Algoritmos y estructuras de datos

Ordenamiento por montículo

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Posgrado en Optimización

26 de mayo de 2021

Cita

D.A.F. (Algorithmus)

Als kleiner Junge,
Ich hab's gewusst,
Mein Leben wird ganz wunderbar.
Ich lebe einfach radikal.
Nach Algorithmen meiner Wahl.
Ein Algorithmus ist ein Ball.
Da drin gefangen eine Zahl.
Befrei die Zahl, und spiel den Ball!

D.A.F. (Algoritmo)

Cuando era joven,
Ya lo sabía,
Mi vida sería maravillosa.
Vivo simplemente radical.
Por algoritmo mi elección.
Un algoritmo es una pelota.
Adentro está atrapado un número.
¡Libera al número y juega la pelota!

Problema abstracto

Dado un arreglo A con n elementos, se desea reacomodar sus elementos de modo que $A_0 \leq A_1 \leq \ldots \leq A_{n-1}$.

Problema concreto

Dado un arreglo `int a[MAX]` con `int n` elementos en las posiciones `a[0], ..., a[n-1]` se desea reacomodar sus elementos de modo que `a[0] <= a[1] <= ... <= a[n-1]`.

El problema de ordenamiento interno es uno de los más estudiados en la computación y para el que se conocen muchos algoritmos. Algunos son:

- Ordenamiento por cuenta (*Counting sort*).
- Ordenamiento de burbuja (*Bubble sort*).
- Ordenamiento por inserción (*Insertion sort*).
- Ordenamiento por mezcla (*Merge sort*).
- Ordenamiento por selección (*Selection sort*).
- Ordenamiento por partición (*Quicksort*).
- Ordenamiento por árbol (*Tree sort*).
- Ordenamiento por montículo (*Heapsort*).

Selección directa

Idea: Selecciona el elemento más grande

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	**9**
3	1	4	1	5	9	2	6	5	3	5	8	3	7	**9**	**9**
3	1	4	1	5	7	2	6	5	3	5	8	3	**9**	**9**	**9**
3	1	4	1	5	7	2	6	5	3	5	3	**8**	**9**	**9**	**9**
3	1	4	1	5	3	2	6	5	3	5	**7**	**8**	**9**	**9**	**9**
3	1	4	1	5	3	2	5	5	3	**6**	**7**	**8**	**9**	**9**	**9**
3	1	4	1	5	3	2	5	3	**5**	**6**	**7**	**8**	**9**	**9**	**9**
3	1	4	1	5	3	2	3	**5**	**5**	**6**	**7**	**8**	**9**	**9**	**9**
3	1	4	1	3	3	2	**5**	**5**	**5**	**6**	**7**	**8**	**9**	**9**	**9**
3	1	2	1	3	3	**4**	**5**	**5**	**5**	**6**	**7**	**8**	**9**	**9**	**9**

Selección directa

Tenemos un arreglo desordenado `a[0]`, `a[1]`, ..., `a[j]` en el que queremos encontrar el máximo `a[i]` e intercambiarlo por `a[j]`. Esto lo hicimos con la llamada a `maximo(j, a)` en `j` pasos. En total se hicieron $1 + 2 + \cdots + (n-1) = \frac{1}{2}n(n-1)$ pasos.

Pregunta

¿Habrá alguna forma de hacer esto más rápido?

Cola de prioridad

Definición

Una cola de prioridad es un tipo de datos abstracto que tiene las siguientes operaciones:

1. Crea una cola de prioridad vacía.
2. Inserta un elemento.
3. Elimina el elemento de valor máximo.

Implementación

Hay muchas formas de implementar colas de prioridad. Implementaremos una forma llamada montículo máximo que se representa en un arreglo y donde las operaciones de inserción y eliminación son razonablemente rápidas.

