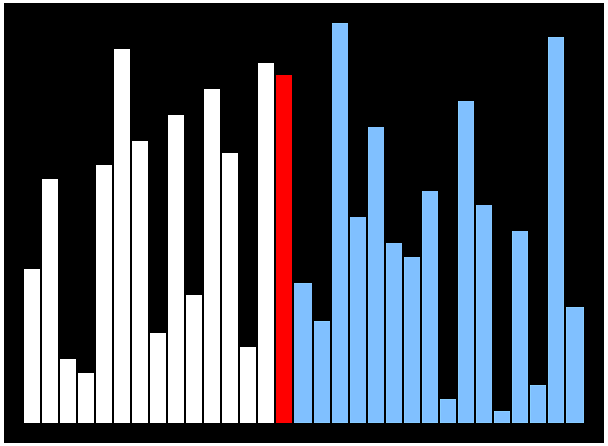
Algoritmos y Estructuras de Datos I

Facultad de Ciencias Exactas y Tecnología

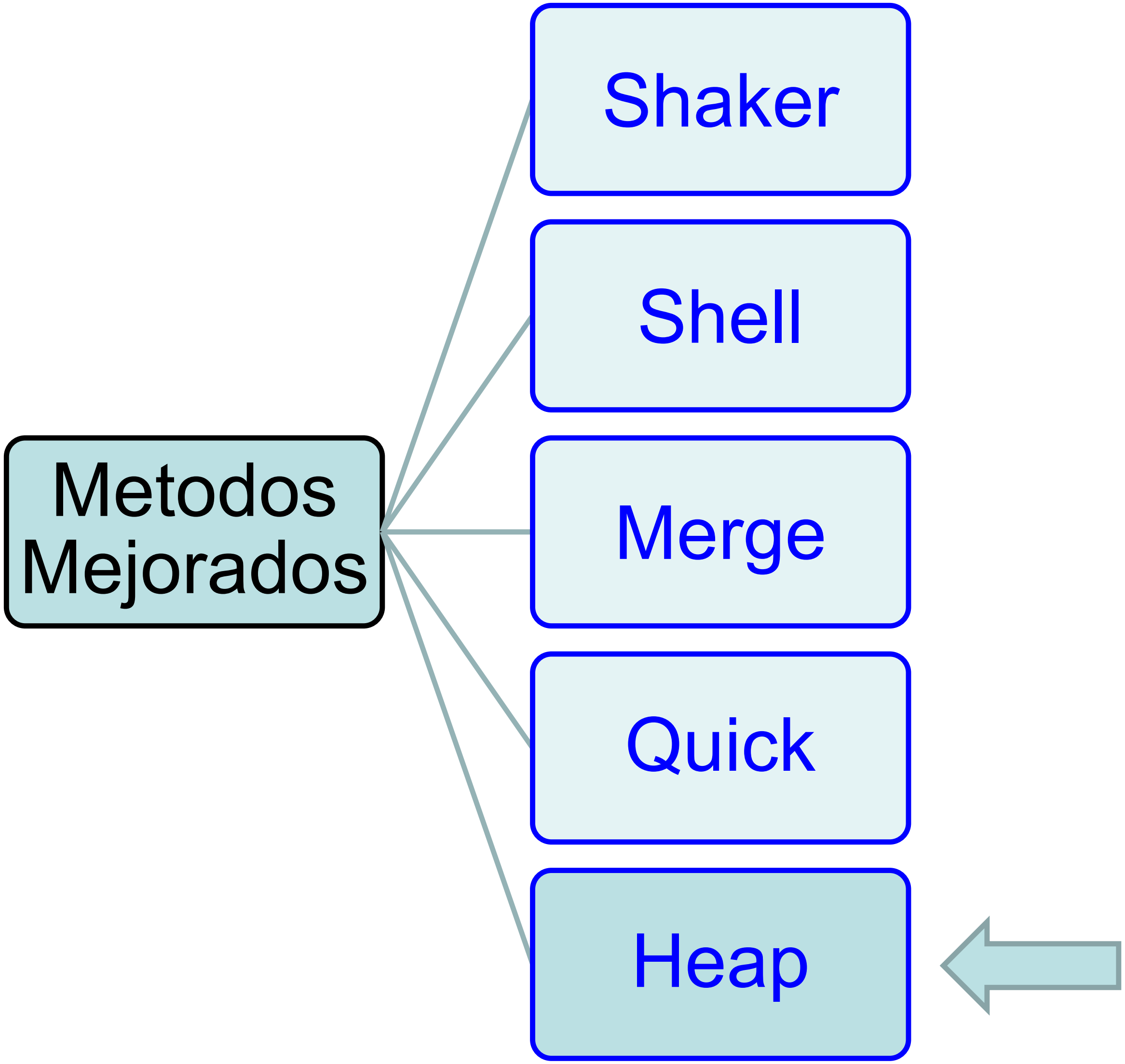
Universidad Nacional de Tucumán

2023

Ordenación(3)



Métodos de Ordenación Interna



MÉTODO del montículo o HEAPSORT

En 1964 J.W.J. Williams presenta este método cuya filosofía es la del método de selección directa, en el sentido de que se encuentra el mayor de una colección de n claves, luego la siguiente mayor y así.

En base a las operaciones de la estructura de datos **Cola de Prioridad** (PQ) se puede definir este método de manera elegante y eficiente.

La idea es simple, armar una PQ con los elementos a ordenar, luego sacarlos de PQ y reubicarlos en el mismo arreglo.

El método se puede implementar de manera que no use memoria extra, mejora propuesta por R. W. Floyd en ese mismo año.

Cola de prioridad (priority queue) **PQ(ITEM)**

Una ***Cola de Prioridad*** es una fila en la que cada ítem tiene asociada una prioridad con una relación de orden **>**.

La operación de selección siempre elige el elemento de mayor prioridad.

Especificación algebraica - OPERACIONES:

A) Sintaxis:

PQVACIA **:** → PQ
ENPQ **:** PQ X ITEM → PQ
ESPQVACIA **:** PQ → BOOLEAN
MAYOR **:** PQ → ITEM U {indefinido}
DEPQ **:** PQ → PQ

Cola de prioridad
PQ(ITEM)

Especificación algebraica - OPERACIONES:
B) Semántica: $\forall$ q $\in$ PQ , $\forall$ i $\in$ ITEM

