1		2
[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	6	6	2	1		
[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	6					
[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	7	18	0	6	3	

[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	7	18	4	6		
[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	7	18	4	6	5	
[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	7	18	4	4	5	
[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	7	18	4	4		
[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	7	18	4	4	4	
[-56, 2, 6, 9, 23, 44, 67, 18, 1, 7]	7	18	4	3	4	
[-56, 2, 6, 9, 23, 44, 67, 67, 1, 7]	7	18	4	3		6
[-56, 2, 6, 9, 23, 44, 44, 67, 1, 7]	7	18	4	3		5
[-56, 2, 6, 9, 23, 23, 44, 67, 1, 7]	7	18	4	3		4
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	7	18	4	3		
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	7					
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	8	1	0	7	3	
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	8	1	0	2	3	
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	8	1	0	2		
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	8	1	0	2	1	
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	8	1	0	0	1	
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	8	1	0	0	0	
[-56, 2, 6, 9, 18, 23, 44, 67, 1, 7]	8	1	1	0	0	7
[-56, 2, 6, 9, 18, 23, 44, 67, 67, 7]	8	1	1	0	0	7

[-56, 2, 6, 9, 18, 23, 44, 44, 67, 7]	8	1	1	0	0	6
[-56, 2, 6, 9, 18, 23, 23, 44, 67, 7]	8	1	1	0	0	5
[-56, 2, 6, 9, 18, 18, 23, 44, 67, 7]	8	1	1	0	0	4
[-56, 2, 6, 9, 9, 18, 23, 44, 67, 7]	8	1	1	0	0	3
[-56, 2, 6, 6, 9, 18, 23, 44, 67, 7]	8	1	1	0	0	2
[-56, 2, 2, 6, 9, 18, 23, 44, 67, 7]	8	1	1	0	0	1
[-56, 1, 2, 6, 9, 18, 23, 44, 67, 7]	8	1	1	0	0	
[-56, 1, 2, 6, 9, 18, 23, 44, 67, 7]	8					
[-56, 1, 2, 6, 9, 18, 23, 44, 67, 7]	9	7	0	8	4	
[-56, 1, 2, 6, 9, 18, 23, 44, 67, 7]	9	7	0	3	4	
[-56, 1, 2, 6, 9, 18, 23, 44, 67, 7]	9	7	0	3	1	
[-56, 1, 2, 6, 9, 18, 23, 44, 67, 7]	9	7	0	2	2	
[-56, 1, 2, 6, 9, 18, 23, 44, 67, 7]	9	7	3	3	3	
[-56, 1, 2, 6, 9, 18, 23, 44, 67, 67]	9	7	4	3		8
[-56, 1, 2, 6, 9, 18, 23, 44, 44, 67]	9	7	4	3		7
[-56, 1, 2, 6, 9, 18, 23, 23, 44, 67]	9	7	4	3		6
[-56, 1, 2, 6, 9, 18, 18, 23, 44, 67]	9	7	4	3		5
[-56, 1, 2, 6, 9, 9, 18, 23, 44, 67]	9	7	4	3		4
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	9	7	4	3		

[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	9					
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]						

El peor caso

El peor caso se produce cuando el array está ordenada de forma inversa y se requiere el máximo número de desplazamientos. La complejidad temporal en el peor caso es del orden de [Big O]: O(nlogn).

El mejor caso

El mejor caso se produce cuando el array ya está ordenada y no es necesario desplazar elementos. La complejidad temporal en el mejor caso es [Big Omega]: O(n).

Complejidad espacial

La complejidad espacial del algoritmo de ordenación binaria es O(n) porque no se necesita más memoria que una variable temporal.

Método de ordenamiento binario

Lista = [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

HEAPSORT:

Descripción Breve: Consiste en remover el mayor elemento que es siempre la raíz del Heap, una vez seleccionado el máximo, lo intercambiamos con el último elemento del vector, decrementamos la cantidad de elementos del Heap.

ANALISIS DE EFICIENCIA:

Un Heap puede ser almacenado en un vector sin ocasionar inconvenientes en el manejo de la estructura, evitando el uso de punteros.

El Heap se almacena en un arreglo A [1..n], y utiliza una variable que indica cuantos elementos tiene.

Este método es más lento que otros métodos, pero es más eficaz en escenarios más rigurosos.

ANALISIS DE LOS CASOS:

El ordenamiento por heap sort forma parte de los algoritmos de ordenamiento cuyas complejidades temporales son O(n log n), aún para el peor caso, a diferencia de muchos otros algoritmos de ordenamiento como los que ya hemos visto, donde para el peor caso se tiene un orden de O(n^2).

Heap sort alcanza esta complejidad temporal de O(n log n) debido esencialmente a que utiliza una estructura de datos de modo que cada operación de salida requiere a lo sumo log i pasos, donde i es el número de elementos restantes. Nótese que este inteligente diseño de estructura de datos es fundamental para el ordenamiento heap sort.