Montículos

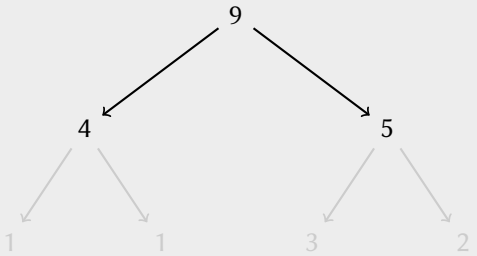
Montículo máximo

Un montículo máximo es un árbol binario que tiene las propiedades:

Orden El elemento a_s del nodo s es mayor o igual que sus sucesores.

Balance Todos los niveles están llenos (el último lleno de izquierda a derecha).

Ejemplo



Montículos

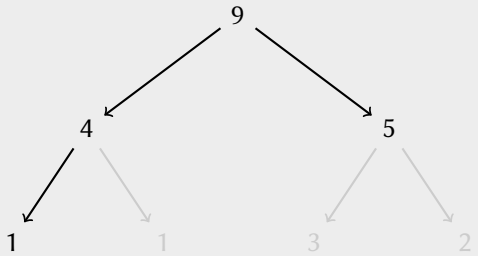
Montículo máximo

Un montículo máximo es un árbol binario que tiene las propiedades:

Orden El elemento a_s del nodo s es mayor o igual que sus sucesores.

Balance Todos los niveles están llenos (el último lleno de izquierda a derecha).

Ejemplo



Montículos

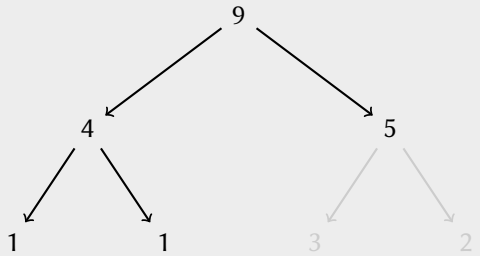
Montículo máximo

Un montículo máximo es un árbol binario que tiene las propiedades:

Orden El elemento a_s del nodo s es mayor o igual que sus sucesores.

Balance Todos los niveles están llenos (el último lleno de izquierda a derecha).

Ejemplo



Montículos

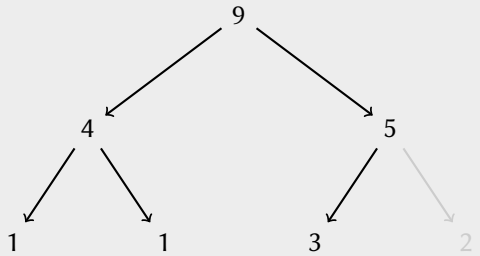
Montículo máximo

Un montículo máximo es un árbol binario que tiene las propiedades:

Orden El elemento a_s del nodo s es mayor o igual que sus sucesores.

Balance Todos los niveles están llenos (el último lleno de izquierda a derecha).

Ejemplo



Montículos

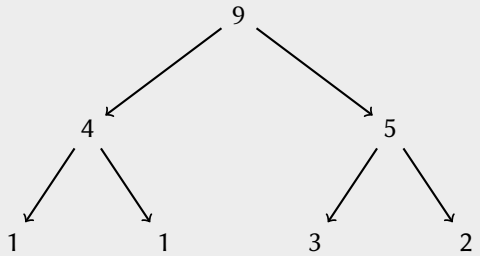
Montículo máximo

Un montículo máximo es un árbol binario que tiene las propiedades:

Orden El elemento a_s del nodo s es mayor o igual que sus sucesores.

Balance Todos los niveles están llenos (el último lleno de izquierda a derecha).

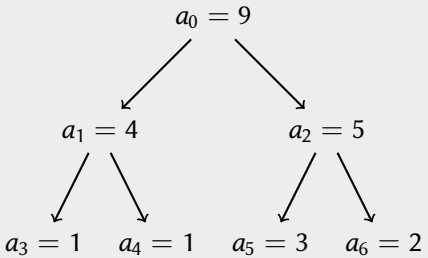
Ejemplo



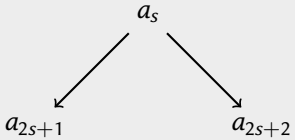
Montículos

Representación implícita en un arreglo

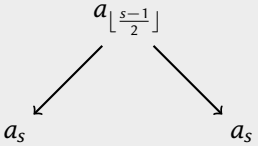
Los elementos del arreglo a representan los nodos del árbol, nivel por nivel, a partir de la raíz a_0, sus sucesores a_1, a_2, etc.



