Instituto Politécnico Nacional
La Técnica al Servicio de la Patria

Unidad Profesional Interdisciplinaria de Ingeniería Campus Tlaxcala UPIIT

Algoritmos y Estructuras de Datos

Esaú Eliezer Escobar Juárez

Ingeniería en Inteligencia Artificial (IIA)

Métodos de Ordenamiento

Concepto de ordenación

- La ordenación de los datos consiste en disponer o clasificar un conjunto de datos (o una estructura) en algún determinado orden con respecto a alguno de sus campos.

  Orden: Relación de una cosa con respecto a otra.
  Clave: Campo por el cual se ordena.

- Una lista de datos está ordenada por la clave k si la lista está en orden con respecto a la clave anterior. Este Orden puede ser:
  - Ascendente: (i<j) entonces (k[i]<=k[j])
  - Descendente: (i>j) entonces (k[i]>=k[j])

Tipos de Ordenamiento

- Ordenamiento interno: Se lleva a cabo completamente en memoria principal. Todos los objetos que se ordenan caben en la memoria principal de la computadora (RAM).

- Ordenamiento externo: No cabe toda la información en memoria principal y es necesario ocupar memoria secundaria. El ordenamiento ocurre transfiriendo bloques de información a memoria principal en donde se ordena el bloque y este es regresado, ya ordenado, a memoria secundaria. Implica el manejo de archivos.

Criterios de Eficiencia

- El número de pasos.
- El número de comparaciones entre llaves para ordenar n registros. Se utiliza cuando la comparación entre llaves es costosa.
- El número de movimientos o intercambios de registros que se requieren para ordenar n registros. Se usa cuando el movimiento de registros es costoso.

Algoritmos de Ordenamiento por Intercambio

- Métodos directos o cuadráticos (n^2):
  - Intercambio directo (burbuja)
  - Sacudida (Shaker Sort)
  - Selección directa
  - Inserción:
    - Directa
    - Binaria
- Métodos logarítmicos (n* log n):
  - Shell Sort
  - QuickSort
  - HeapSort

Intercambio Directo (burbuja)

- El método de intercambio directo puede trabajar de dos maneras diferentes:
  - Llevando los elementos más pequeños hacia la parte izquierda del arreglo.
  - Llevando los elementos más grandes hacia la parte derecha del mismo.
- La idea básica de este algoritmo consiste en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que todos se encuentran ordenados.
- Se realizan (n-1) pasadas, transportando en cada una de las mismas el menor o mayor elemento (según sea el caso) a su posición ideal. Al final de las (n-1) pasadas los elementos del arreglo estarán ordenados.

Ejemplo burbuja

Supóngase que desean ordenar las siguientes claves del arreglo A, transportando en cada pasada el menor elemento hacia la parte izquierda del arreglo.

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son las siguientes:

PRIMERA PASADA

a[6]>a[7] (12>35) no hay intercambio

a[5]>a[6] (27>12) si hay intercambio

a[4]>a[5] (44>12) si hay intercambio

a[3]>a[4] (16>12) si hay intercambio

a[2]>a[3] (08>12) no hay intercambio

a[1]>a[2] (67>08) si hay intercambio

a[0]>a[1] (15>08) si hay intercambio

A: 08 15 67 12 16 44 27 35

Obsérvese que el elemento 08 fue situado en la parte izquierda del arreglo.

Ejemplo burbuja

SEGUNDA PASADA :

a[6]>a[7] (27>35) no hay intercambio

a[5]>a[6] (44>27) si hay intercambio

a[4]>a[5] (16>27) no hay intercambio

a[3]>a[4] (12>16) no hay intercambio

a[2]>a[3] (67>12) si hay intercambio

a[1]>a[2] (15>12) si hay intercambio

A: 08 12 15 67 16 27 44 35

y el segundo elemento más pequeño del arreglo, en este caso 12, fue situado en la segunda posición.

A continuación se muestra el resultado de las siguientes pasadas:

3era. Pasada : 08 12 15 16 67 27 35 44

4ta. Pasada : 08 12 15 16 27 67 35 44

5ta. pasada : 08 12 15 16 27 35 67 44

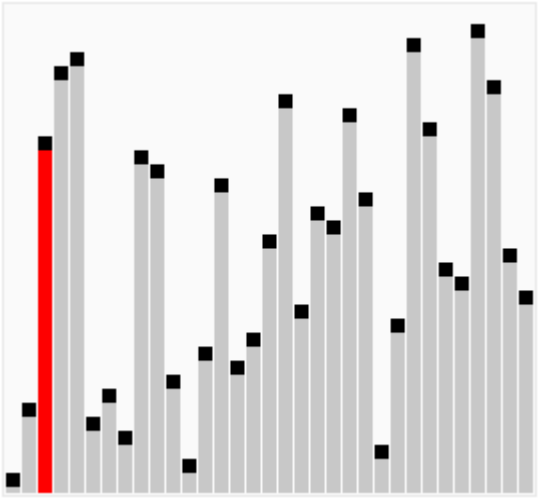
6ta. Pasada : 08 12 15 16 27 35 44 67

7ma. Pasada : 08 12 15 16 27 35 44 67

Consideraciones burbuja

- El proceso para transportar el elemento más grande hacia la derecha es similar al anterior, el recorrido del arreglo se hace en orden inverso y la comparación cambia.

- El método se puede optimizar, deteniendo el proceso si no existen intercambios en una pasada, esto indica que el arreglo ya esta ordenado.



- https://www.youtube.com/watch?v=lyZQPjUT5B4

Análisis de eficiencia del método de la burbuja

Comparaciones: $C = (n-1) + (n-2) + \ldots + 2 + 1 = (n*(n-1))/2$

$$C = (n^2 - n)/2$$

Fómula de la sumatoria

Movimientos:

$M_{min} = 0$

$M_{med} = 0.75 * (n^2 - n)$

$M_{máx} = 1.5 * (n^2 - n)$

En el peor de los casos son 3 operaciones por movimiento

Ejemplo, para ordenar 500 elementos:

Si el arreglo se encuentra ordenado :

124 750 comparaciones y 0 movimientos

Si los elementos del arreglo se encuentran dispuestos en forma aleatoria :

124 750 comparaciones y 187 125 movimientos

Si los elementos del arreglo se encuentran en orden inverso:

124 750 comparaciones y 374 250 movimientos

Método del Shaker Sort (Sacudida)

- Es una optimización del método de intercambio directo. La idea básica de este algoritmo consiste en mezclar las dos formas en que se puede realizar el método de la burbuja.
- El algoritmo tiene 2 etapas:
  - De derecha a izquierda: se trasladan los elementos más pequeños hacia la parte izquierda del arreglo, almacenando en una variable la posición del último elemento intercambiado.
  - De izquierda a derecha: se trasladan los elementos más grandes hacia la parte derecha del arreglo, almacenando en otra variable la posición del último elemento intercambiado.
  - Las sucesivas pasadas trabajan con los componentes del arreglo comprendidos entre las posiciones almacenadas en las variables.
  - El algoritmo termina cuando en una etapa no se producen intercambios o bien, cuando el contenido de la variable que almacena el extremo izquierdo del arreglo es mayor que el contenido de la variable que almacena el extremo derecho.