EJEMPLO:

```
function heapsort (array A[0..n]):
  montículo M
  integer i; // declaro variable i
  for i = 0..n:
    insertar_en_monticulo(M, A[i])
  for i = 0..n:
    A[i] = extraer_cima_del_monticulo(M)
  return A
```

Lista	n	i	Largest	Left	right	temp
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	4	4	9	10	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	10	3	3	7	8	
[23, -56, 67, 18, 9, 44, 6, 2, 1, 7]	10	3	7	7	8	2
[23, -56, 67, 18, 9, 44, 6, 2, 1, 7]	10	7	7	15	16	
[23, -56, 67, 18, 9, 44, 6, 2, 1, 7]	10	3	7			
[23, -56, 67, 18, 9, 44, 6, 2, 1, 7]	10	2	2	5	6	
[23, -56, 67, 18, 9, 44, 6, 2, 1, 7]	10	1				
[23, -56, 67, 18, 9, 44, 6, 2, 1, 7]	10	1	1	3	4	
[23, 18, 67, -56, 9, 44, 6, 2, 1, 7]	10	1	3	3	4	-56
[23, 18, 67, -56, 9, 44, 6, 2, 1, 7]	10	3	3	7	8	
[23, 18, 67, 2, 9, 44, 6, -56, 1, 7]	10	3	7	7	8	-56
[23, 18, 67, 2, 9, 44, 6, -56, 1, 7]	10	7	7	15	16	
[23, 18, 67, 2, 9, 44, 6, -56, 1, 7]	10	1				
[23, 18, 67, 2, 9, 44, 6, -56, 1, 7]	10	0	0	1	2	
[67, 18, 23, 2, 9, 44, 6, -56, 1, 7]	10	0	2	1	2	23
[67, 18, 23, 2, 9, 44, 6, -56, 1, 7]	10	2	2	5	6	
[67, 18, 44, 2, 9, 23, 6, -56, 1, 7]	10	2	5	5	6	23
[67, 18, 44, 2, 9, 23, 6, -56, 1, 7]	10	5	5	11	12	
[67, 18, 44, 2, 9, 23, 6, -56, 1, 7]	10	0				
[7, 18, 44, 2, 9, 23, 6, -56, 1, 67]	10	9	67			
[7, 18, 44, 2, 9, 23, 6, -56, 1, 67]	9	0	0	1	2	

[7, 18, 44, 2, 9, 23, 6, -56, 1, 67]	9	0	1	1	2	
[44, 18, 7, 2, 9, 23, 6, -56, 1, 67]	9	0	2	1	2	7
[44, 18, 7, 2, 9, 23, 6, -56, 1, 67]	9	2	2	5	6	
[44, 18, 23, 2, 9, 7, 6, -56, 1, 67]	9	2	5	5	6	7
[44, 18, 23, 2, 9, 7, 6, -56, 1, 67]	9	5	5	11	12	
[44, 18, 23, 2, 9, 7, 6, -56, 1, 67]	10	9				
[1, 18, 23, 2, 9, 7, 6, -56, 44, 67]	10	8				44
[1, 18, 23, 2, 9, 7, 6, -56, 44, 67]	8	0	0	1	2	
[1, 18, 23, 2, 9, 7, 6, -56, 44, 67]	8	0	1	1	2	
[23, 18, 1, 2, 9, 7, 6, -56, 44, 67]	8	0	2	1	2	1
[23, 18, 1, 2, 9, 7, 6, -56, 44, 67]	8	2	2	5	6	
[23, 18, 7, 2, 9, 1, 6, -56, 44, 67]	8	2	5	5	6	1
[23, 18, 7, 2, 9, 1, 6, -56, 44, 67]	8	5	5	11	12	
[23, 18, 7, 2, 9, 1, 6, -56, 44, 67]	10	8				
[-56, 18, 7, 2, 9, 1, 6, 23, 44, 67]	10	7				23
[-56, 18, 7, 2, 9, 1, 6, 23, 44, 67]	7	0	0	1	2	
[18, -56, 7, 2, 9, 1, 6, 23, 44, 67]	7	0	1	1	2	-56
[18, -56, 7, 2, 9, 1, 6, 23, 44, 67]	7	1	1	3	4	
[18, -56, 7, 2, 9, 1, 6, 23, 44, 67]	7	1	3			

