



# Análisis y Diseño de Algoritmos.

**Sesión 3.** 29 de Agosto de 2017.

Maestría en Sistemas Computacionales.



# ¿Qué veremos hoy?

- ▶ Diseño y análisis de algoritmos de ordenamiento:
  - ▶ de complejidad cuasi-lineal
  - ▶ iterativos
- ▶ ¿Cuáles son?
  - ▶ Shell (ya se vio)
  - ▶ Heapsort
  - ▶ Radix
- ▶ Ordenamiento por conteo: complejidad lineal

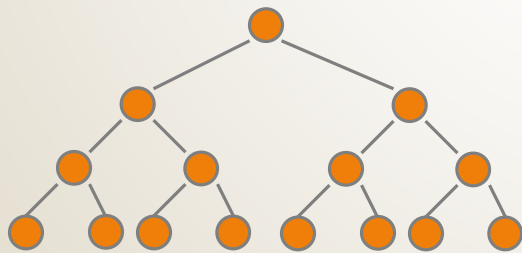
# Heap Sort

- ▶ También llamado ordenamiento por *montículos*.
- ▶ Su complejidad temporal es  $N \log N$  en los casos mejor, peor y promedio.
- ▶ Quicksort es con frecuencia más rápido, pero HeapSort es mejor en los casos críticos.
- ▶ El primer paso del algoritmo consiste en construir un montículo (heap) a partir del arreglo.
- ▶ El segundo paso (e iterativo) es eliminar el elemento más grande y sustituirlo por el que está colocado al final del montículo.

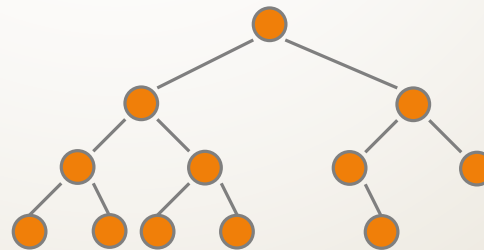
# ¿Qué es un montículo?

1 / 2

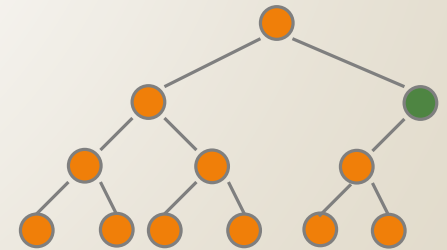
- Es un árbol binario con estas características:
  1. Cada nodo tiene un **valor comparable** tal que ningún nodo tiene un valor más grande que el de su padre.
  2. Está **balanceado**: cada nodo tiene 2 hijos, excepto los de los últimos dos niveles.
  3. Está alineado a la **izquierda**: si un nodo sólo tiene un hijo, debe ser el izquierdo.