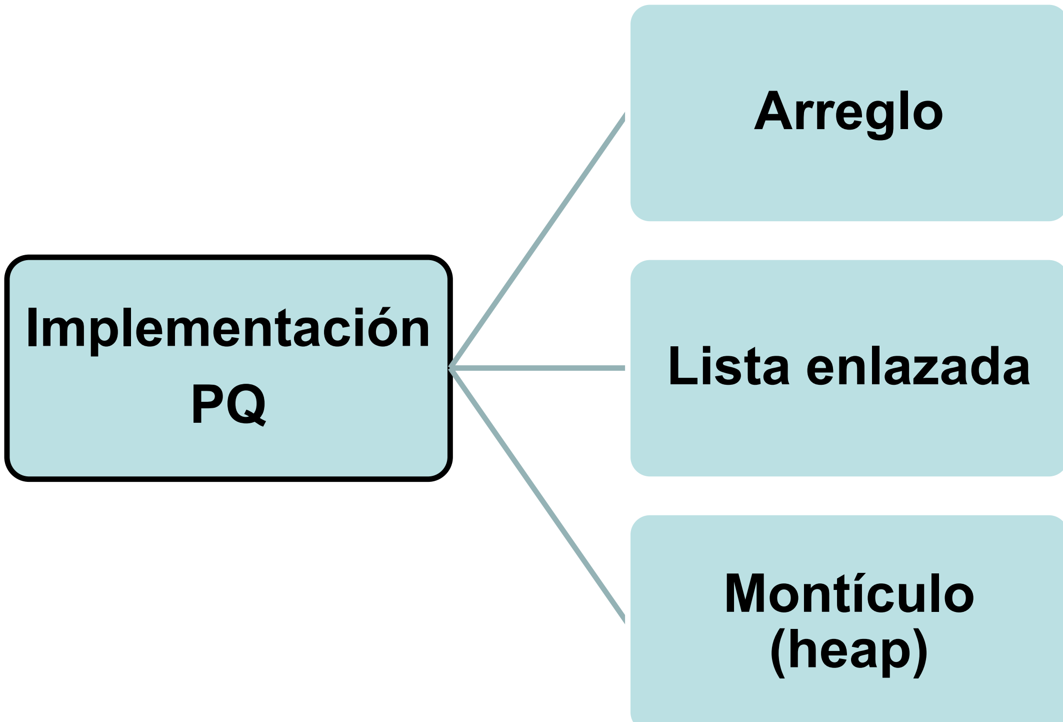
ESPQVACIA(PQVACIA) $\equiv$ TRUE
ESPQVACIA(ENPQ(q,i)) $\equiv$ FALSE

MAYOR(PQVACIA) $\equiv$ indefinido
MAYOR(ENPQ(q,i)) $\equiv$ si ESPQVACIA(q) entonces i
sino si i>MAYOR(q) entonces i
sino MAYOR(q)

DEPQ(PQVACIA) $\equiv$ PQVACIA
DEPQ(ENPQ(q,i)) $\equiv$ si ESPQVACIA(q) entonces PQVACIA
sino si i>MAYOR(q) entonces q
sino ENPQ(DEPQ(q),i)

PQ(ITEM)
Implementación

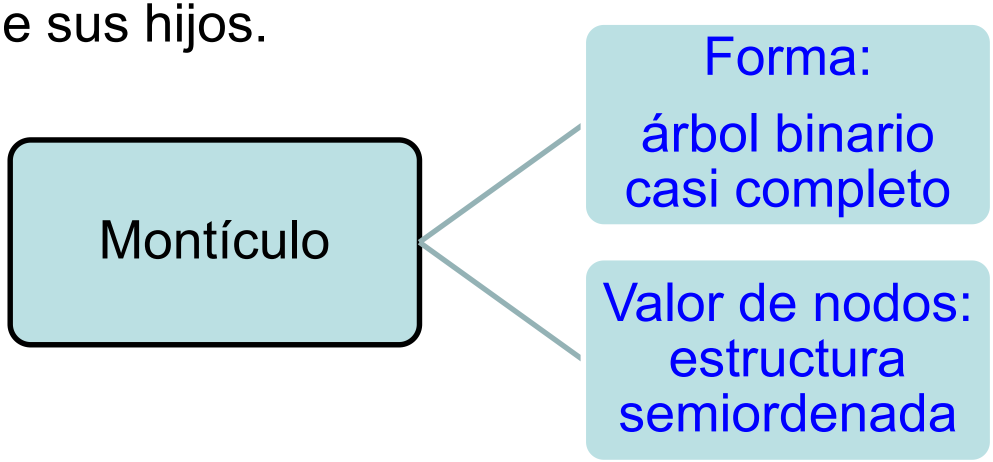
Se puede realizar la **Implementación de PQ** con:



Definición de Montículo

Un ***Montículo*** es un árbol que cumple:

- Es un ***árbol binario casi completo*****,** en el penúltimo nivel tiene todos los nodos, en el ultimo nivel las hojas se ubican de izquierda a derecha.
- Es una estructura ***semiordenada*****,** el ítem almacenado en cada nodo es mayor que sus hijos.



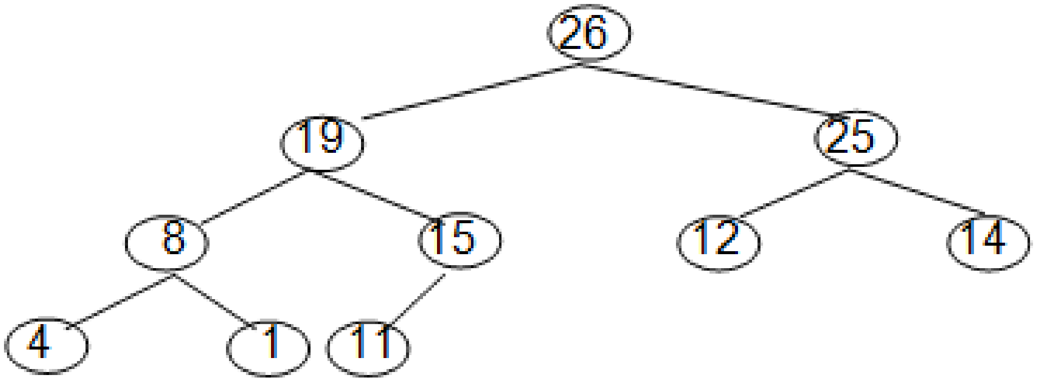
Un montículo **se implementa con un arreglo** que es la estructura ideal para albergar los ab casi completos.

Implementación de un Montículo

Los arboles binarios casi completos se pueden representar secuencialmente en un arreglo.

- la raíz se ubica en el índice 1
- sus hijos en las posiciones 2 y 3
- el nivel siguientes en las posiciones 4, 5, 6 y 7

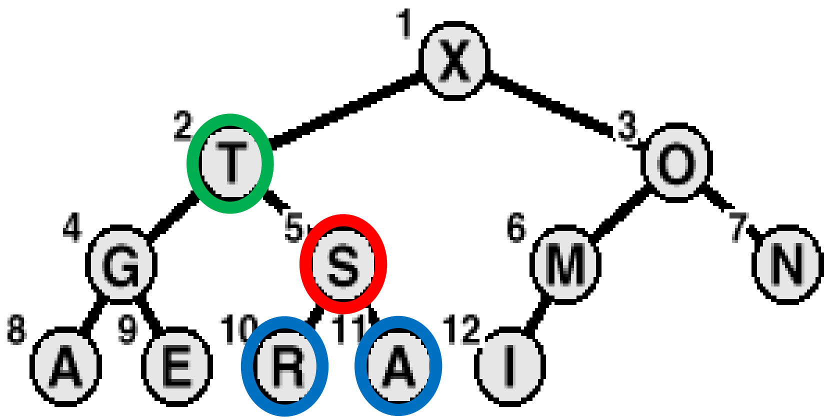
Por Ejemplo:



A = [26, 19, 25, 8, 15, 12, 14, 4, 1, 11]

Implementación de un Montículo

Ejemplo:



	Índice del arreglo
Raíz(árbol)	1
Padre(nodo(i))	i/2
HijoIzq(nodo(i))	2*i
HijoDer(nodo(i))	2*i+1