Ejemplo Shaker Sort (Sacudida)

Arreglo: 15 67 08 16 44 27 12 35

PRIMERA PASADA

Primera etapa (de derecha a izquierda).

A[6]>A[7] (12>35) no hay intercambio
A[5]>A[6] (27>12) sí hay intercambio
A[4]>A[5] (44>12) sí hay intercambio
A[3]>A[4] (16>12) sí hay intercambio
A[2]>A[3] (08>12) no hay intercambio
A[1]>A[2] (67>08) sí hay intercambio
A[0]>A[1] (15>08) sí hay intercambio

Última posición de intercambio de derecha a izquierda : 1

A: 08 15 67 12 16 44 27 35

Segunda etapa (de izquierda a derecha)

A[1]>A[2] (15>67) no hay intercambio
A[2]>A[3] (67>12) si hay intercambio
A[3]>A[4] (67>16) si hay intercambio
A[4]>A[5] (67>44) si hay intercambio
A[5]>A[6] (67>27) si hay intercambio
A[6]>A[7] (67>35) si hay intercambio

Última posición de intercambio de izquierda a derecha : 7

Ejemplo Shaker Sort (Sacudida)

A: 08 15 12 16 44 27 35 67

SEGUNDA PASADA

Primera etapa (derecha a izquierda)

A[5]>A[6] (27>35) no hay intercambio
A[4]>A[5] (44>27) sí hay intercambio
A[3]>A[4] (16>27) no hay intercambio
A[2]>A[3] (12>16) no hay intercambio
A[1]>A[2] (15>12) sí hay intercambio

Última posición de intercambio de derecha a izquierda : 2

A: 08 12 15 16 27 44 35 67

Segunda etapa (de izquierda a derecha)

A[2]>A[3] (15>16) no hay intercambio
A[3]>A[4] (16>27) no hay intercambio
A[4]>A[5] (27>44) no hay intercambio
A[5]>A[6] (44>35) sí hay intercambio

Última posición de intercambio de izquierda a derecha : 6

A: 08 12 15 16 27 34 44 67

Al realizar la primera etapa de la tercera pasada se observa que no se realizaron intercambios, por lo tanto la ejecución del algoritmo se termina.

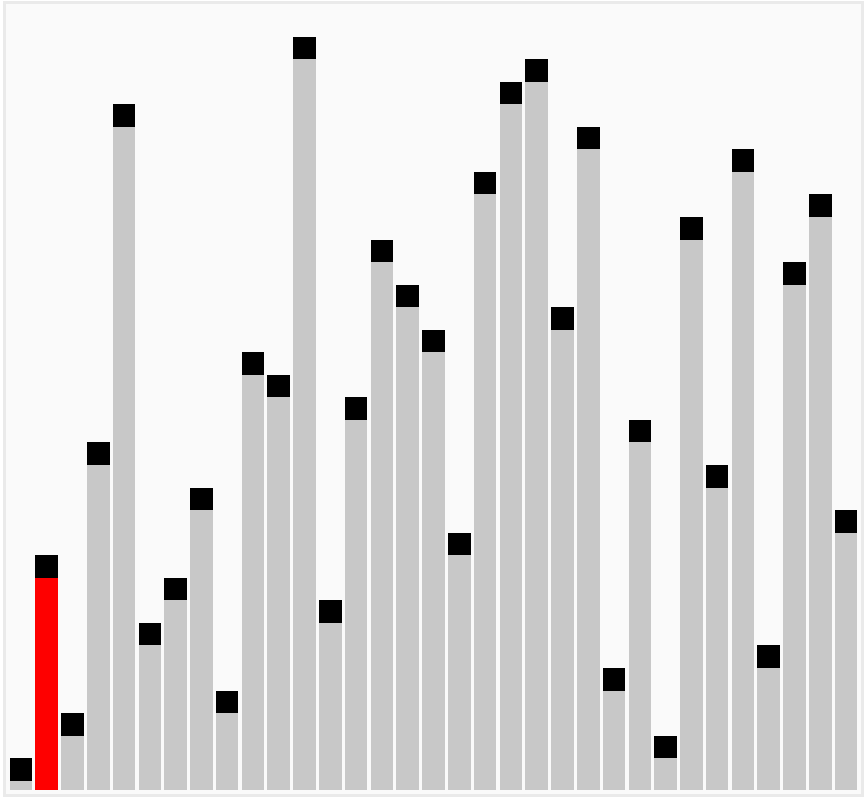
Análisis de eficiencia del método Shaker Sort

- El análisis del método de la sacudida y en general el de los métodos mejorados son más complejos.

- Para el análisis de este método es necesario tener en cuenta tres factores que afectan directamente al tiempo de ejecución del algoritmo: las comparaciones entre las claves, los intercambios entre las mismas y las pasadas que se realizan.

- Los estudios demuestran que sólo pueden reducirse las dobles comparaciones entre claves; pero debe recordarse que la operación de intercambio es una tarea más complicada y costosa que la de comparación. Por lo tanto, es posible afirmar que las hábiles mejoras realizadas sobre el método de intercambio directo sólo producen resultados apreciables si el arreglo está parcialmente ordenado $O(n)$, en el caso peor y promedio será como se esperaría $O(n^2)$

Animación



Método de selección directa

- La idea básica de este algoritmo consiste en buscar el menor elemento en el arreglo y colocarlo en primera posición. Luego se busca el segundo elemento más pequeño del arreglo y se lo coloca en segunda posición. El proceso continua hasta que todos los elementos delarreglo hayan sido ordenados.
  - El método se basa en los siguientes principios:
    1. Seleccionar el menor elemento del arreglo.
    2. Intercambiar dicho elemento con el primero.
    3. Repetir los pasos anteriores con los (n-1), (n-2) elementos y así sucesivamente hasta que sólo quede el elemento mayor.

