UTN-FRBA

GESTION DE DATOS

METODOS DE CLASIFICACION

DIRECTOR CATEDRA: ING. ENRIQUE REINOSA

CLASIFICACION - CONCEPTOS

- Objetivo

El objetivo es dado un conjunto de valores desordenados del tipo {*a1*, *a2*,…, *an*}, devolver un conjunto ordenado de menor a mayor o de mayor a menor

Por ejemplo una secuencia de entrada como {31, 41, 59, 26, 41, 58} retorna la secuencia {26, 31, 41, 41, 58, 59}

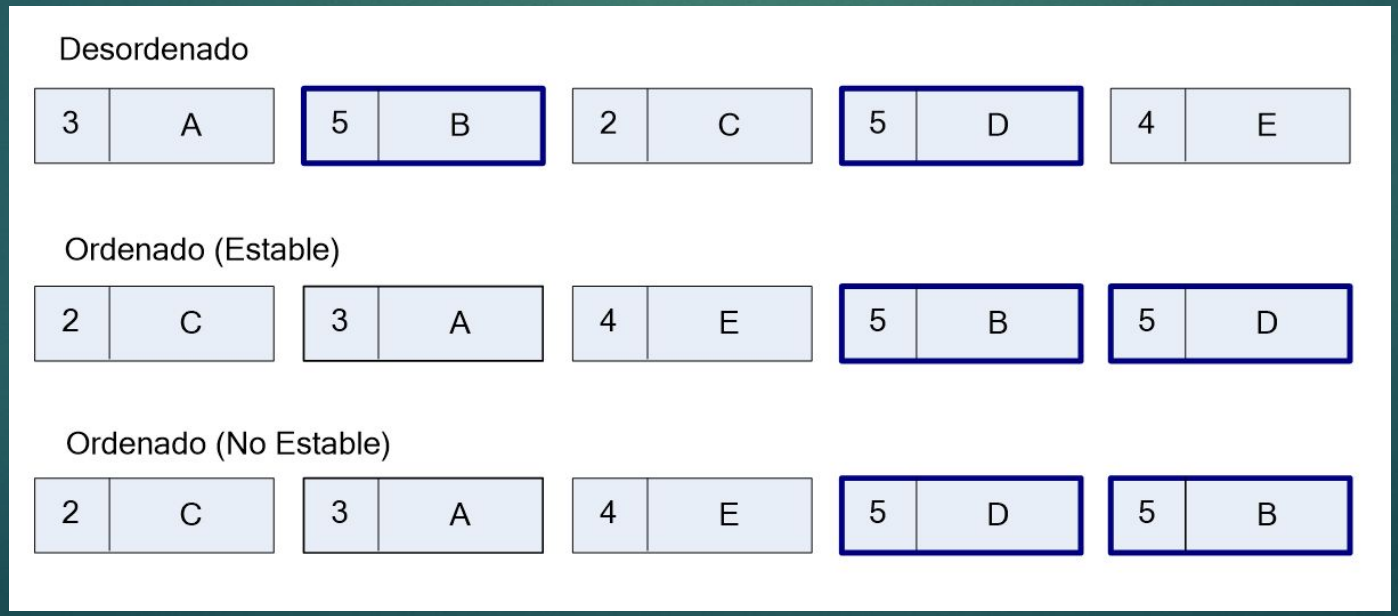
CLASIFICACION - REGISTROS

- En la práctica muy pocas veces los números a ordenar son valores aislados, cada número suele ser parte de una colección de datos llamado registro. Cada registro contiene una clave (key), que es el valor a ser ordenado, y el resto del registro contiene los datos satélites.

- El hecho de ordenar simples números o registros no es relevante para el método de clasificación en sí, por lo que generalmente se asume que la entrada simplemente consiste en números o letras.

CLASIFICACION – ESTABILIDAD

- Un ordenamiento se considera estable si mantiene el orden relativo que tenían originalmente los elementos con claves iguales. Si se tiene dos registros A y B con la misma clave en la cual A aparece primero que B, entonces el método se considera estable cuando A aparece primero que B en el archivo ordenado



CLASIFICACION – IN SITU

- Los métodos in situ son los que transforman una estructura de datos usando una cantidad extra de memoria, siendo ésta pequeña y constante. Generalmente la entrada es sobrescrita por la salida a medida que se ejecuta el algoritmo. Por el contrario los algoritmos que no son in situ requieren gran cantidad de memoria extra para transformar una entrada.

- Esta característica es fundamental en lo que respecta a la optimización de algoritmos, debido a que el hecho de utilizar la misma estructura disminuye los tiempos de ejecución, debido a que no se debe utilizar tiempo en crear nuevas estructuras, ni copiar elementos de un lugar a otro.

CLASIFICACION – IN SITU

```
void invertirArray(int[] a)
  { int[] aux = new int[ a.length ];
    for(int c = 0; c < a.length ; c++)
            { aux[c] = a[a.length - c - 1];}
    a = aux;}

void invertirArrayInSitu(int[] a)
 {  int temp;
    for(int c = 0; c < a.length / 2; c++)
            { temp = a[c];
                    a[c] = a[a.length - c - 1];
                    a[a.length - c - 1] = temp;}}
```

CLASIFICACION – INTERNA Y EXTERNA

- METODO INTERNO: Si el archivo a ordenar cabe en memoria principal, entonces el método de clasificación es llamado método interno.

- METODO EXTERNO: si ordenamos archivos desde un disco u otro dispositivo que no es memoria, se llama método de clasificación externo.