Sucesores del nodo s:



Precursor del nodo s:



Montículos

Ejemplo de creación de un montículo

3 1 4 1 5 9 2 6 5 3 5 8 9 7 9

3

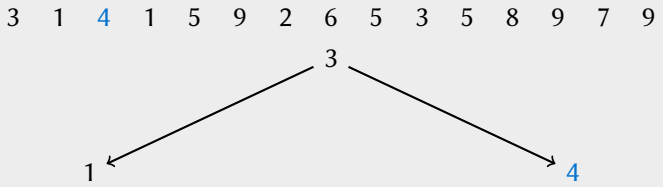
Montículos

Ejemplo de creación de un montículo

3 1 4 1 5 9 2 6 5 3 5 8 9 7 9

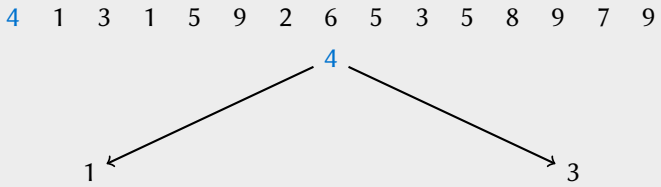
Montículos

Ejemplo de creación de un montículo



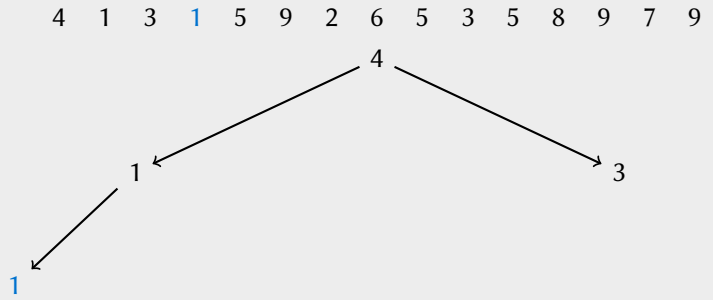
Montículos

Ejemplo de creación de un montículo



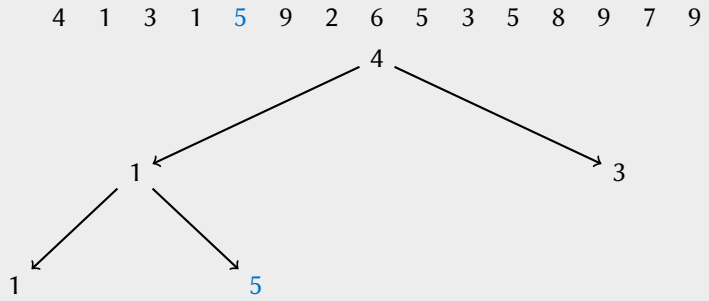
Montículos

Ejemplo de creación de un montículo



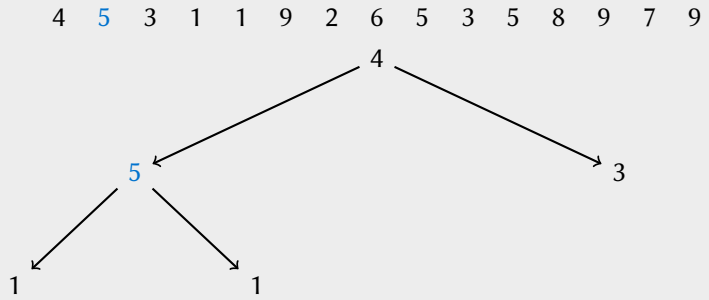
Montículos

Ejemplo de creación de un montículo



Montículos

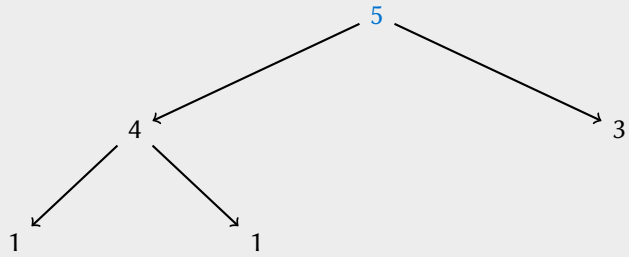
Ejemplo de creación de un montículo