Ejemplo Selección directa

Arreglo: 15 67 08 16 44 27 12 35

PRIMERA PASADA

menor=A[0] (15)

(A[1]<menor) (67<15) no se cumple

(A[2]<menor) (08<15) Si menor=A[2] (08)

(A[3]<menor) (16<08) no se cumple

(A[4]<menor) (44<08) no se cumple

(A[5]<menor) (27<08) no se cumple

(A[6]<menor) (12<08) no se cumple

(A[7]<menor) (35<08) no se cumple

Se intercambia A[2] (08) con A[0] (15)

Arreglo: 08 67 15 16 44 27 12 35

2da: 08 12 15 16 44 27 67 35

3ra: 08 12 15 16 44 27 67 35

4ta: 08 12 15 16 44 27 67 35

5ta: 08 12 15 16 27 44 67 35

6ta: 08 12 15 16 27 35 67 44

7ma: 08 12 15 16 27 35 44 67

Análisis de eficiencia de Selección directa

- Comparaciones:

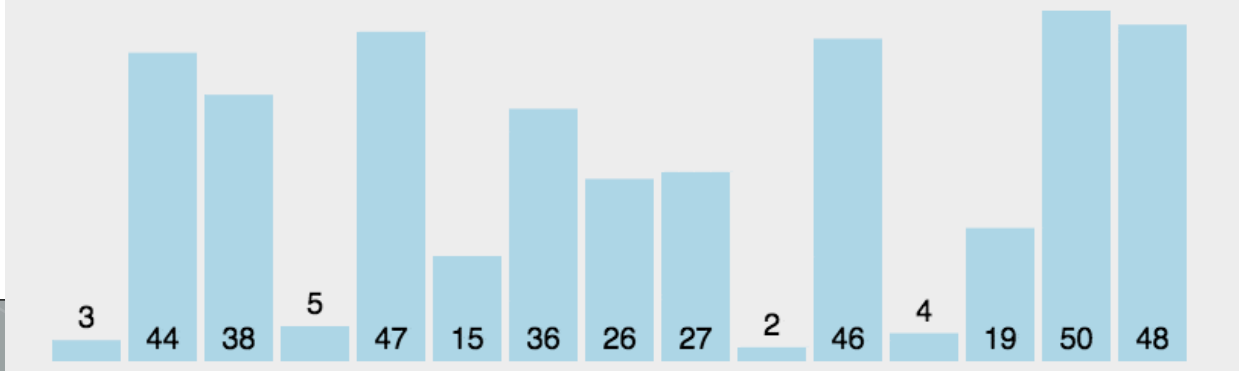
$$C=(n-1)+(n-2)+\ldots+2+1=(n*(n-1))/2= (n^2-n)/2$$

- Intercambios o Movimientos:

$$M= n-1$$

Si se quisiera ordenar un arreglo de 500 elementos:

Se efectuarán 124,750 comparaciones. Se efectuarán 499 movimientos.



Inserción Directa

- El método de ordenación por inserción directa es el que generalmente utilizan los jugadores de cartas cuando ordenan éstas, de ahí que también se conozca con el nombre de método de la baraja.

- La idea central de este algoritmo consiste en insertar un elemento del arreglo en la parte izquierda del mismo, que ya se encuentra ordenada. Este procesos se repite desde el segundo hasta el n-ésimo elemento.

Ejemplo Inserción directa

Arreglo: 15 67 08 16 44 27 12 35

PRIMERA PASADA

A[1]<A[0] (67<15) no hay intercambio

A: 15 67 08 16 44 27 12 35

SEGUNDA PASADA

A[2]<A[1] (08<67) sí hay intercambio

A[1]<A[0] (08<15) sí hay intercambio

A: 08 15 67 16 44 27 12 35

TERCERA PASADA

A[3]<A[2] (16<67) sí hay intercambio

A[2]<A[1] (16<15) no hay intercambio

A: 08 15 16 67 44 27 12 35

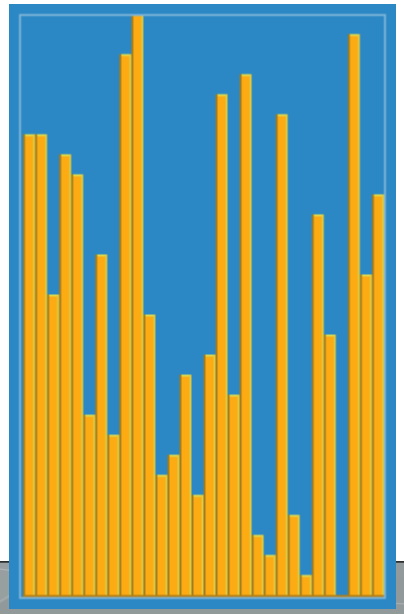
Una vez que se determina la posición correcta del elemento se interrumpen las comparaciones.

4ta: 08 15 16 44 67 27 12 35

5ta: 08 15 16 27 44 67 12 35

6ta: 08 12 15 16 27 44 67 35

7ma: 08 12 15 16 27 35 44 67



Análisis de eficiencia de Inserción directa

- Comparaciones:

$C_{min} = n-1$

$C_{máx} = (n^2-n)/2$

$C_{med} = (n^2+n-2)/4$

- Intercambios o Movimientos:

$M_{min} = 0$

$M_{máx} = (n^2-n)/2$

$M_{med} = (n^2-n)/4$

Inserción Binaria

- Es una mejora del método de inserción directa.
- Consiste en utilizar 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 en la inserción directa, se repite desde el segundo hasta el n-ésimo elemento.

Ejemplo Inserción binaria

Arreglo: 15 67 08 16 44 27 12 35

PRIMERA PASADA

A[1]<A[0] (67<15) no hay intercambio

A: 15 67 08 16 44 27 12 35

SEGUNDA PASADA

A[2]<A[1] (08<15) sí hay intercambio A:

08 15 67 16 44 27 12 35

TERCERA PASADA

A[3]<A[1] (16<15) no hay intercambio

A[3]<A[2] (16<67) sí hay intercambio A:

08 15 16 67 44 27 12 35

CUARTA PASADA

A[4]<A[1] (44<15) no hay intercambio

A[4]<A[2] (44<16) no hay intercambio

A[4]<A[3] (44<67) sí hay intercambio

A: 08 15 16 44 67 27 12 35

QUINTA PASADA

A[5]<A[2] (27<16) no hay intercambio

A[5]<A[3] (27<44) sí hay intercambio

A: 08 15 16 27 44 67 12 35

SEXTA PASADA

A[6]<A[2] (12<16) sí hay intercambio

A[6]<A[0] (12<08) no hay intercambio

A[6]<A[1] (12<15) sí hay intercambio

A: 08 12 15 16 27 44 67 35

SÉPTIMA PASADA

A[7]<A[3] (35<16) no hay intercambio

A[7]<A[5] (35<44) sí hay intercambio

A[7]<A[4] (35<27) no hay intercambio

A: 08 12 15 16 27 35 44 67

Análisis de eficiencia de Inserción binaria

- Caso antinatural: el método efectúa el menor número de comparaciones cuando el arreglo está totalmente desordenado y el máximo cuando está ordenado.
- En una búsqueda secuencial se necesitan k comparaciones para insertar un elemento, en una búsqueda binaria se necesitará la mitad de las k comparaciones, por lo tanto:

$$C_{max}=(n^2-n)/4$$

- Intercambios o Movimientos: igual a la inserción directa.