[18, 9, 7, 2, -56, 1, 6, 23, 44, 67]	7	1	4			-56
[18, 9, 7, 2, -56, 1, 6, 23, 44, 67]	7	4	4	9	10	
[18, 9, 7, 2, -56, 1, 6, 23, 44, 67]	10	7				
[6, 9, 7, 2, -56, 1, 18, 23, 44, 67]	10	6				18
[6, 9, 7, 2, -56, 1, 18, 23, 44, 67]	6	0	0	1	2	
[9, 6, 7, 2, -56, 1, 18, 23, 44, 67]	6	0	1	1	2	6
[9, 6, 7, 2, -56, 1, 18, 23, 44, 67]	6	1	1	3	4	
[9, 6, 7, 2, -56, 1, 18, 23, 44, 67]	10	6				
[1, 6, 7, 2, -56, 9, 18, 23, 44, 67]	10	5				9
[1, 6, 7, 2, -56, 9, 18, 23, 44, 67]	5	0	0	1	2	
[1, 6, 7, 2, -56, 9, 18, 23, 44, 67]	5	0	1	1	2	
[7, 6, 1, 2, -56, 9, 18, 23, 44, 67]	5	0	2	1	2	1
[7, 6, 1, 2, -56, 9, 18, 23, 44, 67]	5	2	2	5	6	
[7, 6, 1, 2, -56, 9, 18, 23, 44, 67]	10	5				
[-56, 6, 1, 2, 7, 9, 18, 23, 44, 67]	10	4				7
[-56, 6, 1, 2, 7, 9, 18, 23, 44, 67]	4	0	0	1	2	
[6, -56, 1, 2, 7, 9, 18, 23, 44, 67]	4	0	1	1	2	-56
[6, -56, 1, 2, 7, 9, 18, 23, 44, 67]	4	1	1	3	4	
[6, 2, 1, -56, 7, 9, 18, 23, 44, 67]	4	1	3	3	4	-56

[6, 2, 1, -56, 7, 9, 18, 23, 44, 67]	4	3	3	7	8	
[6, 2, 1, -56, 7, 9, 18, 23, 44, 67]	10	4				
[-56, 2, 1, 6, 7, 9, 18, 23, 44, 67]	10	3				6
[-56, 2, 1, 6, 7, 9, 18, 23, 44, 67]	3	0	0	1	2	
[2, -56, 1, 6, 7, 9, 18, 23, 44, 67]	3	0	1	1	2	-56
[2, -56, 1, 6, 7, 9, 18, 23, 44, 67]	3	1	1	3	4	
[2, -56, 1, 6, 7, 9, 18, 23, 44, 67]	10	3				
[1, -56, 2, 6, 7, 9, 18, 23, 44, 67]	10	2				2
[1, -56, 2, 6, 7, 9, 18, 23, 44, 67]	2	0	0	1	2	2
[1, -56, 2, 6, 7, 9, 18, 23, 44, 67]	10	2				
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10	1				1
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	1	0	0	1	2	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10	1				
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	10					

Prueba de escritorio método HeapSort

Lista = [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

QUICKSORT RECURSIVO:

Descripción Breve: Este método se basa en la táctica "divide y

vencerás", que consiste en ir subdividiendo el arreglo en arreglos más pequeños, y

ordenar éstos. Para hacer esta división, se toma un valor del arreglo como pivote, y se

mueven todos los elementos menores que este pivote a su izquierda, y los mayores a

su derecha.

A continuación, se aplica el mismo método a cada una de las dos partes en

las que queda dividido el arreglo.

ANALISIS DE EFICIENCIA:

Normalmente se toma como pivote el primer elemento de arreglo, y se realizan dos

búsquedas: una de izquierda a derecha, buscando un elemento mayor que el pivote, y

otra de derecha a izquierda, buscando un elemento menor que el pivote. Cuando se

han encontrado los dos, se intercambian, y se sigue realizando la búsqueda hasta que

las dos búsquedas se encuentran.

ANALISIS DE LOS CASOS:

- Existen tres tipos de casos:

El mejor caso de quick sort: ocurre cuando X divide la lista justo en el centro. Es decir, X produce dos sublistas que contienen el mismo número de elementos.

Lista	low	high	Pivot/index	I	j	temp
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]						
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	9	7	-1	0	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	9	7	0	1	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]				0	1	23
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	9	7	0	2	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	9	7	1	3	
[-56, 2, 67, 23, 9, 44, 6, 18, 1, 7]				1	3	23
[-56, 2, 67, 23, 9, 44, 6, 18, 1, 7]	0	9	7	1	3	
[-56, 2, 67, 23, 9, 44, 6, 18, 1, 7]	0	9	7	1	4	
[-56, 2, 67, 23, 9, 44, 6, 18, 1, 7]	0	9	7	1	5	
[-56, 2, 67, 23, 9, 44, 6, 18, 1, 7]	0	9	7	2	6	
[-56, 2, 6, 23, 9, 44, 67, 18, 1, 7]				2	6	67
[-56, 2, 6, 23, 9, 44, 67, 18, 1, 7]	0	9	7	2	7	
[-56, 2, 6, 23, 9, 44, 67, 18, 1, 7]	0	9	7	3	8	
[-56, 2, 6, 1, 9, 44, 67, 18, 23, 7]				3	8	23
[-56, 2, 6, 1, 9, 44, 67, 18, 23, 7]	0	9	7	3	8	
[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]				4	9	9
[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]	0	9	7	3		
[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]	0	9	4			
[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]	0	3				
[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]	0	3	1	-1	0	