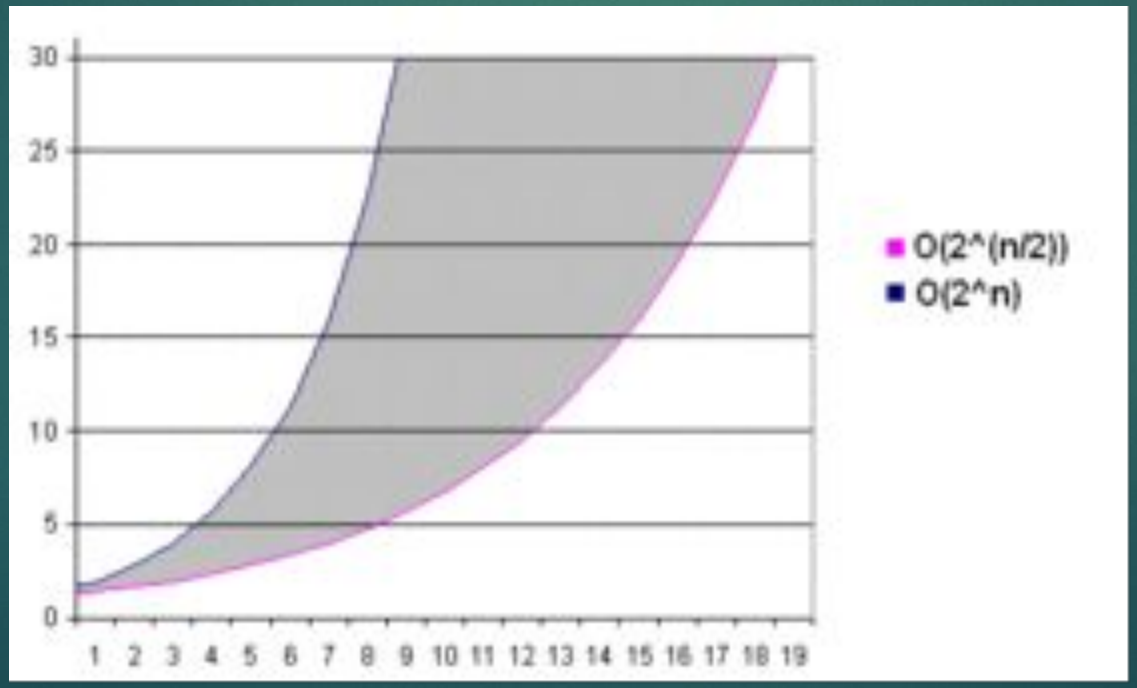
- En función del método aplicado, también será diferente la concepción del algoritmo utilizado, debido que es bastante diferente una clasificación interna a una externa, por las diferencias de tiempo de acceso a los datos y los volúmenes de los mismos.

CLASIFICACION – COMPLEJIDAD

- La teoría de la complejidad computacional es la parte de la teoría de la computación que estudia la complejidad de ejecutar un algorítmo
- La clase de complejidad "P" es el conjunto de los problemas de decisión que pueden ser resueltos en una máquina determinista en tiempo a lo sumo polinómico, lo que corresponde intuitivamente a problemas que pueden ser resueltos aún en el peor de sus casos.
- La clase de complejidad "NP" es el conjunto de los problemas de decisión que pueden ser resueltos por una máquina no determinista en tiempo mayor que polinómico.

CLASIFICACION – COMPLEJIDAD

- El orden de complejidad se describe como ***O(función)*** donde función es la función matemática que acota el comportamiento del algorìtmo en función del tiempo y la cantidad de elementos.



CLASIFICACION – COMPLEJIDAD

Como evaluar la Complejidad de un Algoritmo

- La computadora solo puede realizar operaciones matemáticas y comparaciones por su característica electrónica booleana.
- Para evaluar la complejidad en un algoritmo determinado se evalúan principalmente la cantidad de comparaciones realizadas
- El motivo se basa en que una comparación (if) puede llegar a ser hasta 100 veces más lenta que una operación matemática básica

CLASIFICACION – COMPLEJIDAD

```
void invertirArrayInSitu(int[] a)
 {  int temp;
    for(int c = 0; c < a.length / 2; c++)
            { temp = a[c];
              a[c] = a[a.length - c - 1];
              a[a.length - c - 1] = temp;}}
```

Este algoritmo si bien realizada asignaciones el orden de complejidad se establece a partir de analizar que función matemática acota la cantidad de comparaciones que realizaá en función de los elementos.

De esta forma vemos que este algoritmo realiza n/2 comparaciones, si tomamos como n la cantidad de elementos del array, en función de ello decimos que su orden de complejidad es ***O(n/2)***