Comparación entre métodos cuadráticos

Método		Ordenada	Desordenada	Orden Inverso
Intercambio directo	C	$(n^2-n)/2$	$(n^2-n)/2$	$(n^2-n)/2$
	M	0	$0.75^* (n^2-n)$	$1.5^* (n^2-n)$
Inserción directa	C	$(n-1)$	$(n^2+n-2)/4$	$(n^2-n)/2$
	M	0	$(n^2-n)/4$	$(n^2-n)/2$
Selección directa	C	$(n^2-n)/2$	$(n^2-n)/2$	$(n^2-n)/2$
	M	$(n-1)$	$(n-1)$	$(n-1)$

¿Cuándo un algoritmo es O(log n)?

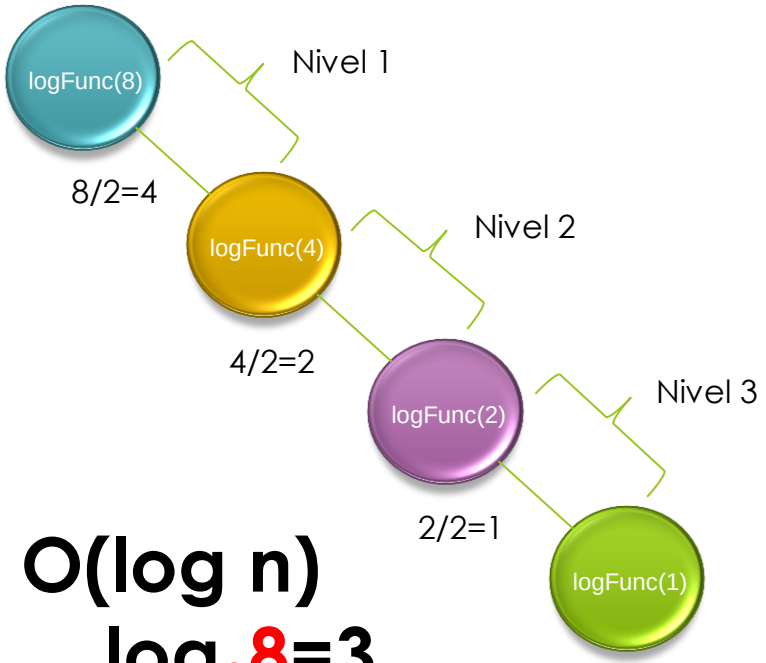
Logaritmo $num^{num} = algun\_num?$

Log base 2 de 8 es

$2^{?} = 8 \rightarrow \log_2 8 = 3$

Ejemplo: n = 8

```
int logFunc(n){
    [          ]
    n = n/2;
    return logFunc(n);
}
```



O(log n)

$\log_2 8 = 3$

Métodos logarítmicos

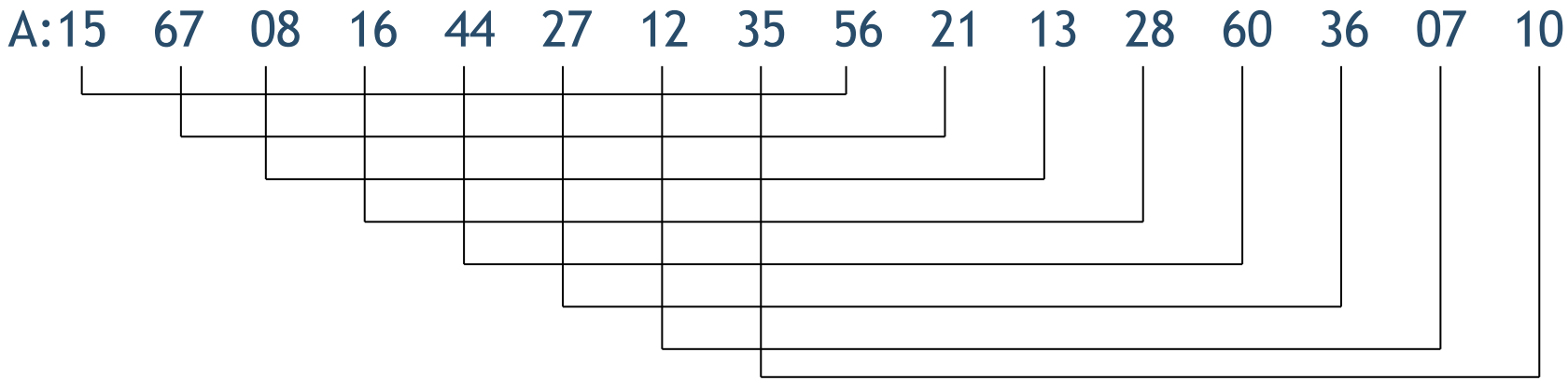
Inserción por Incremento Decreciente (Shell)

- Este método consiste en subdividir el arreglo original en grupos de elementos más pequeños.
- Dichos grupos están compuestos por los elementos separados por k posiciones.
- En seguida se aplica el método de ordenación por inserción directa en cada subgrupo.
- El proceso se repite empezando con una k grande y se decrementa en cada iteración hasta llegar a k = 1.

Ejemplo Shell

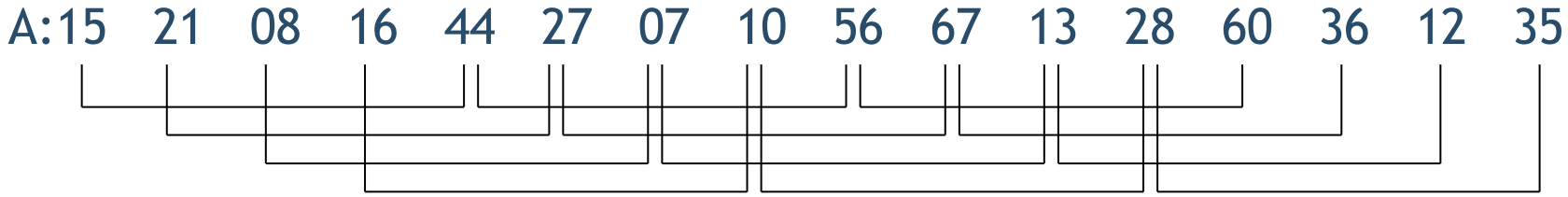
PRIMERA PASADA:

Se dividen los elementos en 8 grupos de 2 elementos cada uno.



SEGUNDA PASADA:

Se dividen los elementos en 4 grupos de 4 elementos cada uno.

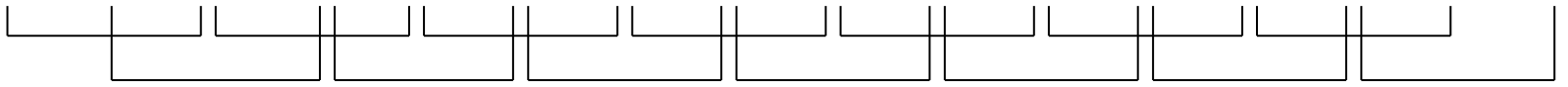


Ejemplo Shell

TERCERA PASADA:

Se dividen los elementos en 2 grupos de 8 elementos cada uno.