Balanceado



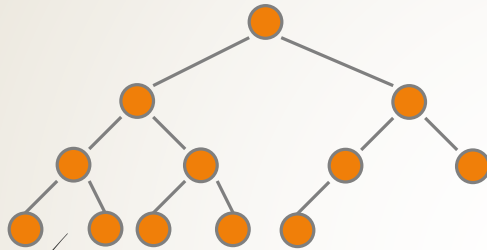
Balanceado



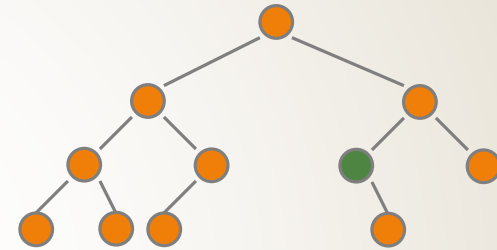
No  
balanceado

# ¿Qué es un montículo?

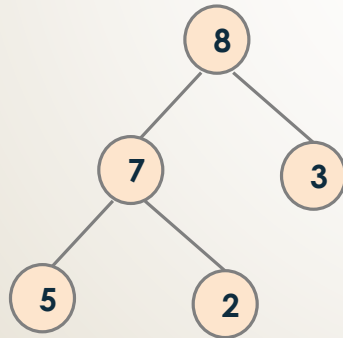
2 / 2



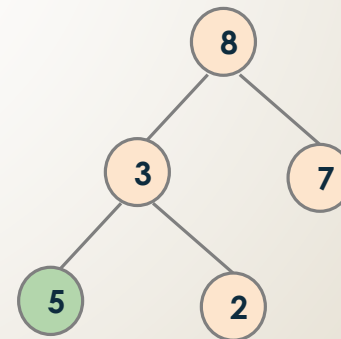
Alineado a la izquierda



No alineado a la izquierda



Cumple propiedad 1

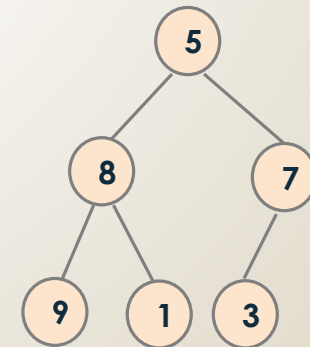


No cumple propiedad 1

# ¿Por qué con montículos?

1. El cumplimiento de la propiedad 1 nos facilitará realizar el ordenamiento en tiempo quasi-lineal.
  2. El cumplimiento de las propiedades 2 y 3 nos permitirán tratar al arreglo recibido como árbol (sin necesidad de crear un árbol binario explícito).
- Correspondencia entre arreglo y árbol binario:
- Nótese que no cumple con la propiedad 1.

5 8 7 9 1 3

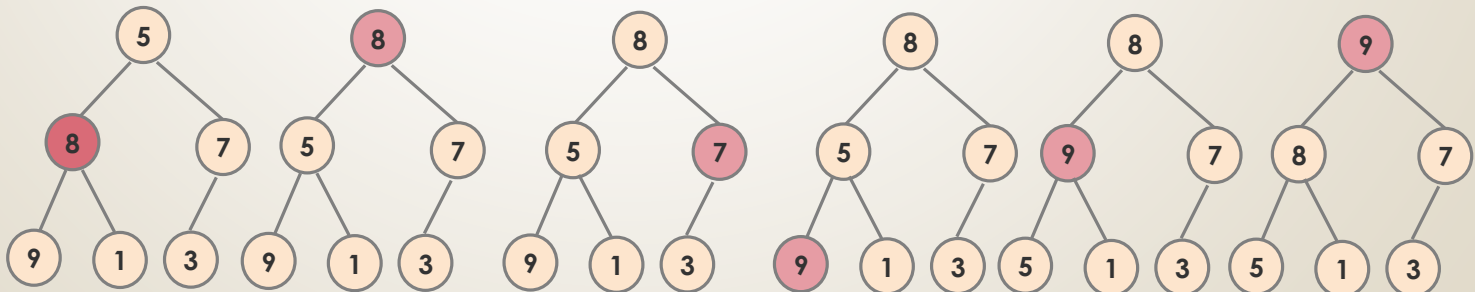


# Primer paso

## ► Convertir el arreglo en montículo:

1. Comenzar en el 2º elemento: hijo izquierdo de la raíz.
2. Obtener el índice del padre... ¿cuál es la fórmula?
3. Si el elemento actual es mayor que su padre:
  - a) Intercambiarlos
  - b) Regresar al paso 3 hasta llegar a la raíz
4. Regresar al paso 1 con el siguiente elemento.


5 8 7 9 1 3 → 9 8 7 5 1 3







# Ejercicio:

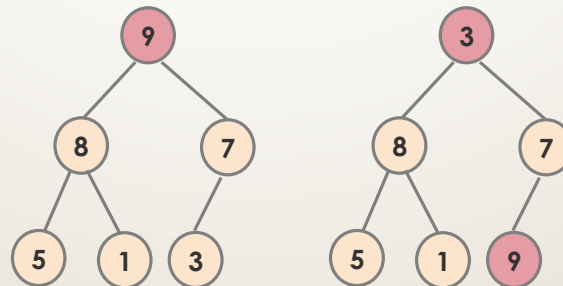
1. Formar el montículo del siguiente arreglo [7, 8, 6, 5, 12, 25, 28]
  2. Formar el montículo a partir de un arreglo con los valores del 1 al 7
- 



# Segundo paso

1 / 3

- Eliminar iterativamente el elemento más grande y sustituirlo por el que está al final del montículo.
  1. El elemento más grande siempre está en la raíz.
  2. Intercambiar el elemento de la raíz por el del final.
- El elemento más grande ya quedó en su posición definitiva: esa posición ya no se visitará y el nuevo final del montículo es la posición anterior.
- Se pierde la propiedad 1 de los montículos.