CLASIFICACION – METODOS

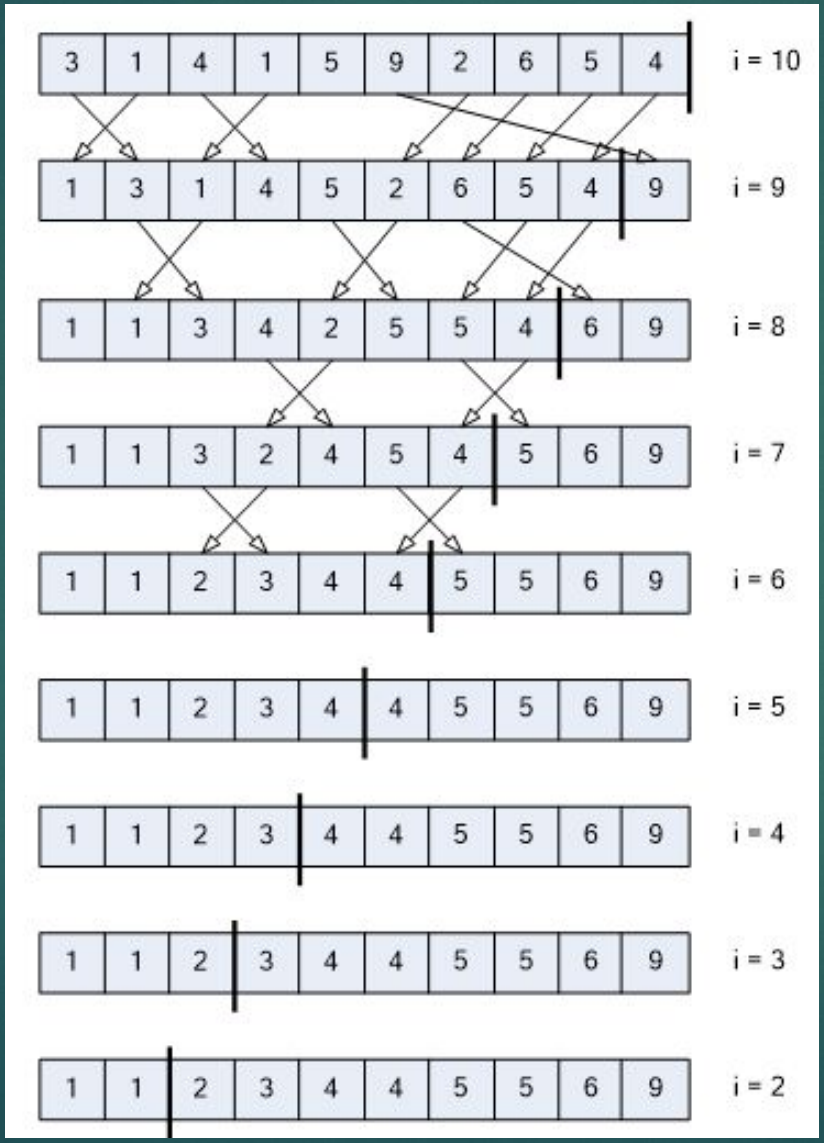
Métodos a analizar

Bubble Sort	*Selection Sort*	*Insertion Sort*
Shell Sort	*Merge Sort*	*Quick Sort*
Bsort	*Meansort*	*Heap Sort*

CLASIFICACION – BUBBLE SORT

- Bubble Sort, o método de la burbuja o de intercambio directo es uno de los métodos más simples y elementales, pero también uno de los más lentos y poco recomendables.
- Consiste en hacer pasadas sobre los datos, donde en cada paso, los elementos adyacentes son comparados e intercambiados si es necesario
- A veces no hace falta hacer pasadas sobre el array para que éste quede ordenado, sino que puede quedar ordenado antes de terminar todas las pasadas.
- El tiempo de ejecución del Bubble Sort en el peor de los casos es de $O(n^2)$, y ocurre cuando el array viene en orden inverso. Sin embargo hay un caso en el que el Bubble Sort puede ordenar en tiempo lineal, y es cuando el array esta previamente ordenado, resultando en un tiempo de ejecución de $O(n)$.

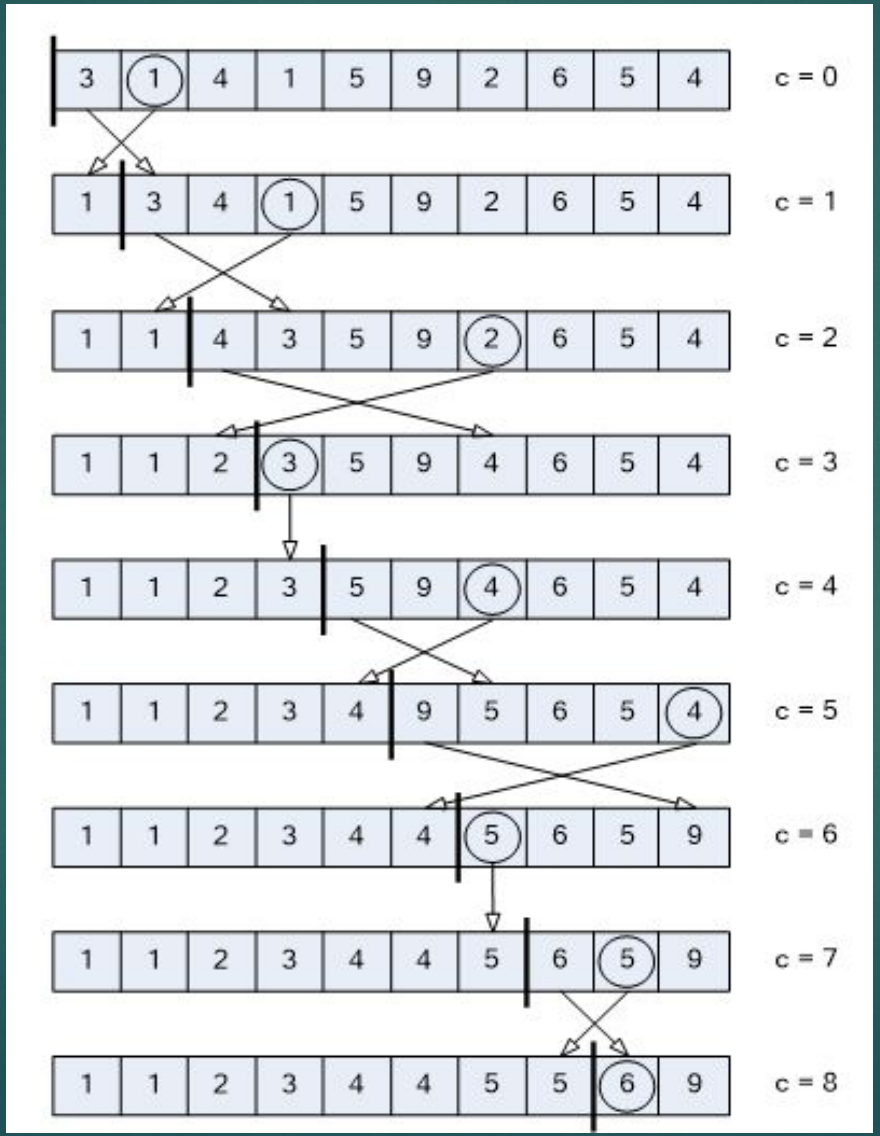
CLASIFICACION – BUBBLE SORT



CLASIFICACION – SELECTION SORT

- Selection Sort u Clasificación por Selección, es otro de los métodos elementales, que es necesario conocer a fin de tratar luego los mas complejos.
- Comienza buscando el elemento más pequeño del array y se lo intercambia con el que esta en la primera posición, luego se busca el segundo elemento más pequeño y se lo coloca en la segunda posición. Se continua con este proceso hasta que todo el array este ordenado.
- Debido a que la mayoría de los elementos se mueven a lo sumo una vez, resulta muy bueno para ordenar archivos que tienen registros muy grandes y claves muy pequeñas.
- A diferencia del Bubble Sort no tiene corte anticipado y su orden de complejidad para el peor caso es $O(n^2)$