[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]	0	3	1	0	0	
[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]				0	0	-56
[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]	0	3	1	0	1	
[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]	0	3	1	0	2	
[-56, 2, 6, 1, 7, 44, 67, 18, 23, 9]	0	3	1	0		
[-56, 1, 6, 2, 7, 44, 67, 18, 23, 9]				1	3	2
[-56, 1, 6, 2, 7, 44, 67, 18, 23, 9]	0	3	1	0		
[-56, 1, 6, 2, 7, 44, 67, 18, 23, 9]	0	3	1			
[-56, 1, 6, 2, 7, 44, 67, 18, 23, 9]	0	0				
[-56, 1, 6, 2, 7, 44, 67, 18, 23, 9]	0	3	1			
[-56, 1, 6, 2, 7, 44, 67, 18, 23, 9]	2	3	1			
[-56, 1, 6, 2, 7, 44, 67, 18, 23, 9]	2	3	2	1	2	
[-56, 1, 6, 2, 7, 44, 67, 18, 23, 9]	2	3	2	1		
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	2	3	2			
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	2	1				
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	2	3	2			
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	3	3				
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	0	9	4			
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	5	9	9	4	5	

[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	5	9	9	4	6	
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	5	9	9	4	7	
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	5	9	9	4	8	
[-56, 1, 2, 6, 7, 44, 67, 18, 23, 9]	5	9	9	4		
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	5	9	9	5	9	44
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	5	9	9	4		
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	5	9	5			
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	5	6				
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	6	9	44	5	6	
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	6	9	44	5	7	
[-56, 1, 2, 6, 7, 9, 67, 18, 23, 44]	6	9	44	6	7	
[-56, 1, 2, 6, 7, 9, 18, 67, 23, 44]				6	7	67
[-56, 1, 2, 6, 7, 9, 18, 67, 23, 44]	6	9	44	6	7	
[-56, 1, 2, 6, 7, 9, 18, 67, 23, 44]	6	9	44	6	8	
[-56, 1, 2, 6, 7, 9, 18, 67, 23, 44]	6	9	44	7	8	
[-56, 1, 2, 6, 7, 9, 18, 23, 67, 44]				7	8	67
[-56, 1, 2, 6, 7, 9, 18, 23, 67, 44]	6	9	44	7	8	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]				8	9	67
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	9	44	7		

[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	9	8			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	7	23	5	6	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]				6	6	18
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	7	23	6	6	
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	7	23	6		
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]				7	7	23
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	7	23	6		
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	7		7		
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	6				
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	7	7			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	8	7				
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6	9	8			
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	9	9				

En este caso, la primera ronda requiere n pasos para dividir la lista original en dos listas. Para la ronda siguiente, para cada sublista, de nuevo se necesitan n/2 pasos (ignorando el elemento usado para la división).

En consecuencia, para la segunda ronda nuevamente se requieren 2 • (n/2) = n pasos.

Si se supone que n = 2^P, entonces en total se requieren p*n pasos. Sin embargo, p = log2 n. Así, para el mejor caso, la complejidad temporal del quick sort es O(n log n).

El peor caso del quick sort ocurre cuando los datos de entrada están ya ordenados o inversamente ordenados.

En estos casos, todo el tiempo simplemente se está seleccionando el extremo (ya sea el mayor o el menor). Por lo tanto, el número total de pasos que se requiere en el quick sort para el peor caso es:

n + (n - 1) + ••• + 1 = n/2 (n + 1) = O(n^2).

Para analizar el caso promedio, sea T(n) que denota el número de pasos necesarios para llevar a cabo el quick sort en el caso promedio para n elementos. Se supondrá que después de la operación de división la lista se ha dividido en dos sublistas. La primera de ellas contiene s elementos y la segunda contiene (n - s) elementos.

El valor de s varía desde 1 hasta n y es necesario tomar en consideración todos los casos posibles a fin de obtener el desempeño del caso promedio.
Prueba de escritorio Quicksort recursivo

Lista = [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

RADIX:

Descripción Breve: Es un algoritmo de ordenamiento que ordena enteros procesando sus dígitos de forma individual, cuenta con una única restricción, que es que no acepta números negativos

ANALISIS DE EFICIENCIA:

Este algoritmo evita las comparaciones insertando elementos en cubos de acuerdo con el radix (Radix/Base es el número de dígitos únicos utilizados para representar números.