# Segundo paso

2 / 3

3. Llevar la nueva raíz tan abajo como sea necesario hasta que se recupere la propiedad 1 de los montículos: *push-down*.

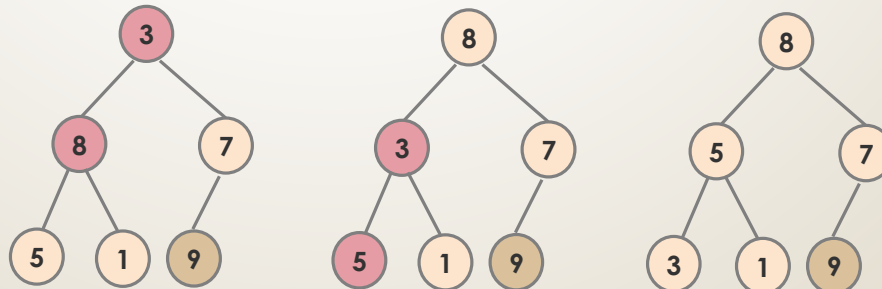
a. Compararlo con el mayor de los hijos.

¿Cuál es la fórmula para acceder a los dos hijos?

- Si el hijo derecho no existe, el mayor será el hijo izquierdo.
- No se puede dar que el hijo izquierdo no existe y el derecho sí.

b. Si es menor,

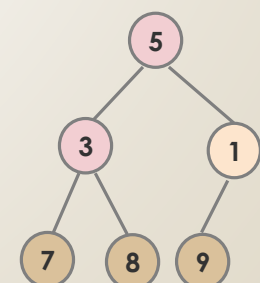
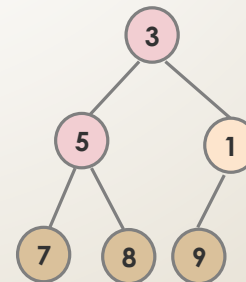
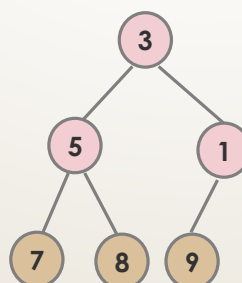
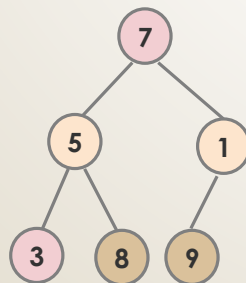
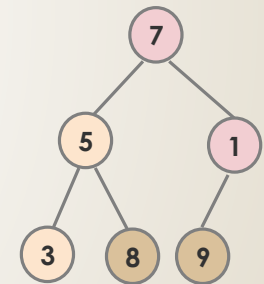
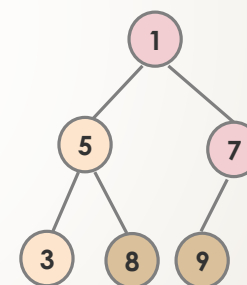
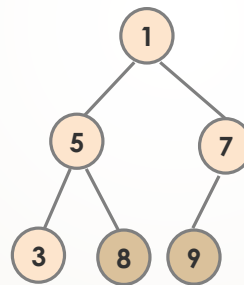
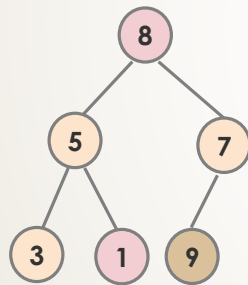
- Intercambiarlos.
- Regresar al paso 3.a hasta llegar a las hojas del montículo.