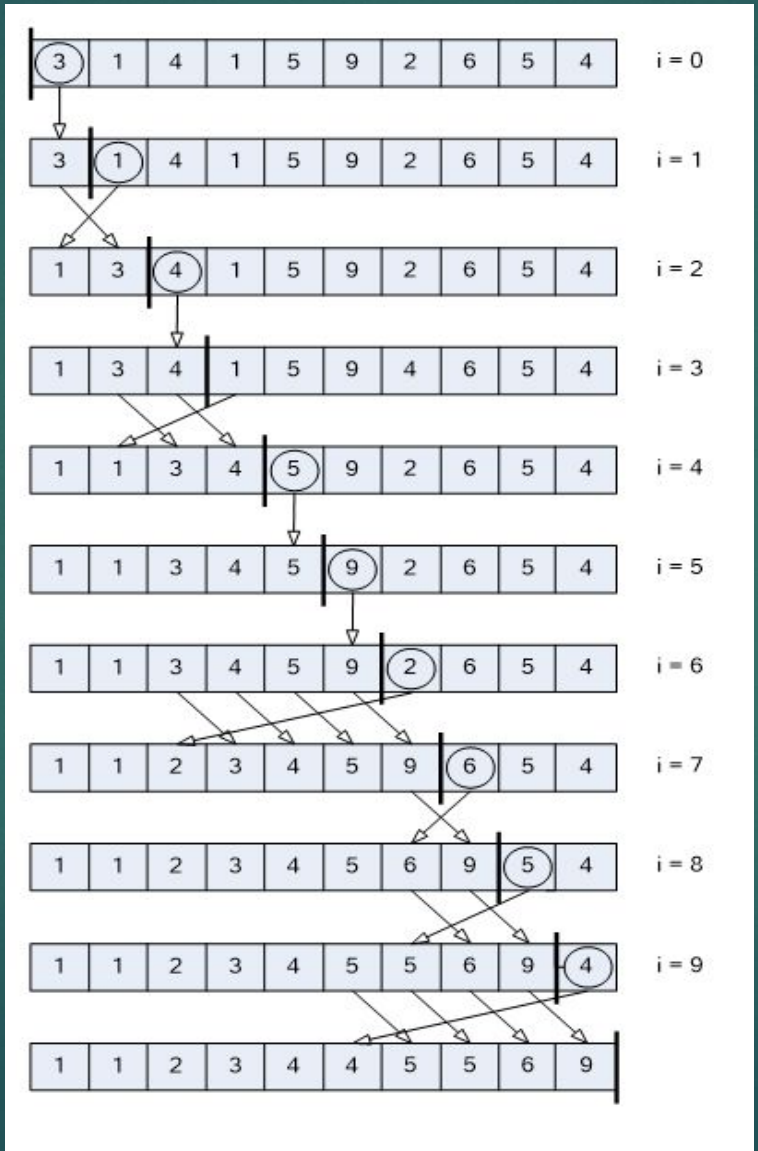
CLASIFICACION – SELECTION SORT



CLASIFICACION – INSERTION SORT

- Este método se basa en la idea del ordenamiento parcial, en el cual hay un marcador que apunta a una posición donde a su izquierda se considera que están los elementos parcialmente ordenados, es decir ordenados entre ellos pero no necesariamente en sus posiciones finales.
- El algoritmo comienza eligiendo el elemento marcado para poder insertarlo en su lugar apropiado en el grupo parcialmente ordenado, para eso sacamos temporalmente al elemento marcado y movemos los restantes elementos hacia la derecha. Nos detenemos cuando el elemento a ser cambiado es más pequeño que el elemento marcado, entonces ahí se intercambian el elemento que esta en esa posición con la del elemento marcado.
- El tiempo de ejecución es $O(n^2)$ y es alcanzable si el array viene ordenado en orden inverso.

CLASIFICACION – INSERTION SORT



CLASIFICACION – SHELL SORT

- Es una modificación del Insertion Sort que disminuye la cantidad de intercambios de elementos. Por ejemplo si hay un elemento muy pequeño muy a la derecha, justo en el lugar donde tendrían que estar los elementos más grandes, para moverlo hacia izquierda se necesitaría hacer cerca de n copias para llegar a la posición indicada.
- No todos los ítems deben ser movidos n espacios, pero en promedio deben moverse n/2 lugares. Por lo tanto lleva n veces n/2 cambios de lugar, resultando en $n^2/2$ copias. Por lo cual el tiempo de ejecución es $\boldsymbol{O(n^2)}$.
- Para evitar la gran cantidad de movimientos, primero compara los elementos más lejanos y luego va comparando elementos mas cercanos para finalmente realizar un Insertion Sort.
- Para lograr esto se utiliza una secuencia $H_1, H_2, \ldots H_n$, denominada secuencia de incrementos. Es importante remarcar que cualquier secuencia es válida siempre que $H_1=1$, es decir que termine realizando un ordenamiento por inserción.
- Si bien no hay un consenso sobre la eficiencia del Shell Sort, se considera que este varia entre $\boldsymbol{O(n^{3/2})}$ y $\boldsymbol{O(n^{7/6})}$