A: 15 21 07 10 44 27 08 16 56 36 12 28 60 67 13 35



CUARTA PASADA:

Se dividen los elementos en 1 grupo de todos los elementos.

A: 07 10 08 16 12 21 13 27 15 28 44 35 56 36 60 67

La ordenación produce:

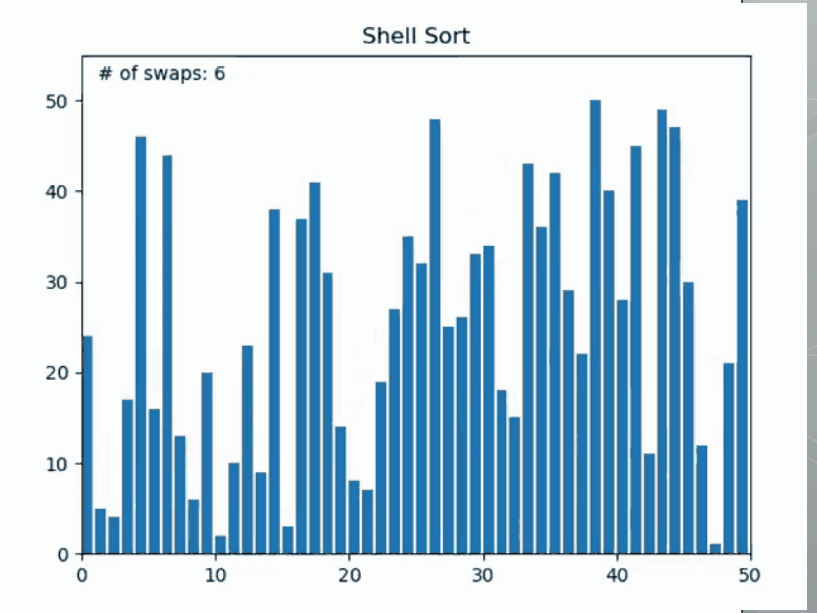
A: 07 08 10 12 13 15 16 21 27 28 35 36 44 56 60 67

Análisis de eficiencia del método Shell

- El tiempo de ejecución del algoritmo es del orden de $n*(\log n)^2$.
- El número de comparaciones y de movimientos depende de la secuencia de intervalos que se escoja.
- https://www.cs.princeton.edu/~rs/shell/paperF.pdf

- Algunos estudios demuestran que las mejores secuencias para valores de N comprendidos entre 100 y 60,000 son las siguientes:
  - $1,3,5,9,\ldots,2^k+1$
  - $1,3,7,15,\ldots,2^k-1$
  - $1,3,5,11,\ldots,(2^k\pm1)/3$
  - $1,4,13,40,\ldots,(3^k-1)/2$

Donde k=0,1,2,3,...



https://www.youtube.com/watch?v=CmPA7zE8mx0

Método de ordenación QuickSort

Es una mejora del método de ordenación por intercambio directo y consiste en:

1. Elegir un elemento x al azar.
2. Buscar por la izquierda del arreglo un elemento mayor que x.
3. Buscar por la derecha un elemento menor o igual que x.
4. Si no se han cruzado los índices izquierda y derecha intercambiar el elemento mayor.
5. Repetir desde el paso 2 mientras no se crucen los índices izquierdo y derecho.
6. Intercambiar x con el elemento de la derecha.
7. En este momento el elemento x divide el arreglo en dos subarreglos tales que el subarreglo izquierdo contiene los elementos menores o iguales que x y el subarreglo derecho mayores que x, por lo tanto x esta ordenado.
8. Hacer quicksort con el arreglo izquierdo.
9. Hacer quicksort con el arreglo derecho.

Ejemplo QuickSort

Arreglo: 15 67 08 16 44 27 12 35

Se selecciona A[0], por lo tanto X=15

PRIMERA PASADA

Recorrido de derecha a izquierda

A[7]>=X (35>=15) no hay intercambio

A[6]>=X (12>=15) sí hay intercambio

A: 12 67 08 16 44 27 15 35

Recorrido de izquierda a derecha

A[2]<=X (67<=15) sí hay intercambio

A: 12 15 08 16 44 27 67 35

SEGUNDA PASADA

Recorrido de derecha a izquierda

A[5]>=X (27>=15) no hay intercambio

A[4]>=X (44>=15) no hay intercambio

A[3]>=X (16>=15) no hay intercambio

A[2]>=X (08>=15) sí hay intercambio

A: 12 08 15 16 44 27 67 35

Como el recorrido de izquierda y derecha debería iniciarse en la misma posición donde se encuentra el elemento X, el proceso termina, ya que el elemento X se encuentra en la posición correcta.

A: 12 08 15 16 44 27 67 35

1er. Conjunto 2do. Conjunto

A: 12 08 15 16 44 27 67 35

A: 12 08 15 16 35 27 44 67

A: 12 08 15 16 27 35 44 67

A: 08 12 15 16 27 35 44 67

Análisis de eficiencia del QuickSort

- Es el método más rápido de ordenación interna.
- Si se escoge en cada pasada el elemento central del conjunto de datos a analizar, el número de pasadas necesarias para ordenarlo es del orden de log n.
- El número de comparaciones promedio se calcula como:

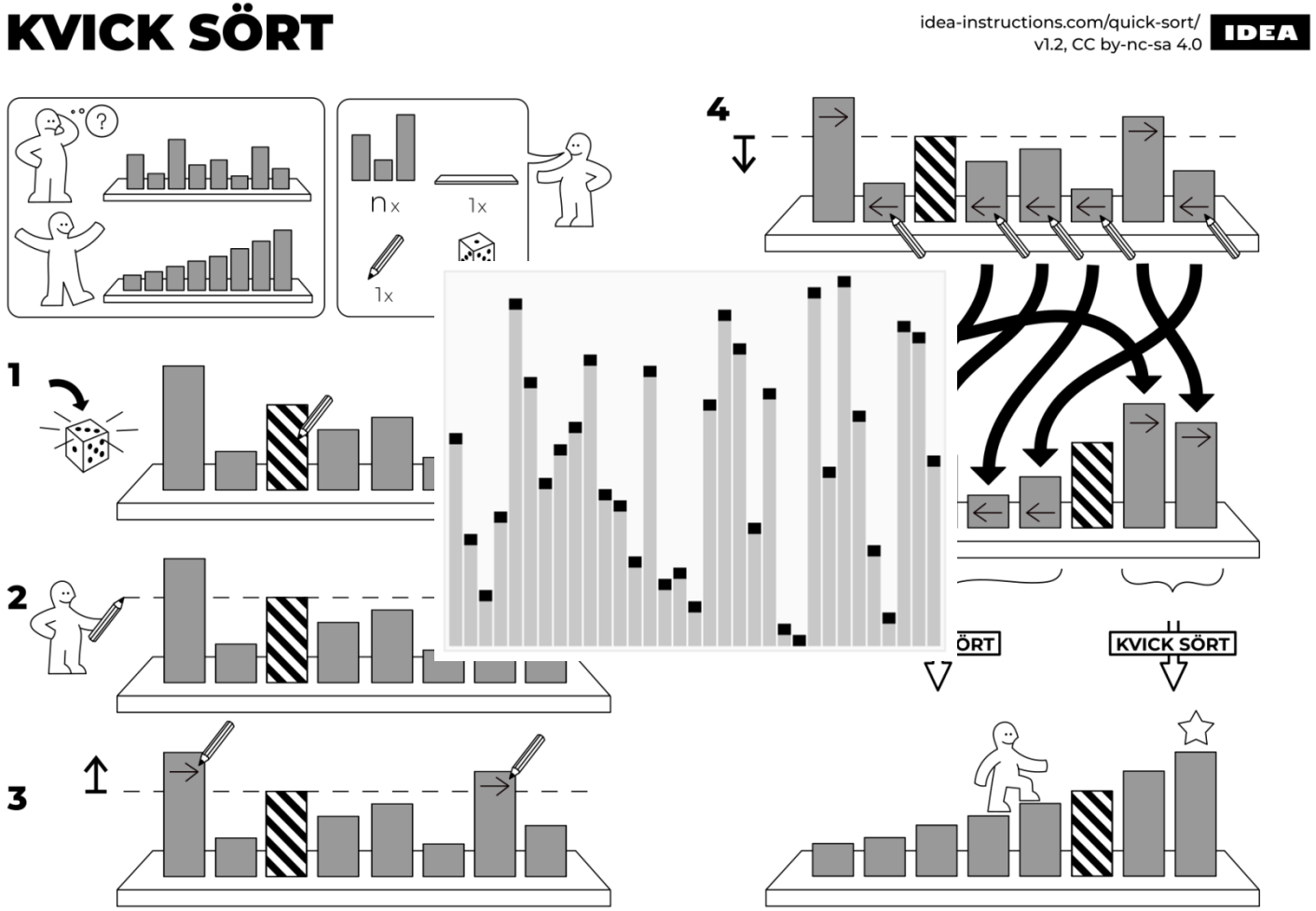
$$C=(n-1)*\log n$$

- Puesto que encontrar el elemento central del conjunto de datos es una tarea difícil, se recomienda que el elemento se escoja arbitrariamente.
- El peor caso sucede cuando el arreglo ya esta ordenado o cuando esta en orden inverso y número de comparaciones máximo sera:

$$C_{máx}=(n^2+n)/2-1$$

- El tiempo promedio de ejecución es $O(n*\log n)$ y para el peor caso es $O(n^2)$.

QuickSort



https://www.youtube.com/watch?v=ywWBy6J5gz8

Método de ordenación Merge Sort

Es un ejemplo del principio "divide y vencerás" y consiste en:

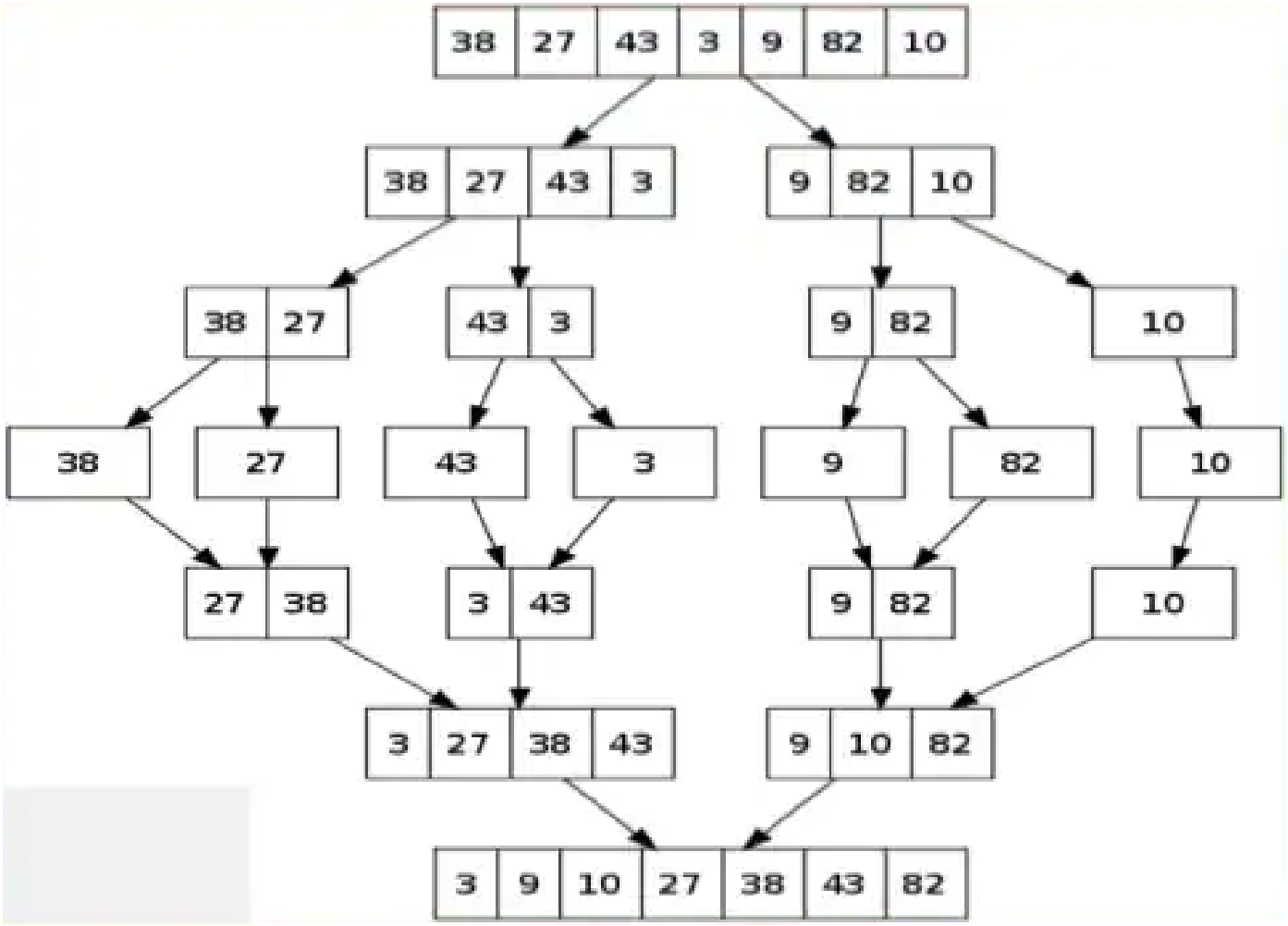
1. Dividir el arreglo de entrada en dos partes de tamaño similar
2. Llamar recursivamente al algoritmo con cada una de las 2 mitades y después mezclar las 2 mitades ordenadas
3. Se usa la función mezcla() para mezclar las 2 mitades.