# Segundo paso

3 / 3

4. Regresar al paso 2 tomando como último elemento del montículo, el anterior al elegido en la pasada anterior.



...

# Análisis de Heapsort

1 / 6

- ▶ Los pasos 1 y 2 llevan la misma lógica en el recorrido del arreglo, pero en sentido contrario: analizaremos uno y multiplicaremos por 2 el resultado.
- ▶ Nos interesa saber cuánto es lo más que puede tardar: peor caso.
  - ▶ El peor caso es cuando cada elemento actual tiene que recorrerse siempre hasta la raíz o hasta la hoja.
- ▶ Tomaremos dos casos extremos y calcularemos promedio:
  1. El último nivel del árbol está lleno.
  2. El último nivel del árbol tiene una hoja.
- ▶ ¿Por qué sí podemos calcular promedio?
  - ▶ La longitud del recorrido de todos los nodos del mismo nivel es igual.

# Análisis de Heapsort

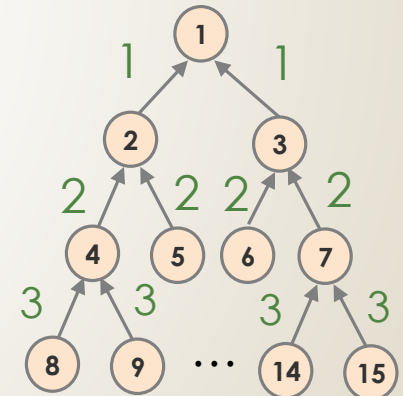
2 / 6

1. El último nivel está lleno.

- ▶  $N = 3, 7, 15, 31, 63, \dots$  [ $N = 1$  es un caso trivial,  $f(1) = 0$ ]
- ▶ # de niveles de un árbol binario de  $N$  elementos:
- ▶  $\lfloor \log_2(N) \rfloor + 1$

**$f(N)$  = movimientos**

N	$f(N)$
3	$2(1)$
7	$4(2) + f(3)$
15	$8(3) + f(7)$
31	$16(4) + f(15)$
63	$32(5) + f(31)$
N	$\frac{1}{2}(N + 1) (\lg(N + 1) - 1) + f(\frac{1}{2} N)$



# Análisis de Heapsort

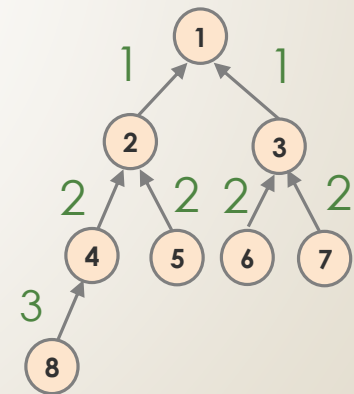
3 / 6

N	f(N)	f(N) / N	Log (N + 1) - 2
3	2	0.66	0
7	10	1.42	1
15	34	2.26	2
31	98	3.16	3
63	258	4.09	4
127	642	5.05	5
255	1,538	6.03	6
511	3,586	7.01	7
N	$f(N) \approx N (\lg(N + 1) - 2)$		

# Análisis de Heapsort

2. El último nivel tiene una hoja.
- ▶  $N = 2, 4, 8, 16, 32, 64, \dots$
  - ▶ Estará en función del caso anterior.

N	f(N)
2	1
4	$f(3) + 2$
8	$f(7) + 3$
16	$f(15) + 4$
32	$f(31) + 5$
N	$f(N - 1) + \lg(N)$





# Análisis de Heapsort

5 / 6

N	f(N)	f(N) / N	Log(N) - 2
2	1	0.5	-1
4	4	1	0
8	13	1.62	1
16	38	2.38	2
32	103	3.21	3
64	264	4.12	4
128	649	5.07	5
256	1546	6.04	6
512	3595	7.02	7
N	$f(N) \approx N (\lg(N) - 2)$		