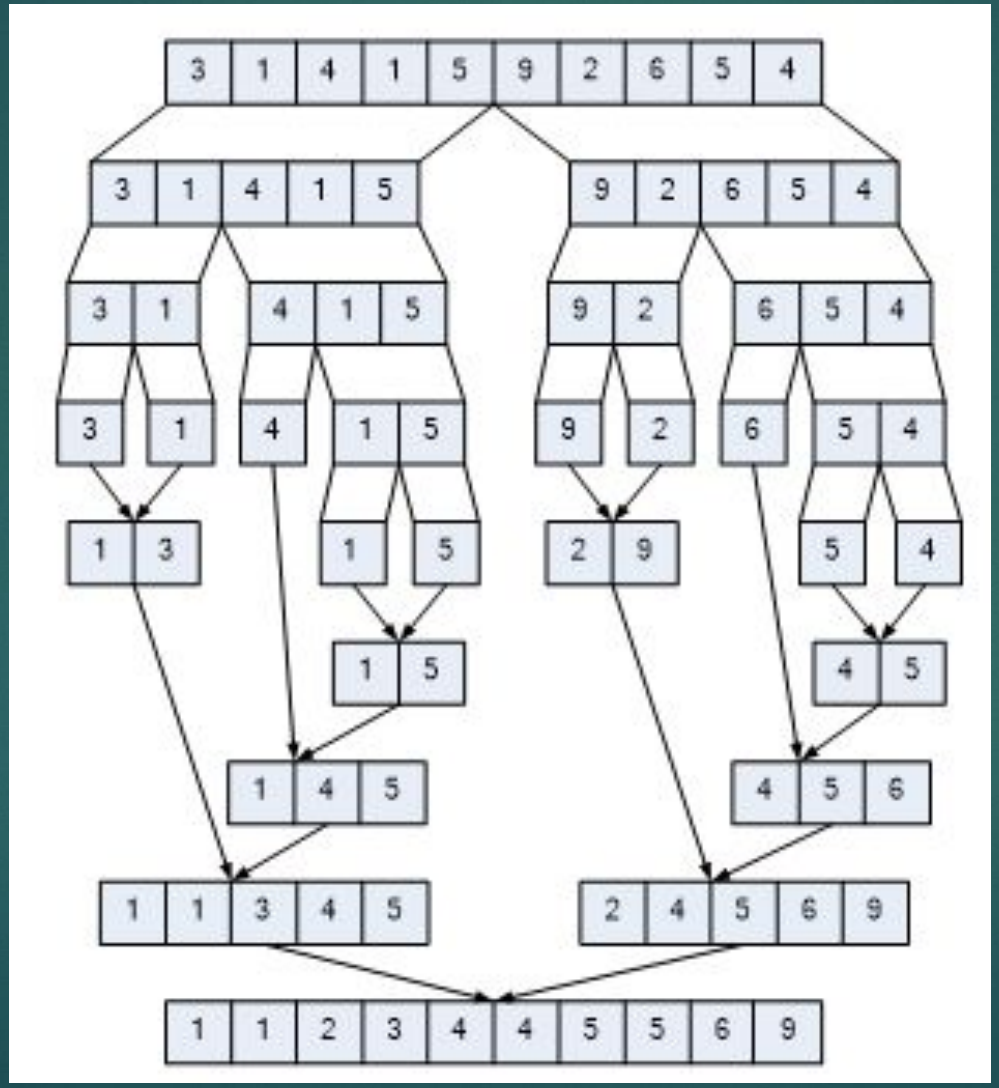
CLASIFICACION – SHELL SORT

Shell Sort después de cada paso luego de la secuencia de incrementos (1,3,5).