Montículos

Ejemplo de creación de un montículo

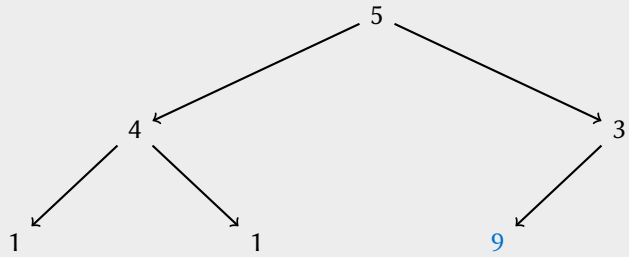
5 4 3 1 1 9 2 6 5 3 5 8 9 7 9



Montículos

Ejemplo de creación de un montículo

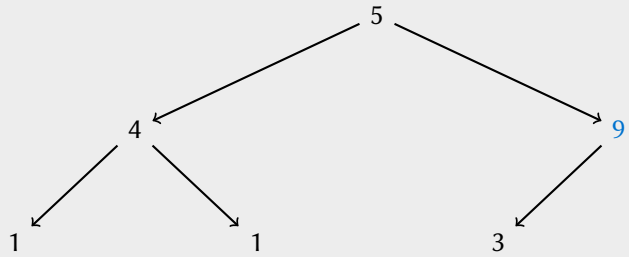
5 4 3 1 1 9 2 6 5 3 5 8 9 7 9



Montículos

Ejemplo de creación de un montículo

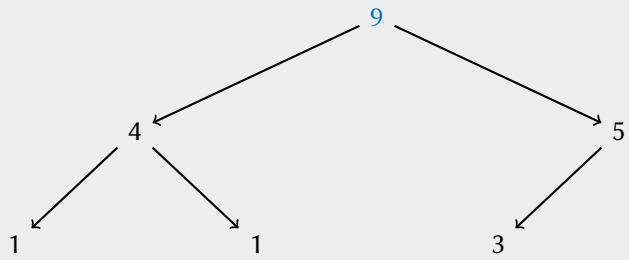
5 4 9 1 1 3 2 6 5 3 5 8 9 7 9



Montículos

Ejemplo de creación de un montículo

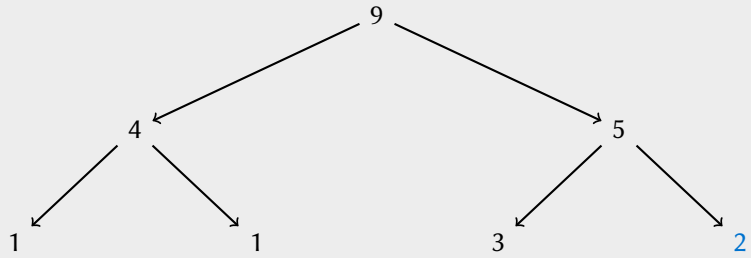
9 4 5 1 1 3 2 6 5 3 5 8 9 7 9



Montículos

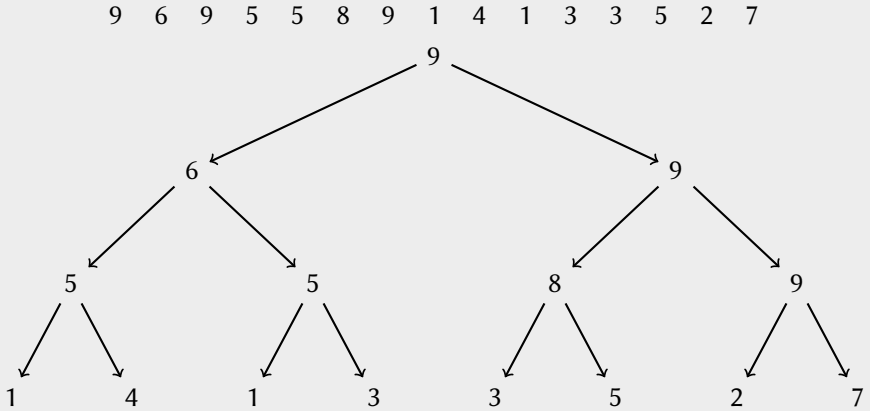
Ejemplo de creación de un montículo

9 4 5 1 1 3 2 6 5 3 5 8 9 7 9



Montículos

Ejemplo de creación de un montículo