Por ejemplo, los números decimales tienen diez dígitos únicos). Ordena los elementos basándose en los dígitos de los elementos individuales. Realiza la ordenación por conteo de los dígitos desde el menos significativo hasta el

más significativo. También se ha llamado ordenación en cubo o digital y es muy útil en máquinas paralelas

ANALISIS DE LOS CASOS:

- Caso medio:

La ordenación radix tiene una complejidad temporal de O(n + b) donde b es el rango de entrada. Si hay d dígitos en el elemento máximo maxm, la complejidad temporal de Radix Sort es O(d*(n + b)). Como d y b suelen ser pequeños, la complejidad temporal es del orden de [Big Theta]: O(n).

El peor caso:

La complejidad temporal en el peor de los casos es del orden de [Big O]: O(n).

El mejor caso:

La complejidad temporal en el mejor de los casos es [Big Omega]: O(n). Es la misma que la complejidad temporal del peor caso.

Complejidad espacial

La complejidad espacial del algoritmo de Ordenamiento Radix es O(n+b), donde b es el rango de entrada. Proviene de los Arrays count &output en la ordenación radix.

A veces b puede ser mayor que n, lo que hace que la complejidad no sea lineal.

Para el ejemplo el Vector original es:

25 57 48 37 12 92 86 33

Asignamos los elementos en colas basadas en el dígito menos significativo de cada uno de ellos.

0:

1:

2:12 92

3:33

4:

5:25

6:86

7:57 37

8:48

9:

Después de la primera pasada, la ordenación queda:

12 92 33 25 86 57 37 48

Colas basadas en el dígito más significativo.

0:

1:12

2:25

3:33 37

4:48

5:57

6:

7:

8:86

9:92

Lista ordenada:

12 25 33 37 48 57 86 92

Prueba de escritorio Radix

Lista = [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

Lista	Max	i	exp	N	digit	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	23	1				
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	23	2				
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	67	2				
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	67	3				
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	67	4				

……A partir de aquí lo único que cambia es la i hasta la posición 9

[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	67	9				
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	67					
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	67		1	10		
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	67	0	1	10		
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	67	1	1	10		

……A partir de aquí lo único que cambia es la i hasta la posición 9

[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	67	9	1	10		
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7] [0, 0, 0, 1, 0, 0, 0, 0, 0]	67	0	1	10	3	

[23, -56, 67, 2, 9, 44, 6, 18, 1, 7] [0, 0, 0, 1, 0, 0, 0, 0, 0]	67	1	1	10	-6	

En este método sale error al intentar acomodar el arreglo ya que no acepta números negativos

Iteracion

Por intercalación de archivos se entiende la unión o fusión de dos o más archivos, previamente ordenados, en un solo archivo, el cual debe quedar ordenado al hacer la intercalación.

Si se cuenta con dos archivos con datos previamente ordenados, el proceso de intercalación entre los dos archivos, consiste en extraer el primer elemento de cada archivo y determinar cuál es el menor, para colocarlo en el tercer archivo, extraer el siguiente elemento del archivo y compararlo nuevamente contra el otro elemento que ya se tenía del otro archivo, para determinar cuál ingresa al tercer archivo, este proceso se repita hasta que uno de los archivos originales llegue hasta el fin, en este caso, solo resta transcribir los números del archivo que no se ha llegado a su fin al tercer archivo.

Prueba de escritorio método externo Intercalación

Lista = [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

Lista	left	right	mid	i	j	k	Tem	x
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	9	4					
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2					
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1					
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0					
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	0						
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0					
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	1	1	0					
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	0	1	0	[-56, 0]	
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	0	2	1	[-56, 23]	
[-56, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	1	2	2	[-56, 23]	0
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	1	2	2	[-56, 23]	1
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	1						
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1					
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	2	2						
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	0	2	0	[0, 0, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	0	2	0	[-56, 0, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	1	2	1	[-56, 23, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	2	2	2	[-56, 23, 67]	

[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	2	3	3	[-56, 23, 67]	0
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	2	3	3	[-56, 23, 67]	1
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	2	3	3	[-56, 23, 67]	2
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2					
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3					
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3					
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	4	4						
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	3	4	0	[0, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	4	4	1	[2, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	4	5	2	[2, 9]	0
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	4	5	2	[2, 9]	1
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4						
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	0	3	0	[0, 0, 0, 0, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	0	3	0	[-56, 0, 0, 0, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	1	3	1	[-56, 2, 0, 0, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	1	4	2	[-56, 2, 9, 0, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	1	5	3	[-56, 2, 9, 23, 0]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	2	5	4	[-56, 2, 9, 23, 67]	
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	3	5	5	[-56, 2, 9, 23, 67]	0