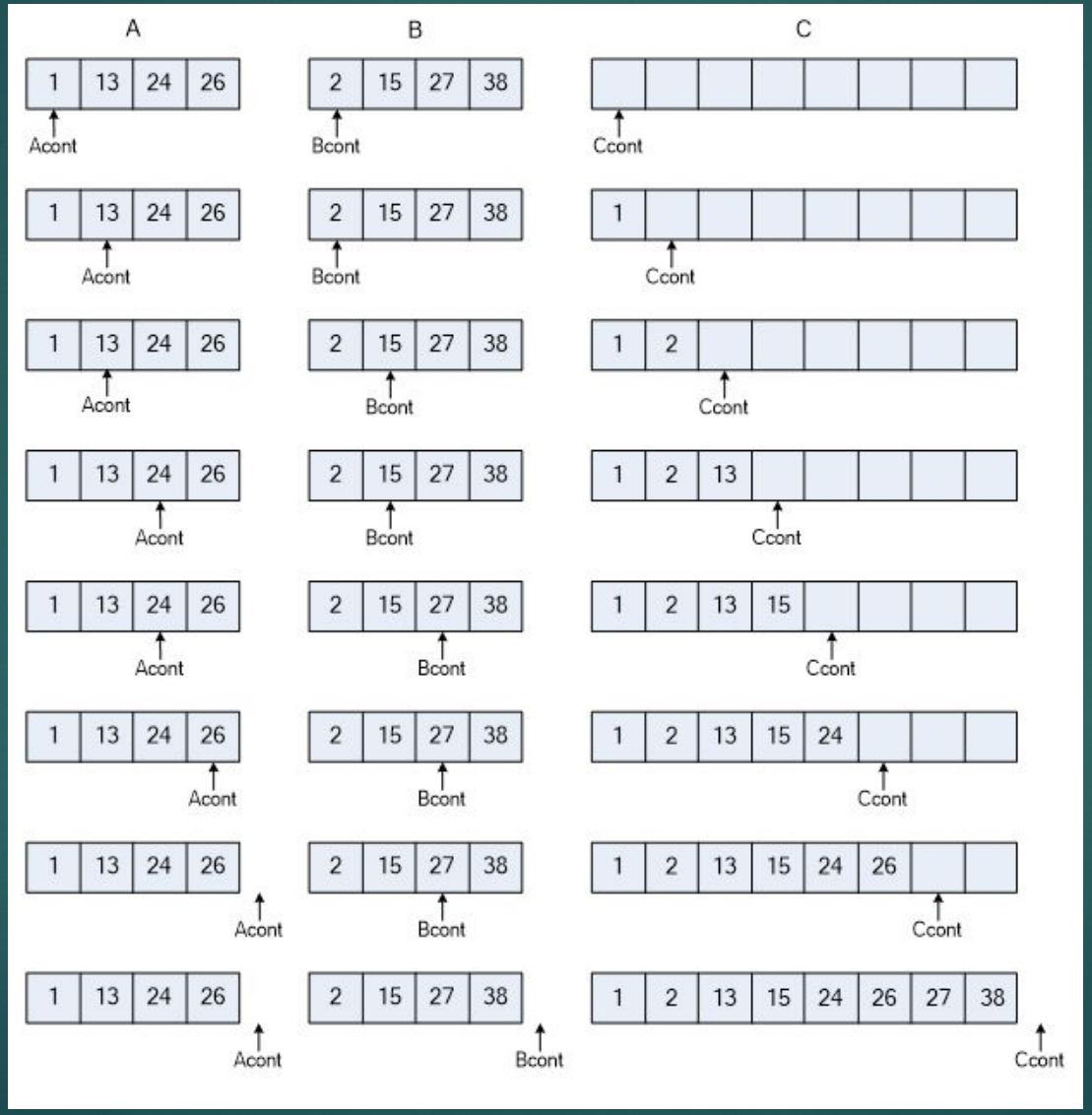
CLASIFICACION – MERGE SORT

- Es un algoritmo recursivo que utiliza la técnica de divide y vencerás para obtener un tiempo de ejecución ***O(nlogn)*** sin importar cual sea la entrada.
- Se basa en la fusión de dos o mas secuencias ordenadas en una única secuencia ordenada. Una de las desventajas de este algoritmo es que requiere de memoria extra proporcional a la cantidad de elementos del array. Es un algoritmo a considerar si estamos buscando velocidad, estabilidad, donde no se tolera un 'peor de los casos' y además disponemos de memoria extra. Algo que hace mas atractivo a Merge Sort es que suele acceder de forma secuencial a los elementos y es de gran utilidad para ordenar en ambientes donde solo se dispone de acceso secuencia a los registros.
- El algoritmo tiene como caso base una secuencia con exactamente un elemento en ella. Y ya que esa secuencia esta ordenada, no hay nada que hacer. Por lo tanto para ordenar una secuencia n > 1 elementos se deben seguir los siguientes pasos:
  - Dividir la secuencia en dos subsecuencias más pequeñas.
  - Ordenar recursivamente las dos subsecuencias
  - Fusionar las subsecuencias ordenadas para obtener el resultado final.