# Análisis de Heapsort

6 / 6

- ▶ Falta calcular el promedio de los dos resultados obtenidos, y luego multiplicar por dos debido a los dos pasos del algoritmo; por tanto, sólo hay que sumar los dos resultados:
  1.  $N (\lg(N + 1) - 2)$
  2.  $N (\lg N - 2)$
- ▶  $N (\lg(N + 1) + \lg N - 4) \approx 2N(\lg N - 4) \in O(N \lg N)$ 
  - ▶ Tienden a ser iguales conforme  $N$  crece.
  - ▶ Esta ecuación representa el promedio de los peores casos.
  - ▶ Nota: este análisis no contempla comparaciones entre los hijos.
- ▶ Comprobar de manera práctica que un arreglo aleatorio efectúa menos del 60% de comparaciones, uno ordenado efectúa poco más del 100%, y uno invertido es el mejor caso de los 3 ( $N \geq 10^6$ ).

# Radixsort

1 / 2

- ▶ El ordenamiento se basa en procesar los dígitos que componen cada número de forma individual
- ▶ Se hacen tantas pasadas como número de dígitos tenga el número más grande
  - ▶ Se crean 10 listas de tamaño variable
  - ▶ En la primer lista se guardarán todos aquellos que terminan en 0; la segunda lista los que terminan con 1, y así ....
  - ▶ En la siguiente pasada, se crean otras 10 listas
  - ▶ En la primer lista nueva se guardarán los números cuyo penúltimo dígito es 0, ..., siguiendo el orden que tenían en las 10 listas de la pasada anterior

# Radixsort

- $List_a = \{5, 67, 58, 34, 25, 31, 19, 20, 9, 24, 26, 17, 10, 16, 52\}$ 
    - $List_{a_0} = \{20, 10\}$
    - $List_{a_2} = \{52\}$
    - $List_{a_4} = \{34, 24\}$
    - $List_{a_6} = \{26, 16\}$
    - $List_{a_8} = \{58\}$
  - $List_{a_0} = \{05, 09\}$
  - $List_{a_2} = \{20, 24, 25, 26\}$
  - $List_{a_4} = \{\}$
  - $List_{a_6} = \{67\}$
  - $List_{a_8} = \{\}$
- 
- $List_{a_1} = \{31\}$
  - $List_{a_3} = \{\}$
  - $List_{a_5} = \{5, 25\}$
  - $List_{a_7} = \{67, 17\}$
  - $List_{a_9} = \{19, 9\}$
  - $List_{a_1} = \{10, 16, 17, 19\}$
  - $List_{a_3} = \{31, 34\}$
  - $List_{a_5} = \{52, 58\}$
  - $List_{a_7} = \{\}$
  - $List_{a_9} = \{\}$

<http://cs.armstrong.edu/liang/animation/web/RadixSort.html>

# Ordenamiento por Conteo 1 / 3

- ▶ Veamos un caso especial:
  - ▶ Ordenar una lista de  $N$  enteros diferentes con valores de 0 a  $N - 1$ :
  - ▶  $4, 3, 5, 1, 6, 0, 2 \Rightarrow 0, 1, 2, 3, 4, 5, 6$
  - ▶ ¿Se puede lograr con una complejidad temporal menor a  $N \lg N$ ?
  - ▶ El valor estará en función de la posición:  $O(N)$ .
- ▶ Otro caso especial (más interesante):
  - ▶ Ordenar una lista de  $N$  enteros con valores de 0 a  $M - 1$ ,  $M < N$ :
  - ▶  $3, 1, 3, 1, 2, 1, 0 \Rightarrow 0, 1, 1, 1, 2, 3, 3$

# Ordenamiento por Conteo 2 / 3

► La lista a ordenar es: 3, 1, 3, 1, 2, 1, 0

#0 #1 #2 #3

1. Contar ocurrencias de cada valor: Conteos = {1, 3, 1, 2}
2. Acumular los conteos: Conteos = {1, 4, 5, 7}
3. Proceder de derecha a izquierda y escribir en una nueva lista:

[0] Conteos = {0, 4, 5, 7}. Lista' = {0, , , , , }.

[1] Conteos = {0, 3, 5, 7}. Lista' = {0, , , 1, , }.

[2] Conteos = {0, 3, 4, 7}. Lista' = {0, , , 1, 2, , }.