[-56, 2, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	3	5	5	[-56, 2, 9, 23, 67]	1
[-56, 2, 9, 2, 9, 44, 6, 18, 1, 7]	0	4	2	3	5	5	[-56, 2, 9, 23, 67]	2
[-56, 2, 9, 23, 9, 44, 6, 18, 1, 7]	0	4	2	3	5	5	[-56, 2, 9, 23, 67]	3
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	0	4	2	3	5	5	[-56, 2, 9, 23, 67]	4
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	0	4						
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	0	9	4					
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	9	7					
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	7	6					
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5					
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	5						
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5					
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	6	6						
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5	5	6	0	[0, 0]	
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5	5	6	0	[6, 0]	
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5	5	7	1	[6, 44]	
[-56, 2, 9, 23, 67, 6, 6, 18, 1, 7]	5	6	5	6	7	2	[6, 44]	0
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	6	5	6	7	2	[6, 44]	1
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	6						
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6					

[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	7	7						
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6	5	7	0	[0, 0, 0]	
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6	5	7	0	[6, 0, 0]	
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6	6	7	1	[6, 18, 0]	
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6	6	8	2	[6, 18, 44]	
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6	7	8	3	[6, 18, 44]	0
[-56, 2, 9, 23, 67, 6, 18, 18, 1, 7]	5	7	6	7	8	3	[6, 18, 44]	1
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	7	6	7	8	3	[6, 18, 44]	2
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	7						
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7					
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8					
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	8						
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8					
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	9	9						
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8				[0, 0]	
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8	8	9	0	[1, 0]	
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8	9	9	1	[1, 7]	
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8	9	10	2	[1, 7]	0
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8	9	10	2	[1, 7]	1

[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9						
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7				[0, 0, 0, 0, 0]	
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7	5	8	0	[1, 0, 0, 0, 0]	
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7	5	9	1	[1, 6, 0, 0, 0]	
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7	6	9	2	[1, 6, 7, 0, 0]	
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7	6	10	3	[1, 6, 7, 18, 0]	
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7	7	10	4	[1, 6, 7, 18, 44]	
[-56, 2, 9, 23, 67, 1, 18, 44, 1, 7]	5	9	7	8	10	5	[1, 6, 7, 18, 44]	0
[-56, 2, 9, 23, 67, 1, 6, 44, 1, 7]	5	9	7	8	10	5	[1, 6, 7, 18, 44]	1
[-56, 2, 9, 23, 67, 1, 6, 7, 1, 7]	5	9	7	8	10	5	[1, 6, 7, 18, 44]	2
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 7]	5	9	7	8	10	5	[1, 6, 7, 18, 44]	3
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	5	9	7	8	10	5	[1, 6, 7, 18, 44]	4
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	5	9	4					
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4				[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	0	5	0	[-56, 0, 0, 0, 0, 0, 0, 0, 0, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	1	5	1	[-56, 1, 0, 0, 0, 0, 0, 0, 0, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	1	6	2	[-56, 1, 2, 0, 0, 0, 0, 0, 0, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	2	6	3	[-56, 1, 2, 6, 0, 0, 0, 0, 0, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	2	7	4	[-56, 1, 2, 6, 7, 0, 0, 0, 0, 0]	

[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	2	8	5	[-56, 1, 2, 6, 7, 0, 0, 0, 0, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	2	8	5	[-56, 1, 2, 6, 7, 9, 0, 0, 0, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	3	8	6	[-56, 1, 2, 6, 7, 9, 18, 0, 0, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	3	9	7	[-56, 1, 2, 6, 7, 9, 18, 23, 0, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	4	9	8	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 0]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	4	10	9	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	0
[-56, 1, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	1
[-56, 1, 2, 23, 67, 1, 6, 7, 18, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	2
[-56, 1, 2, 6, 67, 1, 6, 7, 18, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	3
[-56, 1, 2, 6, 7, 1, 6, 7, 18, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	4
[-56, 1, 2, 6, 7, 9, 6, 7, 18, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	5
[-56, 1, 2, 6, 7, 9, 18, 7, 18, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	6
[-56, 1, 2, 6, 7, 9, 18, 23, 18, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	7
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 44]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	8
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	0	9	4	5	10	10	[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	9
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	0	9						

Mezcla Directa

Mezcla Directa

El método de ordenación por mezcla directa es probablemente el más utilizado por su fácil comprensión.

La idea central de este algoritmo consiste en la realización sucesiva de una partición y una fusión que produce secuencias ordenadas de longitud cada vez mayor. En la primera pasada la partición es de longitud 1, y la fusión o mezcla produce secuencias ordenadas de longitud 2. En la segunda pasada la partición es de longitud 2, y la fusión o mezcla produce secuencias ordenadas de longitud 4. Este proceso se repite hasta que la longitud de la secuencia de la secuencia para la partición sea mayor o igual que el número de elementos del archivo original.