Arreglo:

padre — Hijo izquierdo

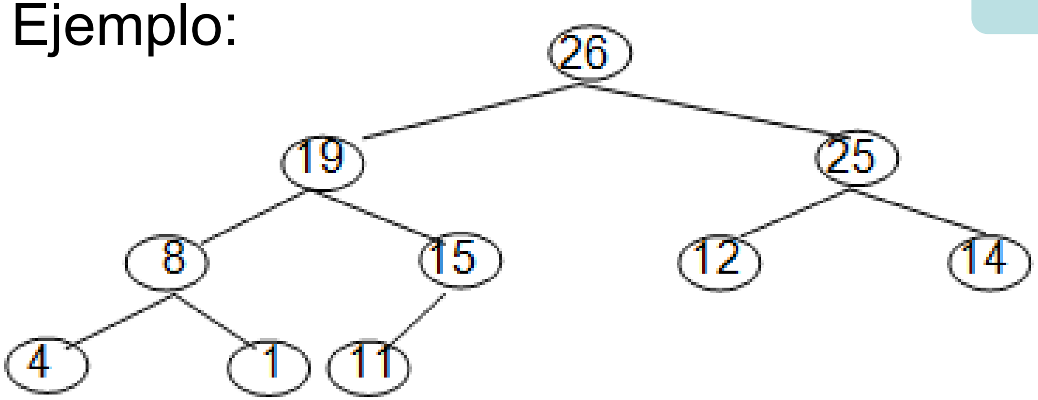
Índice:	0	1	2	3	4	5	6	7	8	9	10	11	12
Valor:		X	T	O	G	S	M	N	A	E	R	A	I

Hijo derecho

Operaciones para Montículo

Operaciones
- Seleccionar el mayor
- Insertar un nodo
- Borrar un nodo

Ejemplo:



Operaciones para Montículo

INSERTAR un item: la inserción de un nuevo nodo debe mantener la condición de heap, debe seguir siendo un ab casi completo y semiordenado.

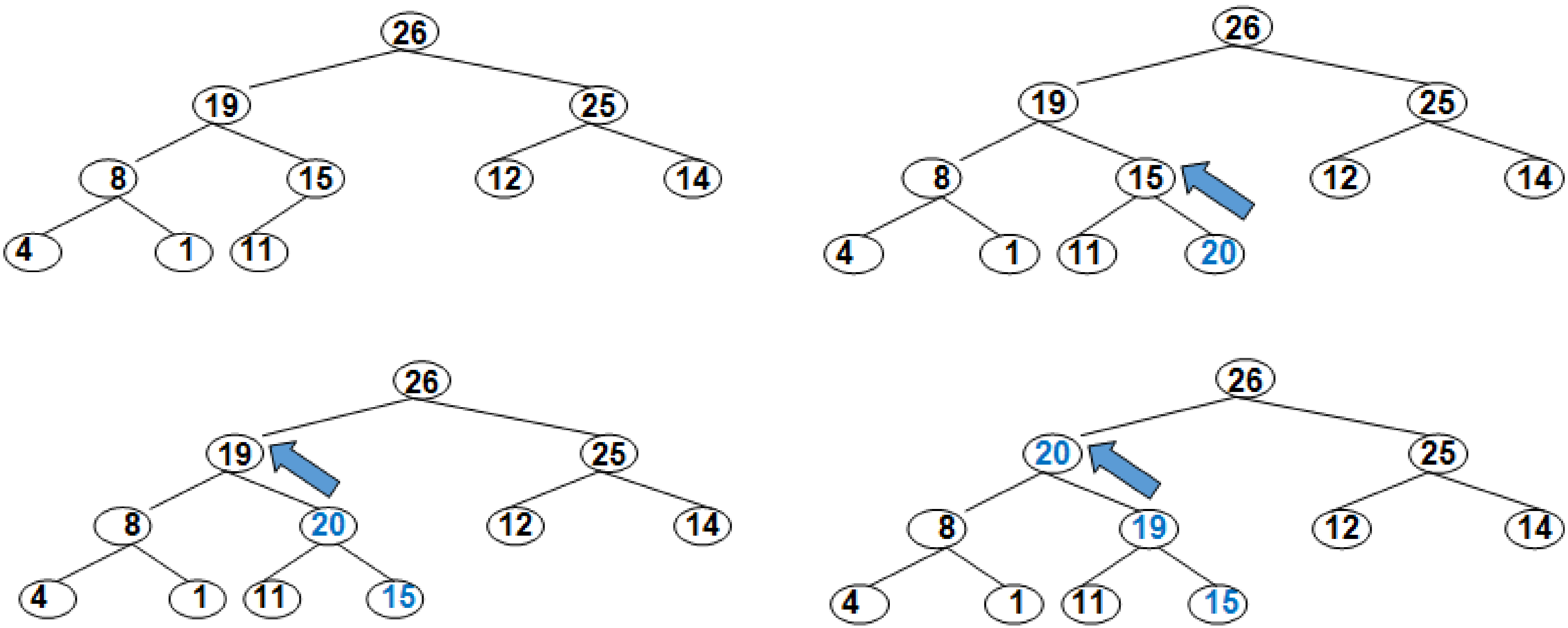
El proceso es el siguiente:

- Si el heap tiene **n** nodos, se inserta en la *primera componente libre* del arreglo, la **n+1**-ésima.
- Si viola con su valor la condición de heap, se lo *intercambia con su padre*. Y así se sigue con los intercambios hasta que ocupe su lugar sin violar la condición de orden o hasta llegar a la raíz del árbol.

El algoritmo para que el ítem suba a su posición final es Arriba:

Algoritmos para Montículo

Ejemplo Algoritmo Arriba: insertar 20



Algoritmos para Montículo

Algoritmo Arriba (A, k)

ENTRADA:
A $\in$ ARREGLO(item), de tamaño n
k: índice de A donde está ubicado el item.
SALIDA:
A $\in$ ARREGLO(item), con todos sus elementos en condición de heap
Auxiliares: aux $\in$ item