Suponga que se tiene un montículo en el arreglo `a[0], ..., a[j-1]` y se quiere insertar el dato `a[j]` en el montículo.

```
void insertaMonMax(int j, int a[]) {
  while (j > 0) {
    int p = (j-1)/2;          // precursor de j
    if (a[p] < a[j]) {        // mal puesto
      intercambio(a, p, j); // cambialos
      j = p;                  // y avanza
    } else break;
  }
}
```

Insertamos todos los elementos del arreglo `a` uno por uno en el montículo.

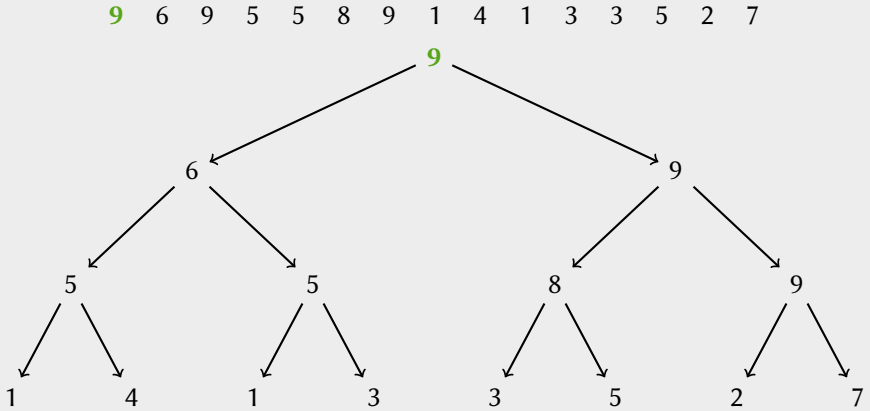
```
void construyeMonMax(int n, int a[]) {
  for (int j = 1; j < n; j++)
    insertaMonMax(j, a);
}
```

Tiempo de ejecución

Cada inserción tarda $\leq \log_2 n$ pasos. En total se hacen $\leq n \log_2 n$ pasos.