Método de ordenamiento externo por mezcla directa

Lista= [23, -56, 67, 2, 9, 44, 6, 18, 1, 7]

Lista	left	right	mid	i	j	k	N1	N2	LeftArr	RightArr
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	9								
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2							
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1							
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0							
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	0								
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0							
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	1	1								
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0							
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	0			1	1	[0]	[0]
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	0			1	1	[23]	[0]
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0		0		1	1	[23]	[-56]
[23, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	0	0	0	1	1	[23]	[-56]
[-56, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	0	0	0	1	1	[23]	[-56]
[-56, -56, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	0	1	1	1	1	[23]	[-56]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	1	1	1	1	1	[23]	[-56]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	1	0	1	1	2	1	1	[23]	[-56]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1							
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	2	2								

[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1				2	1	[0, 0]	[0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	0			2	1	[-56, 0]	[0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	1			2	1	[-56, 23]	[0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1		0		2	1	[-56, 23]	[67]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1				2	1	[-56, 23]	[67]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	0	0	0	2	1	[-56, 23]	[67]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	1	0	1	2	1	[-56, 23]	[67]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	2	0	2	2	1	[-56, 23]	[67]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2	1	2	1	3	2	1	[-56, 23]	[67]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	2								
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2							
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3							
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	3								
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3							
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	4	4								
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3				1	1	[0]	[0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	0			1	1	[2]	[0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3		0		1	1	[2]	[9]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	0	0	3	1	1	[2]	[9]

[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	1	0	3	1	1	[2]	[9]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	1	0	4	1	1	[2]	[9]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	1	1	4	1	1	[2]	[9]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4	3	1	1	5	1	1	[2]	[9]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	3	4								
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	0			3	2	[0, 0, 0]	[0, 0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	0			3	2	[-56, 0, 0]	[0, 0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	1			3	2	[-56, 23, 0]	[0, 0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	2			3	2	[-56, 23, 67]	[0, 0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2		0		3	2	[-56, 23, 67]	[2, 0]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2		1		3	2	[-56, 23, 67]	[2, 9]
[-56, 23, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	0	0	0	3	2	[-56, 23, 67]	[2, 9]
[-56, 2, 67, 2, 9, 44, 6, 18, 1, 7]	0	4	2	1	0	1	3	2	[-56, 23, 67]	[2, 9]
[-56, 2, 9, 2, 9, 44, 6, 18, 1, 7]	0	4	2	1	1	2	3	2	[-56, 23, 67]	[2, 9]
[-56, 2, 9, 23, 9, 44, 6, 18, 1, 7]	0	4	2	1	2	3	3	2	[-56, 23, 67]	[2, 9]
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	0	4	2	2	2	4	3	2	[-56, 23, 67]	[2, 9]
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	0	4	2	3	2	5	3	2	[-56, 23, 67]	[2, 9]
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	0	4								
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	0	9	4							

[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	9	7							
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	7	6							
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5							
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	5								
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5							
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	6	6								
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5				1	1	[0]	[0]
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5	0			1	1	[44]	[0]
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5		0		1	1	[44]	[6]
[-56, 2, 9, 23, 67, 44, 6, 18, 1, 7]	5	6	5	0	0	5	1	1	[44]	[6]
[-56, 2, 9, 23, 67, 6, 6, 18, 1, 7]	5	6	5	0	0	5	1	1	[44]	[6]
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	6	5	0	1	6	1	1	[44]	[6]
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	6	5	1	1	7	1	1	[44]	[6]
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	6								
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6							
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	7	7								
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6							
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6				2	1		
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6				2	1	[0, 0]	[0]

[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6	0			2	1	[6, 0]	[0]
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6	1			2	1	[6, 44]	[0]
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6		0		2	1	[6, 44]	[18]
[-56, 2, 9, 23, 67, 6, 44, 18, 1, 7]	5	7	6	0	0	5	2	1	[6, 44]	[18]
[-56, 2, 9, 23, 67, 6, 18, 18, 1, 7]	5	7	6	1	0	6	2	1	[6, 44]	[18]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	7	6	1	1	7	2	1	[6, 44]	[18]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	7	6	2	1	8	2	1	[6, 44]	[18]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	7								
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7							
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8							
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	8								
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8							
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	9	9								
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8							
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8				1	1		
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8				1	1	[0]	[0]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8	0			1	1	[1]	[0]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8		0		1	1	[1]	[7]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8	0	0	8	1	1	[1]	[7]