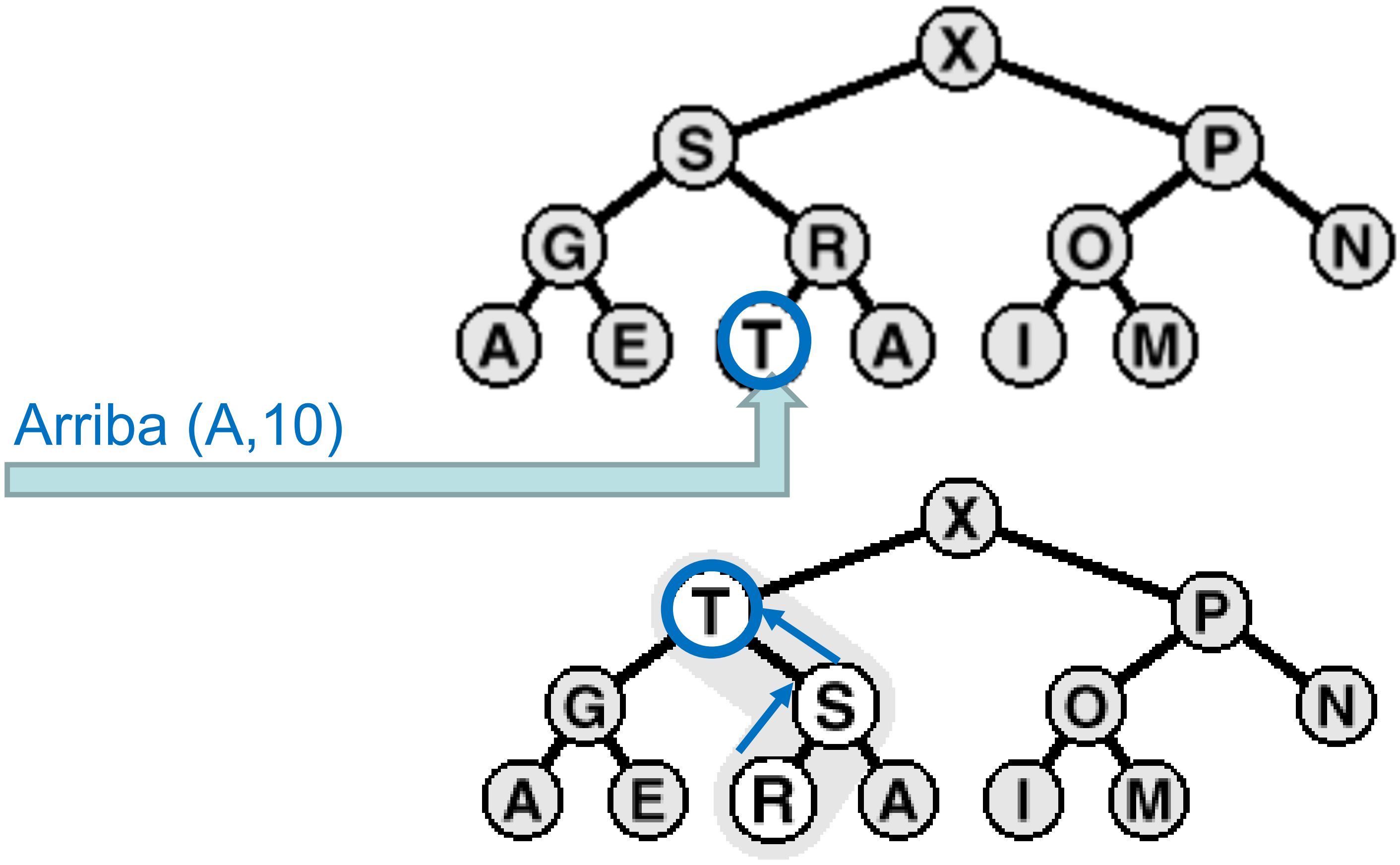
aux $\leftarrow$ A(k); *// se guarda el valor del nodo que tiene que subir en el árbol*
MIENTRAS (k > 1) AND (A(k / 2) $\leq$ aux) HACER *// tiene padre y es menor que el hijo*
    A(k) $\leftarrow$ A(k / 2) *// se reemplaza al hijo por su padre*
    k $\leftarrow$ k / 2 *// se sube un nivel en el arbol*
A(k) $\leftarrow$ aux; *// se coloca al nodo en su posición final*

FIN

Peor caso para este algoritmo es $O(\log_2 n)$, donde n es el nro. de ítems del arreglo.

Algoritmos para Montículo

Ejemplo: aplicación de Arriba:



Operaciones para Montículo

BORRAR un item: la operación más frecuente es borrar el item de mayor valor del heap, esto es el que está en la raíz del árbol.

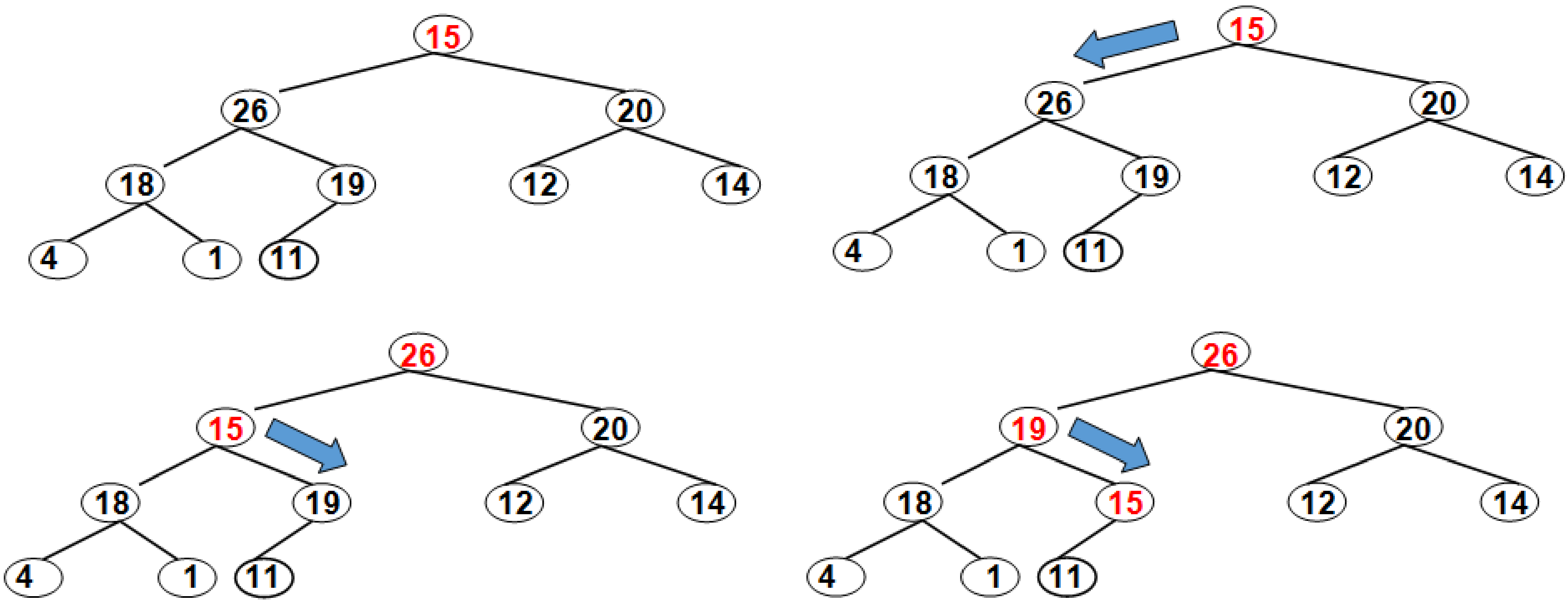
El procedimiento es el siguiente:

- Para borrar el mayor elemento, primero se lo sustituye por el nodo ce la *última hoja* que pasa a ser la raíz.
- Para restablecer la condición de heap si es que no se cumple que sea mayor que sus hijos, se lo *intercambia con el mayor de sus hijos*, y así se sigue con los intercambios mientras corresponda o hasta que se llegue a una hoja.