[1] Conteos = {0, 2, 4, 7}. Lista' = {0, , 1, 1, 2, , }.

[3] Conteos = {0, 2, 4, 6}. Lista' = {0, , 1, 1, 2, , 3}.

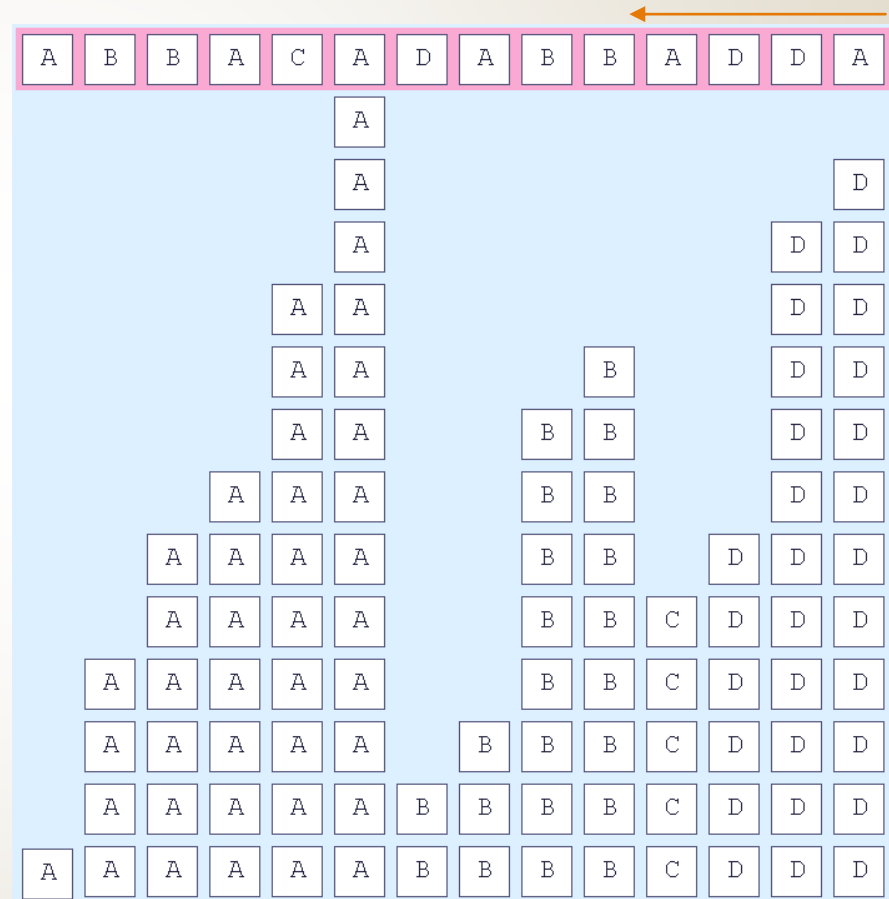
[1] Conteos = {0, 1, 4, 6}. Lista' = {0, 1, 1, 1, 2, , 3}.

[3] Conteos = {0, 1, 4, 5}. Lista' = {0, 1, 1, 1, 2, 3, 3}.

# Ordenamiento por Conteo

3 / 3

- ▶ ¿Qué complejidad tiene el algoritmo?
  - ▶ Temporal
  - ▶ Espacial
- ▶ Este algoritmo funciona cuando las claves son números enteros
- ▶ ¿Y si fueran números reales o letras?
  - ▶ Mapear los valores a índices del arreglo en tiempo constante







# Tarea



- ▶ ¿Qué quisieran hacer como proyecto de obtención de grado para el cual esta materia pueda ayudar?
  - ▶ Desde el punto de vista de las técnicas que se verán o líneas de investigación de interés.
  - ▶ O desde el punto de vista de la aplicación o problema que se desea resolver.
  - ▶ Procesar muchas restricciones, o mucha información, información no estructurada (audio, video, imágenes, texto), ej. Clasificar imágenes, buscar en videos o audio, grafos de datos, etc.
  - ▶ Leer proyectos propuestos.