[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8	1	1	9	1	1	[1]	[7]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9	8	1	1	10	1	1	[1]	[7]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	8	9								
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7							
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7				3	2	[0, 0, 0]	[0, 0]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7	0			3	2	[6, 0, 0]	[0, 0]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7	1			3	2	[6, 18, 0]	[0, 0]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7	2			3	2	[6, 18, 44]	[0, 0]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7		0		3	2	[6, 18, 44]	[1, 0]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7		1		3	2	[6, 18, 44]	[1, 7]
[-56, 2, 9, 23, 67, 6, 18, 44, 1, 7]	5	9	7	0	0	5	3	2	[6, 18, 44]	[1, 7]
[-56, 2, 9, 23, 67, 1, 18, 44, 1, 7]	5	9	7	0	0	5	3	2	[6, 18, 44]	[1, 7]
[-56, 2, 9, 23, 67, 1, 6, 44, 1, 7]	5	9	7	0	1	6	3	2	[6, 18, 44]	[1, 7]
[-56, 2, 9, 23, 67, 1, 6, 7, 1, 7]	5	9	7	1	1	7	3	2	[6, 18, 44]	[1, 7]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 7]	5	9	7	1	2	8	3	2	[6, 18, 44]	[1, 7]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	5	9	7	2	2	9	3	2	[6, 18, 44]	[1, 7]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	5	9	7	3	2	10	3	2	[6, 18, 44]	[1, 7]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	5	9								
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4				5	5	[0, 0, 0, 0, 0]	[0, 0, 0, 0, 0]

[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	0			5	5	[-56, 0, 0, 0, 0]	[0, 0, 0, 0, 0]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	1			5	5	[-56, 2, 0, 0, 0]	[0, 0, 0, 0, 0]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	2			5	5	[-56, 2, 9, 0, 0]	[0, 0, 0, 0, 0]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	3			5	5	[-56, 2, 9, 23, 0]	[0, 0, 0, 0, 0]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	4			5	5	[-56, 2, 9, 23, 67]	[0, 0, 0, 0, 0]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4		0		5	5	[-56, 2, 9, 23, 67]	[1, 0, 0, 0, 0]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4		1		5	5	[-56, 2, 9, 23, 67]	[1, 6, 0, 0, 0]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4		2		5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 0, 0]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4		3		5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 0]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4		4		5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 2, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	0	0	0	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 9, 23, 67, 1, 6, 7, 18, 44]	0	9	4	1	0	1	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 2, 23, 67, 1, 6, 7, 18, 44]	0	9	4	1	1	2	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 2, 6, 67, 1, 6, 7, 18, 44]	0	9	4	2	1	3	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 2, 6, 7, 1, 6, 7, 18, 44]	0	9	4	2	2	4	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 2, 6, 7, 9, 6, 7, 18, 44]	0	9	4	2	3	5	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 2, 6, 7, 9, 18, 7, 18, 44]	0	9	4	3	3	6	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 2, 6, 7, 9, 18, 23, 18, 44]	0	9	4	3	4	7	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 44]	0	9	4	4	4	8	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]

[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	0	9	4	4	5	9	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	0	9	4	5	5	10	5	5	[-56, 2, 9, 23, 67]	[1, 6, 7, 18, 44]
[-56, 1, 2, 6, 7, 9, 18, 23, 44, 67]	0	9								

CONCLUSIONES:

Conclusión 1:

El método HeapSort es más eficiente en los arreglos con más datos, ya que por ser un ordenamiento de montículo tiende a demorarse más con arreglos pequeños por contar con datos repetidos.

Conclusión 2:

El método Radix Sort a pesar de ser un método rápido puede demorar un poco su proceso en ciertos números de datos, ya que este ordena uno por uno los dígitos de cada dato.

Conclusión 3:

Los métodos más rápidos fueron Quicksort, HeapSort y HeapSort gracias su complejidad y a que tienen pocos ciclos en su proceso de ordenado.

Conclusión 4:

Los 2 tipos de ordenamientos óptimos según la estructura de datos a utilizar son:

los internos que: el tiempo que se requiere para acceder a cualquier elemento sea el mismo, este ordenamiento se aplica cuando el conjunto de datos a clasificar es lo suficientemente pequeño.

los externos que: en soportes de almacenamiento masivo (cintas o discos) el tiempo de acceso a lectura y escritura influye en la eficiencia del ordenamiento, por lo que se asume que el tiempo que se requiere para acceder a cualquier elemento depende de la última posición accesada.

CONCLUSION GENERAL:

Los métodos de ordenamiento de datos son muy útiles, ya que la forma de arreglar los registros de una tabla en algún orden secuencial de acuerdo a un criterio de ordenamiento, el cual puede ser numérico, alfabético o alfanumérico, ascendente o descendente.

Nos facilita las búsquedas de cantidades de registros en un moderado tiempo, en modelo de eficiencia. Mediante sus técnicas podemos colocar listas detrás de otras y luego ordenarlas, como también podemos comparar pares de valores de llaves, e intercambiarlos si no se encuentran en sus posiciones correctas.