El algoritmo para que el ítem en la raíz baje a su posición definitiva es Abajo:

Algoritmos para Montículo

Ejemplo Algoritmo Abajo



Algoritmos para Montículo

Algoritmo Abajo (A,tamanio)
ENTRADA: A $\in$ ARREGLO(item), tamanio: tamaño real de A
SALIDA: A $\in$ ARREGLO(item), con todos sus elementos en condición de heap
Auxiliares: aux $\in$ item, sigue $\in$ bool

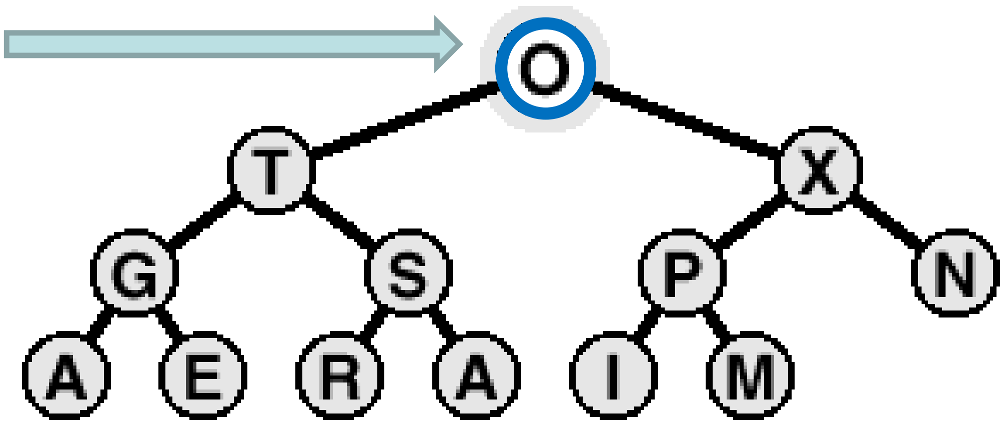
```
k ← 1; aux ← A(k)  ; sigue ← true
MIENTRAS ( k ≤ tamanio/2 AND sigue ) HACER // tiene al menos un hijo
    j ← 2*k;    // hijo izquierdo
    SI  j<tamanio AND A(j) ≤ A(j+1) ENTONCES // ∃ el hijo derecho y es mayor que el izq.
        j ← j+1                       // en índice j esta el mayor de sus hijos
    SI aux ≥ A(j) ENTONCES            // aux ya es mayor que sus hijos
        sigue ← false                 // no itera mas
    SINO
        A(k) ← A(j)                   // reemplaza al padre por el mayor de sus hijos
        k ← j                         // se baja un nivel en el árbol
    FIN SI
FIN MIENTRAS
A(k) ← aux // se coloca al nodo en su posición final
FIN
```

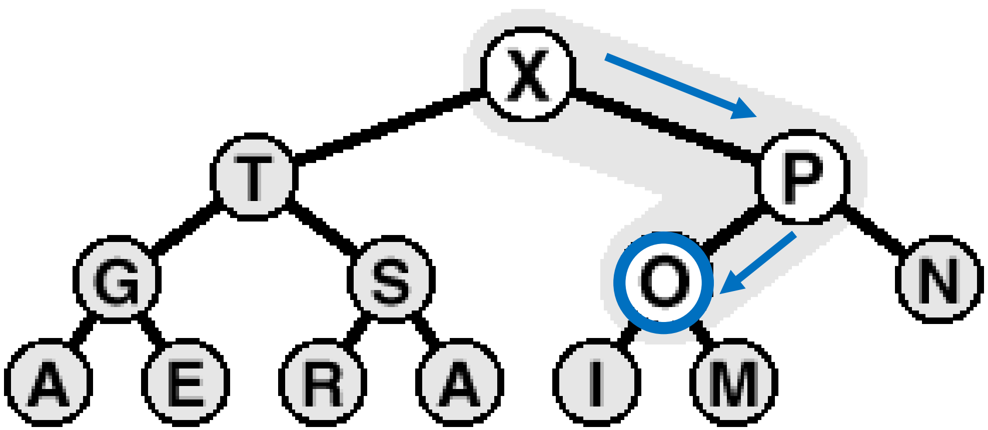
Peor caso para este algoritmo es $O(\log_2 n)$, donde n es el nro. de ítems.

Algoritmos para Montículo

Ejemplo: aplicación de Abajo:

Abajo(A,13)





Operaciones para PQ

Costo de las operaciones de PQ implementada con Monticulo:

Si la PQ tiene n items:

- PQVACIA $\in O(1)$
- ENPQ $\in O(\log_2 n)$
- ESPQVACIA $\in O(1)$
- MAYOR $\in O(1)$
- DEPQ $\in O(\log_2 n)$

MÉTODO HEAPSORT

Algoritmo **Heap** **(A,n)**
Entrada: A: arreglo de (1.. MAX) elementos de tipoitem
n: nro. entero de datos a ordenar
Salida: A arreglo ordenado
Auxiliares: pq $\in$ tipopq; x$\in$tipoitem ; i $\in$ entero

pq ← **pqvacia**; → ϵ O(1)
PARA i=1,n HACER → nit=n
    enpq(pq,A(i)) → ϵ O($\log_2 n$)
ϵ O($n\log_2 n$)
PARA i=n,1 paso (-1) HACER → nit=n
    x ←**mayor**(pq) → ϵ O(1)
    depq(pq) → ϵ O($\log_2 n$)
    A(i) ← x → ϵ O(1)
ϵ O($n\log_2 n$)
FIN