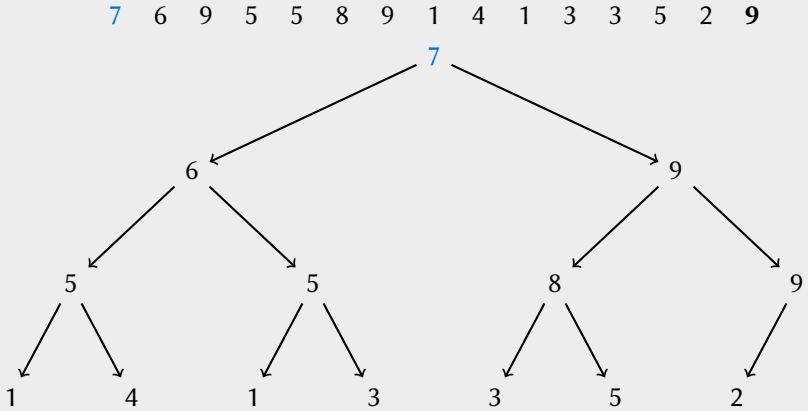
Montículos

Ejemplo de destrucción de un montículo



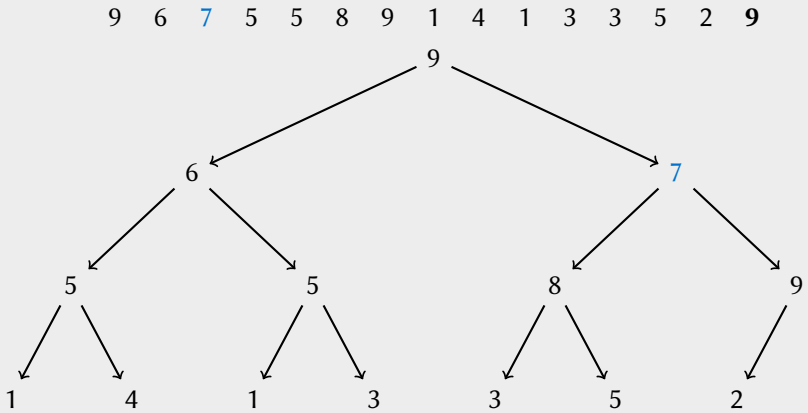
Montículos

Ejemplo de destrucción de un montículo



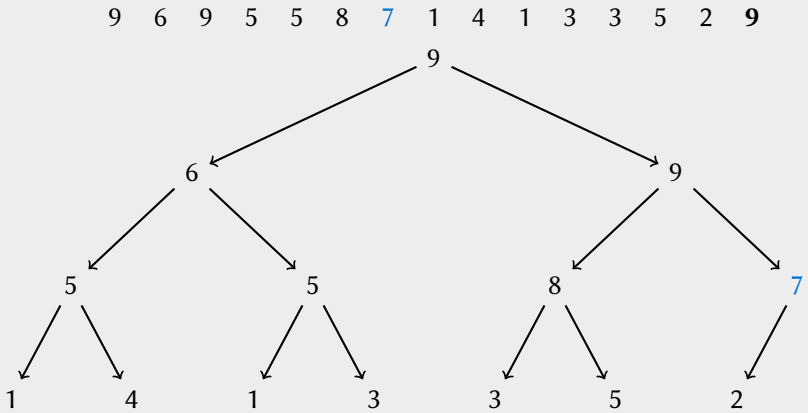
Montículos

Ejemplo de destrucción de un montículo



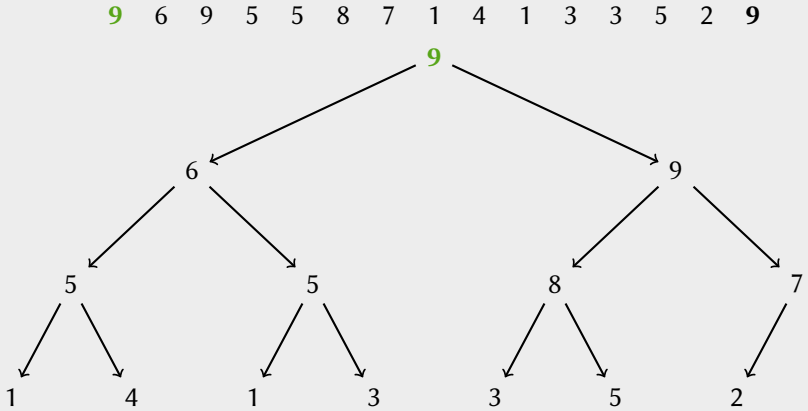
Montículos