Dividir (divide) el problema en sub-problemas que son similares al original pero menores en tamaño.

Resolver (conquer) los sub-problemas mediante llamadas recursivas. Si son suficientemente pequeños resolver de forma directa.

Combinar las soluciones para crear una solución al problema original

Ejemplo MergeSort

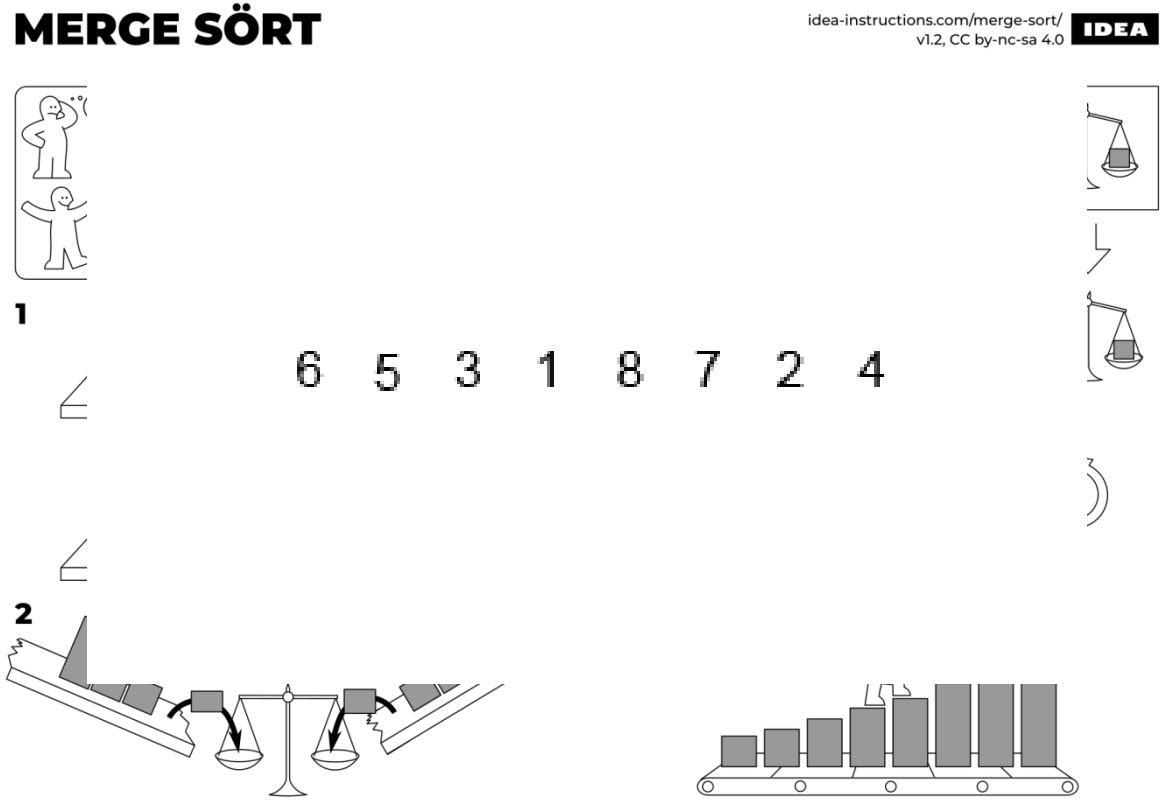


Análisis de eficiencia del MergeSort

- El número de comparaciones se calcula como:

$$C = n * \log n$$

- El tiempo promedio de ejecución y el peor caso es $O(n*\log n)$.



https://www.youtube.com/watch?v=XaqR3G_NVoo

Método de ordenación HeapSort

- El ordenamiento por montículos (Heap sort) es un algoritmo de ordenación con complejidad computacional O(n log n).
- Este algoritmo 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.
- Basa su funcionamiento en una propiedad de los montículos, por la cual, la cima contiene siempre el menor elemento (o el mayor, según se haya definido el montículo) de todos los almacenados en él.
- El proceso de ordenación consta de 2 partes:
  1. Construir el montículo.
  2. Eliminar repetidamente la raíz del montículo.

Concepto de montículo

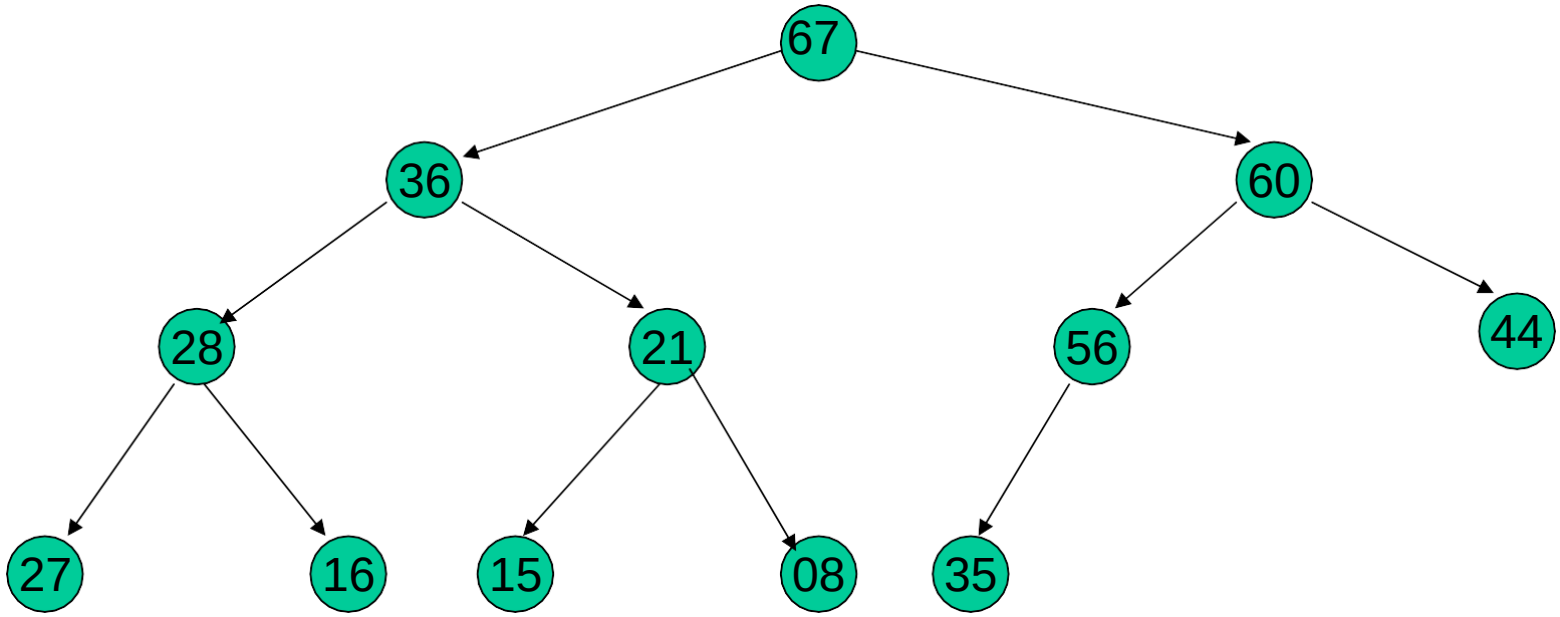
- En computación, un montículo (heap) es una estructura de Árbol con información perteneciente a un conjunto ordenado. Los montículos tienen la característica de que cada nodo tiene un valor mayor o igual que el de todos sus nodos hijos.
- Sean A y B dos nodos de un montículo tal que B es un hijo de A. El montículo debe entonces satisfacer la siguiente condición (Propiedad de montículo):