CLASIFICACION – MERGE SORT



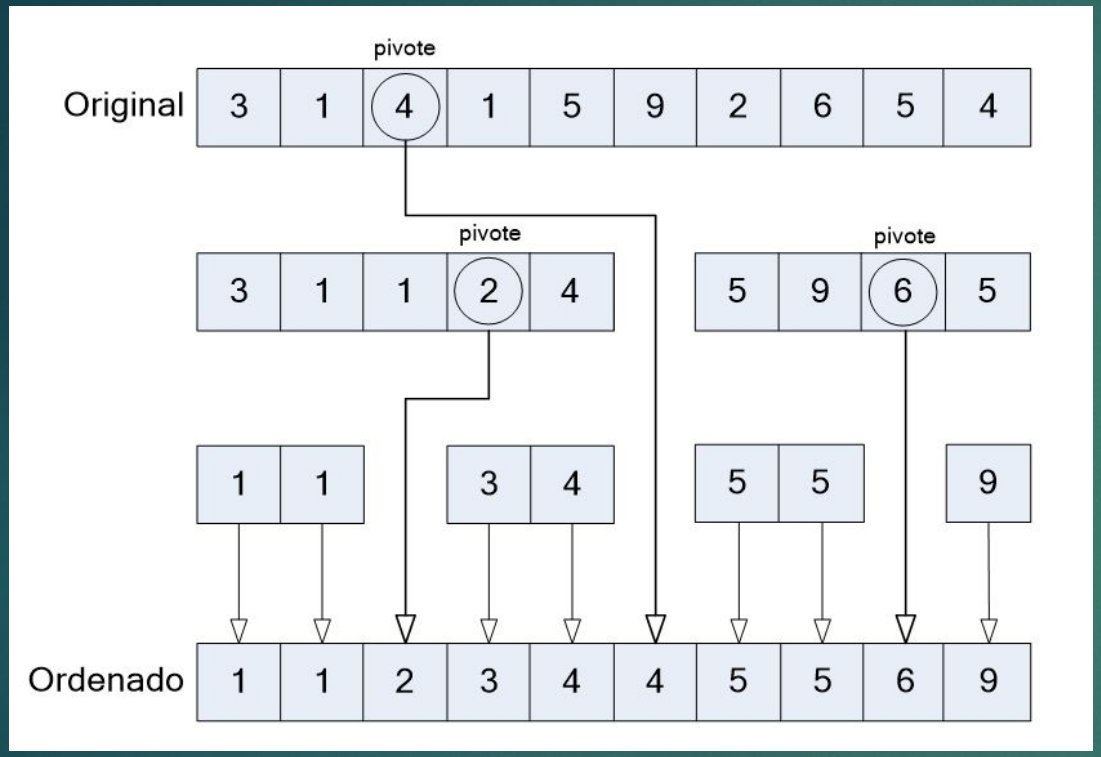
CLASIFICACION – MERGE SORT

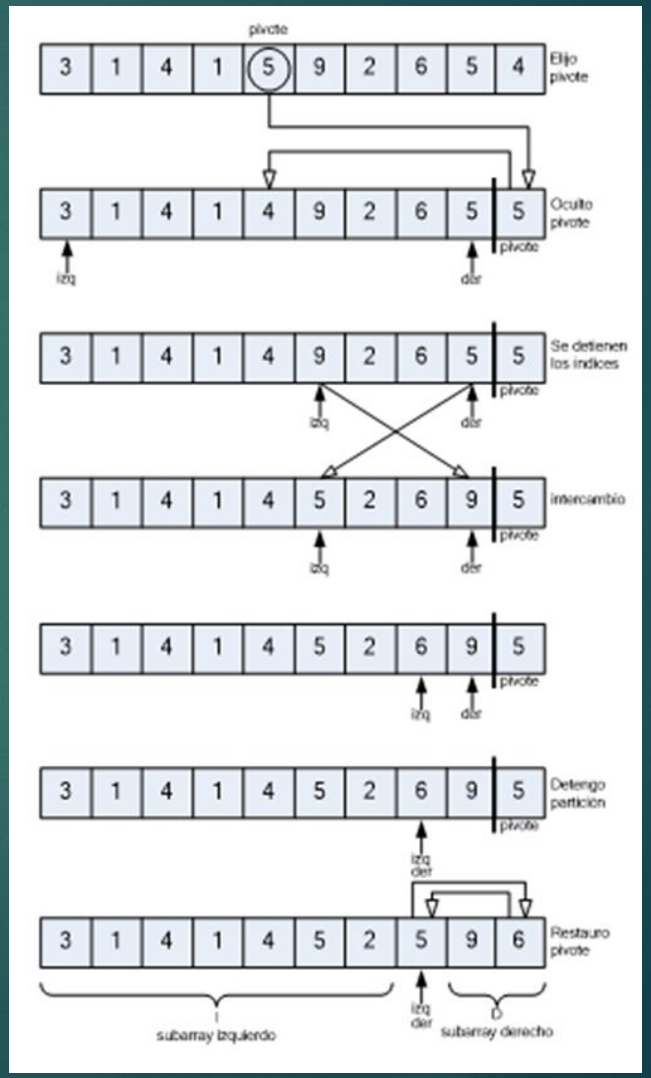


CLASIFICACION – QUICK SORT

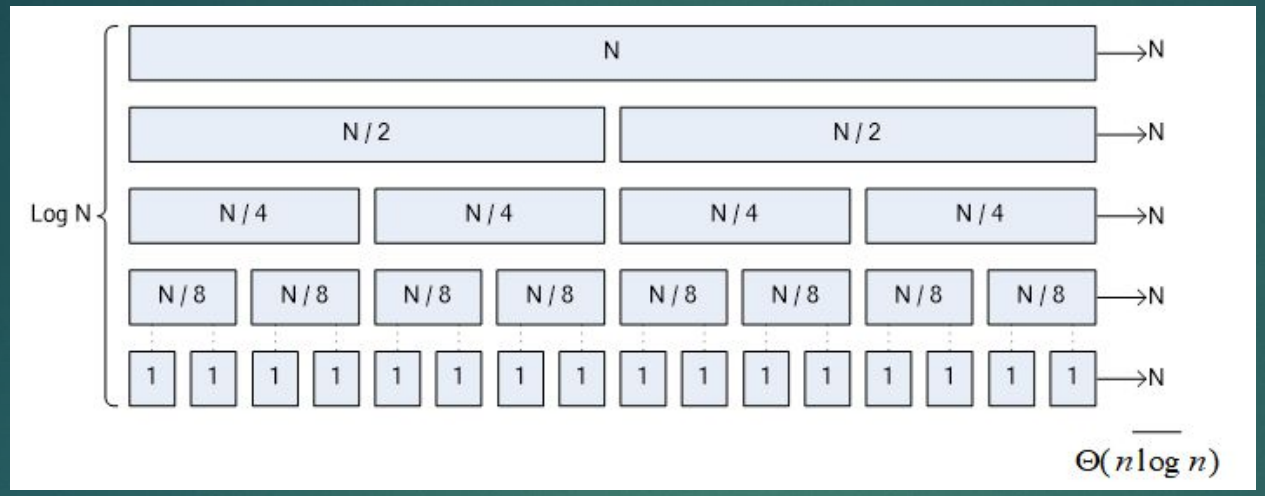
- Es el algoritmo que mejor responde en la mayoría de los casos, con un tiempo promedio de ***O(nlogn)*** y ***O(n^2)*** en el peor de los casos.
- El QuickSort está basado en la idea de divide y vencerás, en el cual un problema se soluciona dividiéndolo en dos o más subproblemas, resolviendo recursivamente cada uno de ellos para luego juntar sus soluciones para obtener la solución del original.
- El algoritmo básico consiste en los siguientes cuatro pasos:
  - Elegir el un elemento como pivote
  - Comparar todos los elementos con el pivote generando dos subconjuntos a izquierda los menores o iguales y a derecha los mayores que el pivote

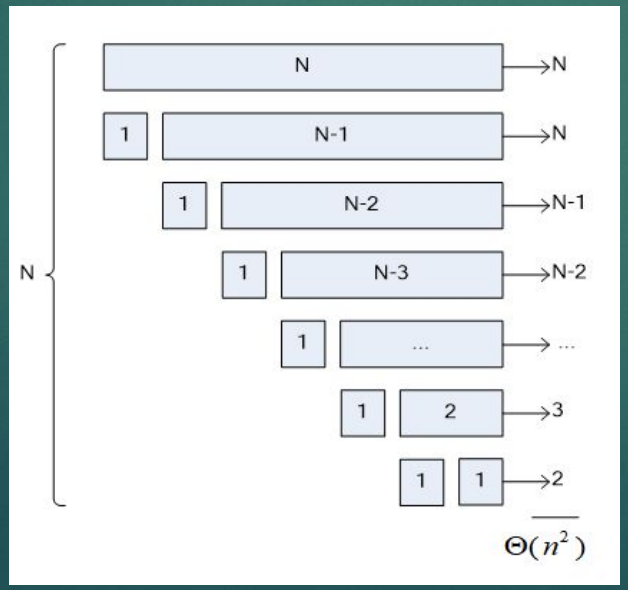
CLASIFICACION – QUICK SORT





CLASIFICACION – QUICK SORT





CLASIFICACION – BSORT - MEANSORT

- BSORT: es una variante del Quicksort donde el funcionamiento del método es igual pero solo cambia la elección del pivote que este caso es el elemento central.

- MEANSORT: es una variante del Quicksort donde el funcionamiento del método es igual pero solo cambia la elección del pivote que este caso es el elemento más próximo a la media.