Ejemplo de destrucción de un montículo



Montículos

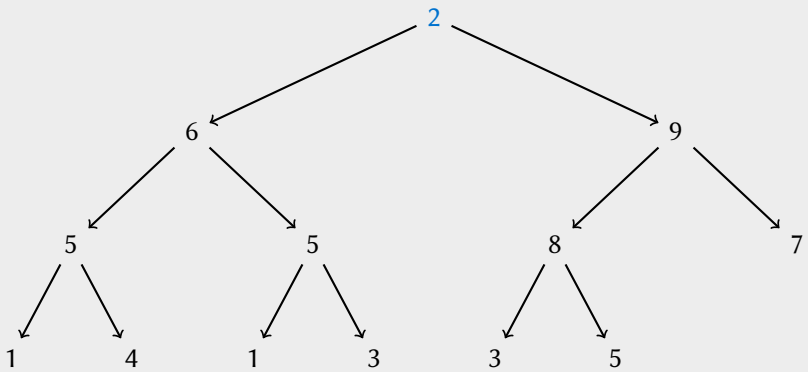
Ejemplo de destrucción de un montículo



Montículos

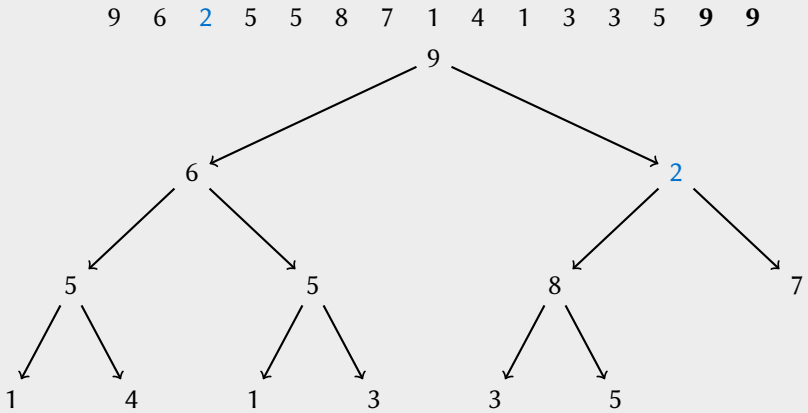
Ejemplo de destrucción de un montículo

2 6 9 5 5 8 7 1 4 1 3 3 5 **9** **9**



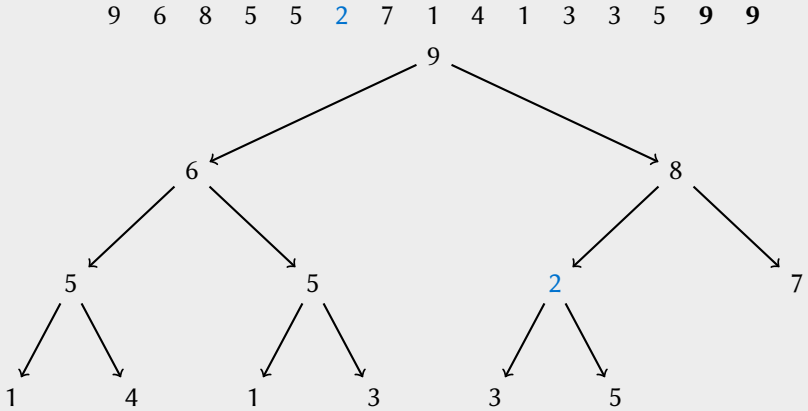
Montículos

Ejemplo de destrucción de un montículo