$$clave(A) \geq clave(B)$$

- Ésta es la única restricción en los montículos. Ella implica que el mayor elemento (o el menor, dependiendo de la relación de orden escogida) está siempre en el nodo raíz.
- Debido a esto, los montículos se utilizan para implementar colas de prioridad.
- La eficiencia de las operaciones en los montículos es crucial en diversos algoritmos de recorrido de grafos y de ordenamiento (Heapsort).
- Las operaciones normalmente utilizadas en un montículo son la inserción de un elemento cualquiera y la eliminación del máximo (el elemento de la raíz).

Representación de un montículo en un arreglo

1. El nodo K se almacena en la posición k correspondiente del arreglo.
2. El hijo izquierdo del nodo k se almacena en la posición (2*(K+1))-1.
3. El hijo derecho del nodo k se almacena en la posición 2*(K+1).



67	36	60	28	21	56	44	27	16	15	08	35

Inserción de un elemento en un montículo

1. Se inserta el elemento en la primera posición disponible.
2. Se verifica si su valor es mayor que el de su padre. Si se cumple esta condición, entonces se efectúa el intercambio. Si no se cumple esta condición, entonces el algoritmo se detiene y el elemento queda ubicado en su posición correcta en el montículo.

Ejemplo:

Si se insertará 63 en el montículo:

67 36 60 28 21 **56** 44 27 16 15 08 35 **63**

63>56 sí hay intercambio:

67 36 **60** 28 21 **63** 44 27 16 15 08 35 56

63>60 sí hay intercambio:

67 36 **63** 28 21 60 44 27 16 15 08 35 56

63>67 no hay intercambio ya queda ubicado el nuevo elemento

Eliminación de un montículo

- El proceso para obtener los elementos ordenados se efectúa eliminando la raíz del montículo en forma repetida. Los pasos para lograr la eliminación de la raíz del montículo son los siguientes:
    1. Se reemplaza la raíz con el elemento que ocupa la última posición del montículo.
    2. Se verifica si el valor de la raíz es menor que el valor más grande de sus hijos. Si se cumple la condición, entonces se efectúa el intercambio. Si no se cumple la condición, entonces el algoritmo se detiene y el elemento queda ubicado en su posición correcta en el montículo.
- Cabe aclarar que el paso 2 se aplica de manera recursiva y desde arriba hacia abajo.
- Al remplazar la raíz por el último elemento del montículo, ésta se coloca en la posición de éste. Es decir, la primera vez la raíz será colocada en posición n, la segunda en (n-1), la tercera en (n-2) y así sucesivamente hasta que quede colocada en las posiciones 2 y luego en la 1.

Ejemplo HeapSort

Montículo: 67 56 60 44 21 28 36 15 35 16 13 08 27 12 07 10

1ra. Eliminación:

Intercambia la raíz 67 con el último elemento 10

A[0]<A[2] (10<60) sí hay intercambio, A[2] es el hijo mayor de A[0]

A[2]<A[6] (10<36) sí hay intercambio, A[6] es el hijo mayor de A[2]

A[6]<A[13] (10<12) sí hay intercambio, A[13] es el hijo mayor de A[6]

60 56 36 44 21 28 12 15 35 16 13 08 27 10 07 67

2da. Eliminación:

Intercambia la raíz 60 con el último elemento 07

A[0]<A[1] (07<56) sí hay intercambio, A[1] es el hijo mayor de A[0]

A[1]<A[3] (07<44) sí hay intercambio, A[3] es el hijo mayor de A[1]

A[3]<A[8] (07<35) sí hay intercambio, A[8] es el hijo mayor de A[4]

56 44 36 35 21 28 12 15 07 16 13 08 27 10 60 67

Ejemplo HeapSort

Eliminación 3: 44 35 36 15 21 28 12 10 07 16 13 08 27 56 60 67

Eliminación 4: 36 35 28 15 21 27 12 10 07 16 13 08 44 56 60 67

Eliminación 5: 35 21 28 15 16 27 12 10 07 08 13 36 44 56 60 67

Eliminación 6: 28 21 27 15 16 13 12 10 07 08 35 36 44 56 60 67

Eliminación 7: 27 21 13 15 16 08 12 10 07 28 35 36 44 56 60 67

Eliminación 8: 21 16 13 15 07 08 12 10 27 28 35 36 44 56 60 67

Eliminación 9: 16 15 13 10 07 08 12 21 27 28 35 36 44 56 60 67

Eliminación 10: 15 12 13 10 07 08 16 21 27 28 35 36 44 56 60 67

Eliminación 11: 13 12 08 10 07 15 16 21 27 28 35 36 44 56 60 67

Eliminación 12: 12 10 08 07 13 15 16 21 27 28 35 36 44 56 60 67

Eliminación 13: 10 07 08 12 13 15 16 21 27 28 35 36 44 56 60 67

Eliminación 14: 08 07 10 12 13 15 16 21 27 28 35 36 44 56 60 67

Eliminación 15: 07 08 10 12 13 15 16 21 27 28 35 36 44 56 60 67

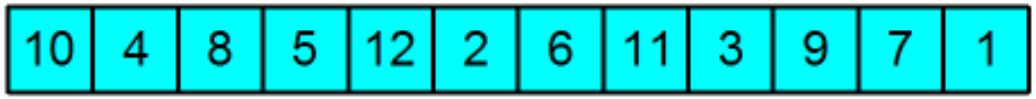
Análisis de eficiencia del HeapSort

- Hay que tomar en cuenta tanto la fase de construcción del montículo como la fase donde se elimina repetidamente la raíz del mismo, para finalmente obtener el arreglo ordenado.
- Es un método muy rápido sobre todo para valores grandes de N.
- El tiempo de ejecución del algoritmo en ambas fases es de:

  $O(n * \log n)$

- El método del montículo puede ser más lento que el quicksort (se estima en 70%). Sin embargo garantiza que aun en el peor caso su tiempo de ejecución es proporcional a O(n*log n).

Heap Sort



Todos los algoritmos juntos

- https://www.toptal.com/developers/sorting-algorithms

Donde encontrarlos