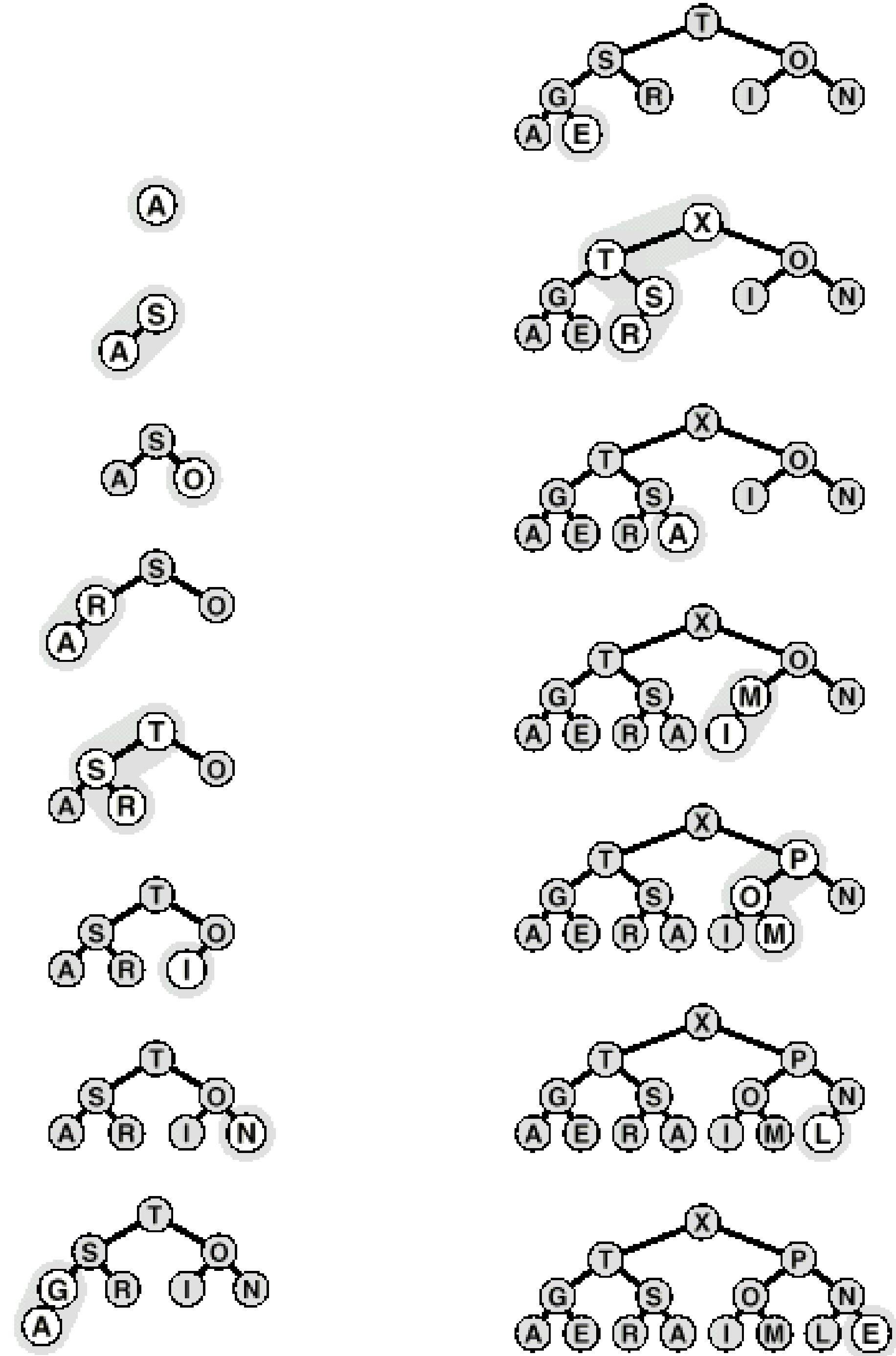
$\boxed{T(n)\ \epsilon\ O(n\log_2 n)}$

HEAPSORT

Ejemplo:. Ordenar con HeapSort la cadena: **ASORTINGEXAMPLE**

PASO 1)

Construcción de un heap (enpq) con los datos a ordenar:

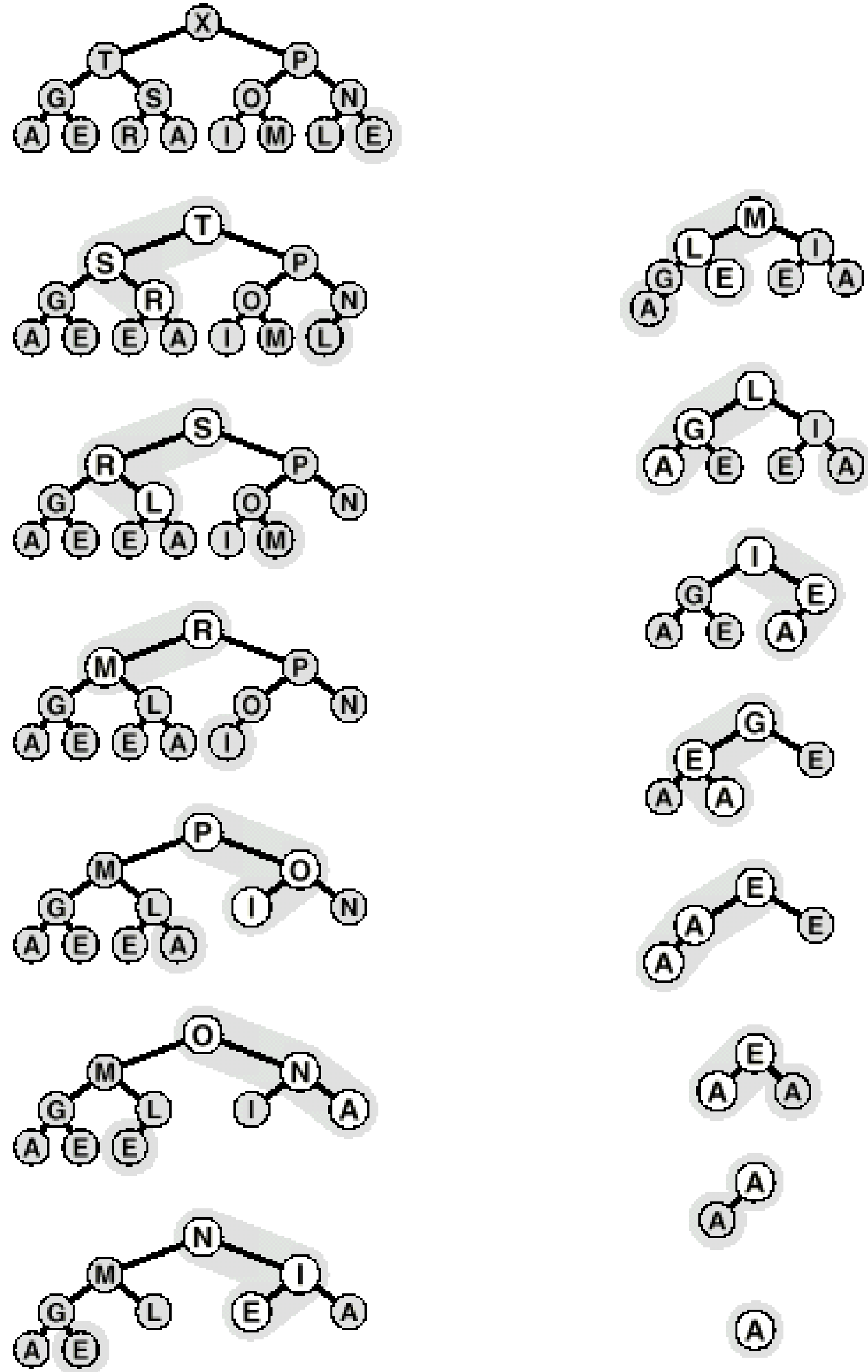


HEAPSORT

Ejemplo:

PASO 2)

Desarmado (depq) para ordenar según el heap: **XTSRPONMLIGEEAA**



Propiedades Algoritmo HEAP

Tiempo de Ejecución: Por la forma de implementación sobre una cola de prioridad Heapsort requiere tiempo proporcional a $O(n.\log_2 n)$ aun en el caso más desfavorable.

$$T(n) \in O(n \cdot \log_2 n)).$$

No es claro en que casos puede esperarse el mejor o peor rendimiento. Pero generalmente el método parece funcionar un poco mejor con secuencias iniciales en orden inverso.

Memoria: si se programa para que el arreglo y el montículo compartan el almacenamiento no necesita espacio extra.

Sensibilidad: nada sensible.

Estabilidad: NO es estable

TIEMPO DE EJECUCION DE DISTINTOS ALGORITMOS DE ORDENACIÓN

La tabla proporcionada por Wirth muestra el tiempo de ejecución en milisegundos consumido por distintos métodos. Las tres columnas corresponden a un arreglo ordenado, una permutación aleatoria y el arreglo ordenado en sentido inverso, todos del mismo tamaño. Los datos que se utilizan están formados sólo por una clave.