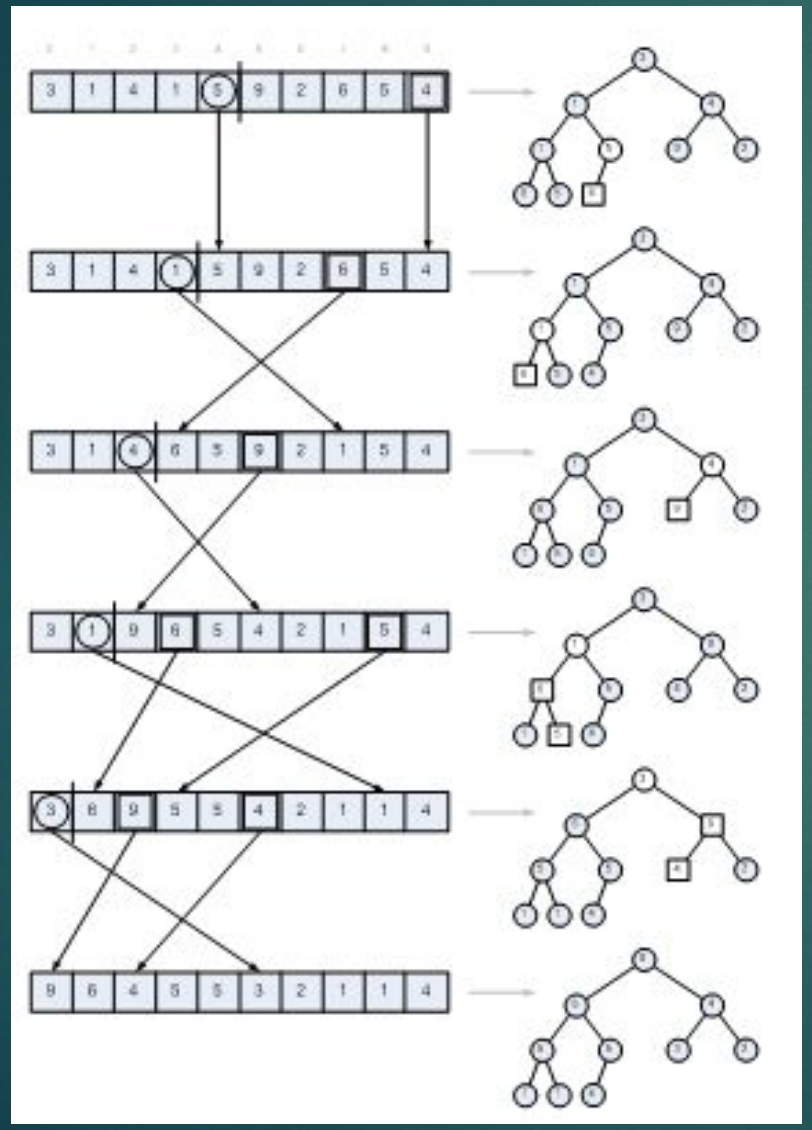
CLASIFICACION – HEAP SORT

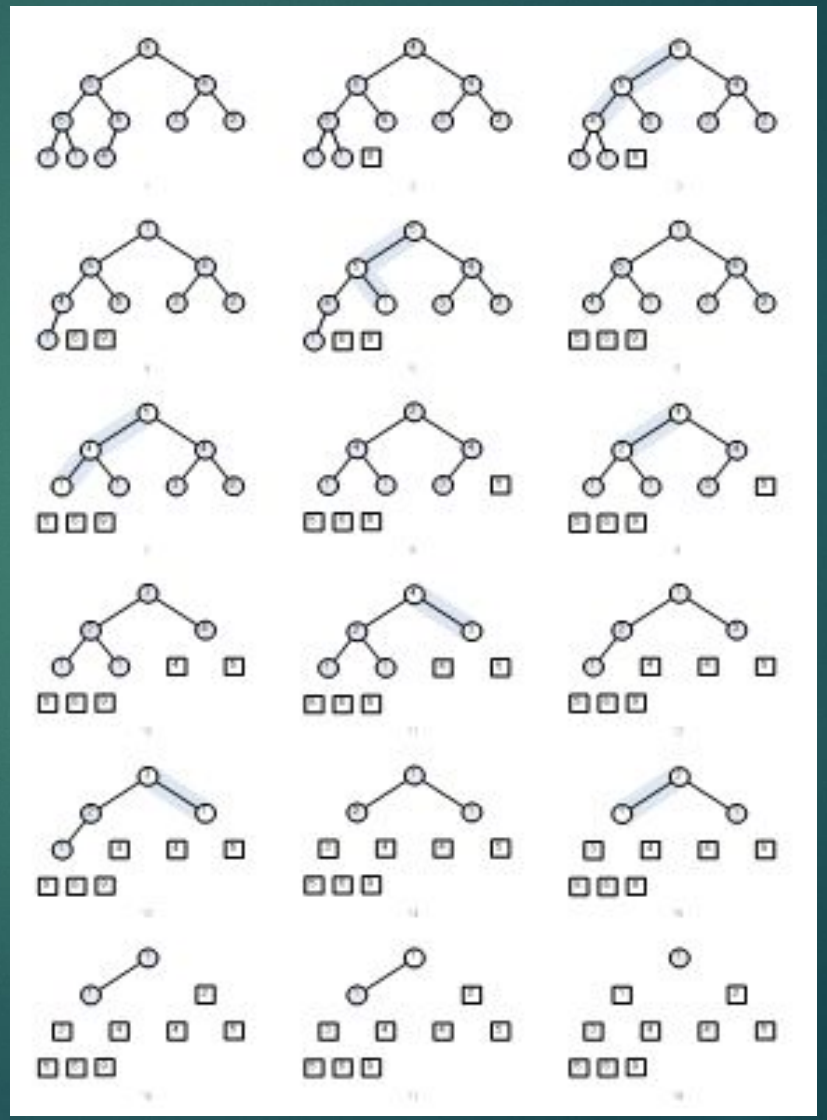
- Heap Sort, o clasificación por montículo (heap), se basa en una estructura de datos llamada montículo binario (heap), que es una de las formas de implementar una cola de prioridad. Un heap es un árbol binario completo en el cual la clave de cada nodo debe ser mayor (o igual) a las claves de sus hijos, si es que tiene. Esto implica que la clave más grande está en la raíz.

- Este algoritmo tiene un orden de complejidad que nunca supera ***O(nlogn)*** y no requiere espacio de memoria adicional (in situ). Es generalmente se usa en sistemas embebidos con restricciones de tiempo real, o en sistemas en donde la seguridad es un factor importante.

- El algoritmo de Heap Sort consiste de dos fases.
  - En la primera fase, con los elementos a ordenar se construye un heap.
  - En la segunda fase, una vez construido el heap, se desarma dicho heap y de esa forma obtendremos los valores ordenados.

CLASIFICACION – HEAP SORT





CLASIFICACION – HEAP SORT

Nombre	Mejor caso	Caso medio	Peor caso	Estable	Comentarios
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	Si	El más lento de todos. Uso pedagógico.
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Si	Apto si queremos que consumir siempre la misma cantidad de tiempo.
Insertion Sort	$O(n)$	$O(n)$	$O(n^2)$	Si	Conveniente cuando el array esta casi ordenado.
Shell Sort	$O(n^{5/4})$	$O(n^{3/2})$	$O(n^2)$	No	Dependiente de la secuencia de incrementos.
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Si	Adecuado para trabajos en paralelo.
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	No	El método acotado en el tiempo muy utilizado para grandes volúmenes de datos
QuickSort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	No	El más rápido en la práctica. Implementado en gran cantidad de sistemas.