M E T O D O	Ordenado	Aleatorio	Inverso
SELECCION DIRECTA	1907	1956	2675
INSERCION DIRECTA	23	1444	2836
INTERCAMBIO DIRECTO	2165	4054	5931
SHAKER-SORT	9	3642	6520
SHELL-SORT	116	349	492
QUICK-SORT	69	146	79
MERGE-SORT	234	242	232
HEAP-SORT	253	241	226

ALGORITMOS DE ORDENACIÓN
Demos

Muy bueno:

- http://www.sorting-algorithms.com/

50+ Sorts, Visualized - Scatter Plot:

- https://www.youtube.com/watch?v=DSMCZZGbZo4

50+ Sorts, Visualized - Reversed Inputs:

- https://www.youtube.com/watch?v=qtRU2Xn76Bc

50+ Sorts, Visualized - Almost Sorted Inputs:

- https://www.youtube.com/watch?v=UdTyfJ4zJDA

METODOS LINEALES

Todos los métodos de ordenación presentados sirven para clasificar ítems con cualquier tipo de campo clave, siempre que en el mismo esté definido una relación de orden. Los algoritmos se definen en términos de operaciones básicas de comparaciones y movimientos.

Se ha demostrado que son necesarios $O(n\log_2 n)$ pasos para ordenar n elementos siempre que sea un método de propósito general para cualquier tipo de claves.

Los métodos con costo lineal no son aplicables a cualquier caso sino a un rango de problemas bien definidos. Sirven para un tipo particular de clave tal que tengan condiciones especiales como: ser números enteros y pertenecer a un cierto rango.

METODOS LINEALES

El método Radix tiene su origen en 1887 cuando se comenzó a usar por Herman Hollerith en las maquinas tabuladoras. En 1923 los algoritmos Radix se popularizaron con las tarjetas perforadas. Recién se programó para una computadora en 1954 (Harold Seward, MIT).

Actualmente Radix sort se usa frecuentemente para ordenar enteros y cadenas de bits. Se ha probado que es mucho mas rápido que cualquier otro metodo de propósito general.

Se puede implementar en sus 2 versiones:

- **LSD** (least significant digit): comienza ordenando las unidades.
- **MSD** (most significant digit):comienza con el dígito mas significativo.

Método Radix- Ejemplo

Método Radix : ***permite clasificar n claves numéricas de M dígitos en cualquier base B.***

EJEMPLO: Arreglo a ordenar:

A=[036 909 100 025 101 049 064 471 381 004
555 186 100 038 671 001 010 999 899 002]

Son n=20 claves numéricas de 3 dígitos decimales en base 10.

- Cada clave se descompone en sus 3 dígitos: d_1 d_2 d_3
- Se usará un algoritmo LSD, de modo que, primero se clasifica por unidades (d_3) , luego por decenas (d_2) y finalmente por centenas (d_1) .
- Se necesitan 10 filas auxiliares.

Método Radix- Ejemplo

ALGORITMO RADIX LST para claves de 3 dígitos ($d_1d_2d_3$):

P0. Leer el vector

P1. Clasificar según las unidades: d_3

P2. Concatenar las filas en orden

P3. Clasificar según las decenas: d_2

P4. Concatenar las filas en orden

P5. Clasificar según las centenas: d_1

P6. Escribir el vector ordenado

P7. FIN

Método Radix- Ejemplo

1º. Clasificar según las unidades (dígito menos significativo)

A=[036 909 100 025 101 049 064 471 381 004
555 186 100 038 671 001 010 999 899 002]

No. fila	frente				final
Fila 0	100	100	010		
Fila 1	101	471	381	671	001
Fila 2	002				
Fila 3					
Fila 4	064	004			
Fila 5	025	555			
Fila 6	036	186			
Fila 7					
Fila 8	038				
Fila 9	909	049	999	899	

Concatenar las filas en orden:

A=[100 100 010 101 471 381 671 001 002 064
004 025 555 036 186 038 909 049 999 899]

Método Radix- Ejemplo

2º. Clasificar según las decenas

A=[100 100 010 101 471 381 671 001 002 064
004 025 555 036 186 038 909 049 999 899]

,

Fila 0	100	100	101	001	002	004	909
Fila 1	010						
Fila 2	025						
Fila 3	036	038					
Fila 4	049						
Fila 5	555						
Fila 6	064						
Fila 7	471	671					
Fila 8	381	186					
Fila 9	999	899					

Concatenar las filas en orden:

A=[100 100 101 001 002 004 909 010 025 036
038 049 555 064 471 671 381 186 999 899]

Método Radix- Ejemplo

3º. Clasificar según las centenas

A=[100 100 101 001 002 004 909 010 025 036
038 049 555 064 471 671 381 186 999 899]

Fila 0	001	002	004	010	025	036	038	049	064
Fila 1	100	100	101	186					
Fila 2									
Fila 3	381								
Fila 4	471								
Fila 5	555								
Fila 6	671								
Fila 7									
Fila 8	899								
Fila 9	909	999							

Concatenar las filas en orden:

A=[001 002 004 010 025 036 038 049 064 100
100 101 186 381 471 555 671 899 909 999]

Algoritmo Radix

Algoritmo Radix (A,n,M)

Entrada:

A: arreglo de (1.. MAX) elementos de enteros en base 10
n: nro. entero de datos a ordenar
M: cantidad de dígitos de la clave

Salida:

A: arreglo ordenado.

Auxiliar:

b, exp, i, j, k Є enteros,
q(10) Є arreglo de filas de enteros (base 10)

Algoritmo Radix

```
Algoritmo Radix (A,n,M)
PARA k=M,1,paso -1 HACER
        PARA j=0,9 HACER
                FILAVACIA(q(j))
        PARA i=1,n HACER
                b ← A(i)
                j ← k-esimo digito de b
                ENFILA(q(j),b)
        i ← 1
        PARA j=0,9 HACER
                MIENTRAS No ESFILAVACIA(q(j)) HACER
                        A(i) ← FRENTE(q(j))
                        DEFILA(q(j))
                        i←i+1
FIN
```

Costo Algoritmo Radix

Algoritmo Radix (A,n,M)

```
PARA k=M,1,paso -1 HACER                      → nit=M
        PARA j=0,9 HACER                      → nit=10
            FILAVACIA(q(j))                   → ϵ O(1)          ϵ O(1)
        PARA i=1,n HACER                      → nit=n
            b ← A(i)                          → ϵ O(1)
            j ← k-esimo digito de b           → ϵ O(1)          ϵ O(n)
            ENFILA(q(j),b)                    → ϵ O(1)                    ϵ O(n.M)
        i ← 1                                 → ϵ O(1)
        PARA j=0,9 HACER
            MIENTRAS No ESFILAVACIA(q(j)) HACER     nit=n
                A(i) ← FRENTE(q(j))           → ϵ O(1)          ϵ O(n)
                DEFILA(q(j))                  → ϵ O(1)
                i←i+1                         → ϵ O(1)
FIN
```

Método Radix- Ejemplo

A_0=(ASORTINGEXAMPLE)

Elegir un código binario para cada letra:

A:00001, B:00010, C:00011, D:00100, E:00101 …

K=5

Filas:

q(0)=

q(1)=

A	0 0 0 0 1
S	1 0 0 1 1
O	0 1 1 1 1
R	1 0 0 1 0
T	1 0 1 0 0
I	0 1 0 0 1
N	0 1 1 1 0
G	0 0 1 1 1
E	0 0 1 0 1
X	1 1 0 0 0
A	0 0 0 0 1
M	0 1 1 0 1
P	1 0 0 0 0
L	0 1 1 0 0
E	0 0 1 0 1

Método Radix- Ejemplo

A_0=(ASORTINGEXAMPLE)

K=5

Llevar las claves a las Filas:

q(0): RTNXPL

q(1): ASOIGEAME

Rearmar el Arreglo:

A_1=(RTNXPLASOIGEAME)

A	0 0 0 0 1
S	1 0 0 1 1
O	0 1 1 1 1
R	1 0 0 1 0
T	1 0 1 0 0
I	0 1 0 0 1
N	0 1 1 1 0
G	0 0 1 1 1
E	0 0 1 0 1
X	1 1 0 0 0
A	0 0 0 0 1
M	0 1 1 0 1
P	1 0 0 0 0
L	0 1 1 0 0
E	0 0 1 0 1

R	1 0 0 1	0
T	1 0 1 0	0
N	0 1 1 1	0
X	1 1 0 0	0
P	1 0 0 0	0
L	0 1 1 0	0
A	0 0 0 0	1
S	1 0 0 1	1
O	0 1 1 1	1
I	0 1 0 0	1
G	0 0 1 1	1
E	0 0 1 0	1
A	0 0 0 0	1
M	0 1 1 0	1
E	0 0 1 0	1

Método Radix- Ejemplo

A_1=(RTNXPLASOIGEAME)

K=4

Llevar las claves a las Filas:

q(0): TXPLAIEAME

q(1): RNSOG

Rearmar el Arreglo:

A_2=(TXPLAIEAMERNSOG)

R	1	0	0	1	0
T	1	0	1	0	0
N	0	1	1	1	0
X	1	1	0	0	0
P	1	0	0	0	0
L	0	1	1	0	0
A	0	0	0	0	1
S	1	0	0	1	1
O	0	1	1	1	1
I	0	1	0	0	1
G	0	0	1	1	1
E	0	0	1	0	1
A	0	0	0	0	1
M	0	1	1	0	1
E	0	0	1	0	1

T	1	0	1	0	0
X	1	1	0	0	0
P	1	0	0	0	0
L	0	1	1	0	0
A	0	0	0	0	1
I	0	1	0	0	1
E	0	0	1	0	1
A	0	0	0	0	1
M	0	1	1	0	1
E	0	0	1	0	1
R	1	0	0	1	0
N	0	1	1	1	0
S	1	0	0	1	1
O	0	1	1	1	1
G	0	0	1	1	1

Método Radix- Ejemplo

A_2=(TXPLAIEAMERNSOG)

K=3

Llevar las claves a las Filas:

q(0): XPAIARS

q(1): TLEMENOG

Rearmar el Arreglo:

A_3=(XPAIARSTLEMENOG)

T	1	0	1	0	0
X	1	1	0	0	0
P	1	0	0	0	0
L	0	1	1	0	0
A	0	0	0	0	1
I	0	1	0	0	1
E	0	0	1	0	1
A	0	0	0	0	1
M	0	1	1	0	1
E	0	0	1	0	1
R	1	0	0	1	0
N	0	1	1	1	0
S	1	0	0	1	1
O	0	1	1	1	1
G	0	0	1	1	1

X	1	1	0	0	0
P	1	0	0	0	0
A	0	0	0	0	1
I	0	1	0	0	1
A	0	0	0	0	1
R	1	0	0	1	0
S	1	0	0	1	1
T	1	0	1	0	0
L	0	1	1	0	0
E	0	0	1	0	1
M	0	1	1	0	1
E	0	0	1	0	1
N	0	1	1	1	0
O	0	1	1	1	1
G	0	0	1	1	1

Método Radix- Ejemplo

A_3=(XPAIARSTLEMENOG)

K=2

Llevar las claves a las Filas:

q(0): PAARSTEEG

q(1): XILMNO

Rearmar el Arreglo:

A_4=(PAARSTEEGXILMNO)

X	1	1	0	0	0
P	1	0	0	0	0
A	0	0	0	0	1
I	0	1	0	0	1
A	0	0	0	0	1
R	1	0	0	1	0
S	1	0	0	1	1
T	1	0	1	0	0
L	0	1	1	0	0
E	0	0	1	0	1
M	0	1	1	0	1
E	0	0	1	0	1
N	0	1	1	1	0
O	0	1	1	1	1
G	0	0	1	1	1

P	1	0	0	0	0
A	0	0	0	0	1
A	0	0	0	0	1
R	1	0	0	1	0
S	1	0	0	1	1
T	1	0	1	0	0
E	0	0	1	0	1
E	0	0	1	0	1
G	0	0	1	1	1
X	1	1	0	0	0
I	0	1	0	0	1
L	0	1	1	0	0
M	0	1	1	0	1
N	0	1	1	1	0
O	0	1	1	1	1

Método Radix- Ejemplo

A_4=(PAARSTEEGXILMNO)

K=1

Llevar las claves a las Filas:

q(0): AAEEGILMNO

q(1): PRSTX

Rearmar el Arreglo:

A_5=(AAEEGILMNOPRSTX)

P	1	0	0	0	0
A	0	0	0	0	1
A	0	0	0	0	1
R	1	0	0	1	0
S	1	0	0	1	1
T	1	0	1	0	0
E	0	0	1	0	1
E	0	0	1	0	1
G	0	0	1	1	1
X	1	1	0	0	0
I	0	1	0	0	1
L	0	1	1	0	0
M	0	1	1	0	1
N	0	1	1	1	0
O	0	1	1	1	1

A	0	0	0	0	1
A	0	0	0	0	1
E	0	0	1	0	1
E	0	0	1	0	1
G	0	0	1	1	1
I	0	1	0	0	1
L	0	1	1	0	0
M	0	1	1	0	1
N	0	1	1	1	0
O	0	1	1	1	1
P	1	0	0	0	0
R	1	0	0	1	0
S	1	0	0	1	1
T	1	0	1	0	0
X	1	1	0	0	0

Método Radix- Ejemplo

A	00001	R	10010	T	10100	X	11000	P	10000	A	00001
S	10011	T	10100	X	11000	P	10000	A	00001	A	00001
O	01111	N	01110	P	10000	A	00001	A	00001	E	00101
R	10010	X	11000	L	01100	I	01001	R	10010	E	00101
T	10100	P	10000	A	00001	A	00001	S	10011	G	00111
I	01001	L	01100	I	01001	R	10010	T	10100	I	01001
N	01110	A	00001	E	00101	S	10011	E	00101	L	01100
G	00111	S	10011	A	00001	T	10100	E	00101	M	01101
E	00101	O	01111	M	01101	L	01100	G	00111	N	01110
X	11000	I	01001	E	00101	E	00101	X	11000	O	01111
A	00001	G	00111	R	10010	M	01101	I	01001	P	10000
M	01101	E	00101	N	01110	E	00101	L	01100	R	10010
P	10000	A	00001	S	10011	N	01110	M	01101	S	10011
L	01100	M	01101	O	01111	O	01111	N	01110	T	10100
E	00101	E	00101	G	00111	G	00111	O	01111	X	11000

Algoritmo Radix

Tiempo de Ejecución:

Para ordenar un vector de ***n*** datos de ***M*** dígitos cada uno:

$$T(n) \in O(\boldsymbol{n \cdot M}).$$

Memoria: Si los datos están en base ***b*****,** se usa espacio extra para implementar las ***b*** filas.

Sensibilidad: nada sensible.

Estabilidad: si es estable

Otros

- Puede usarse para listas enlazadas
- Puede diseñarse una versión recursiva.
- Puede paralelizarse en su versión recursiva.