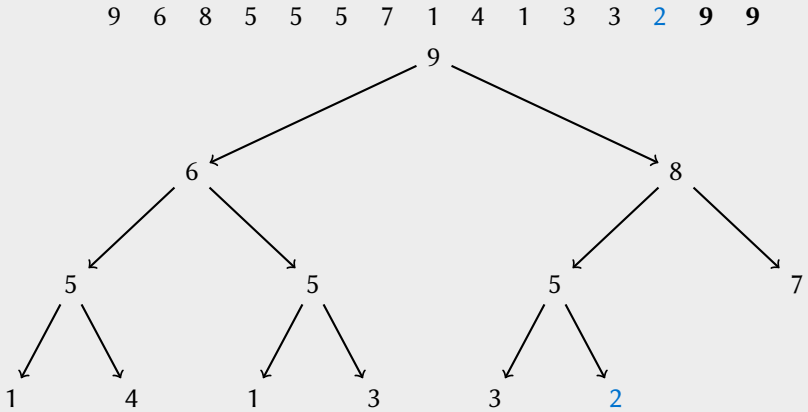
Montículos

Ejemplo de destrucción de un montículo



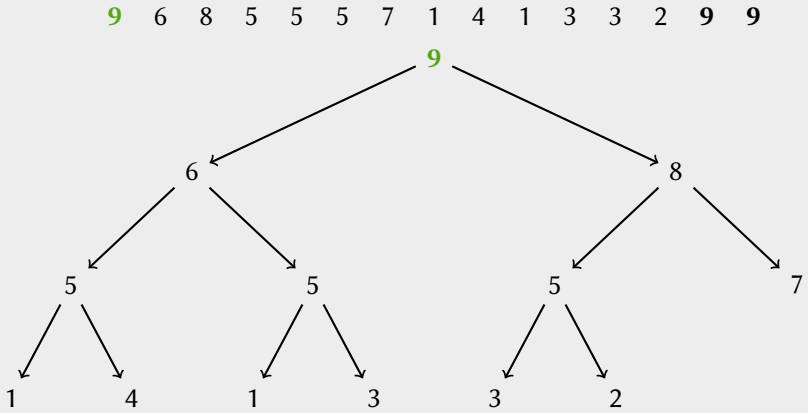
Montículos

Ejemplo de destrucción de un montículo



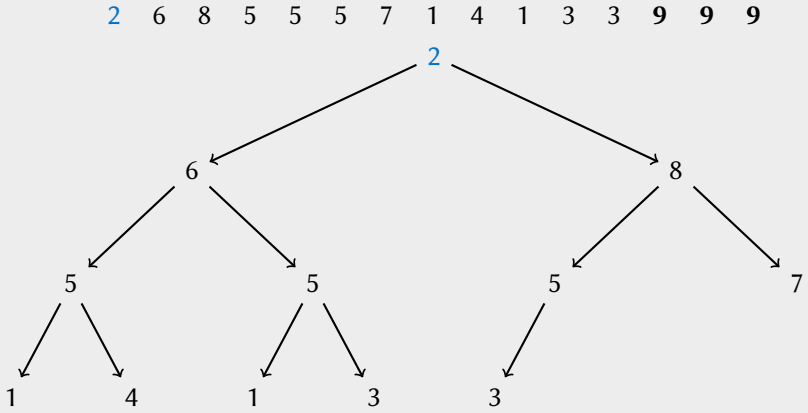
Montículos

Ejemplo de destrucción de un montículo



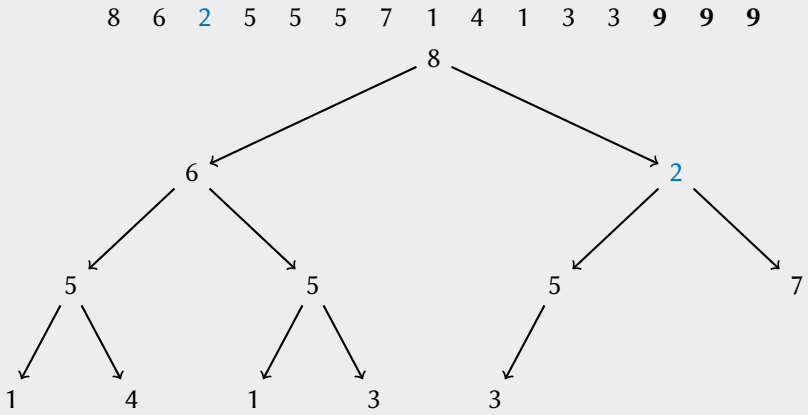
Montículos

Ejemplo de destrucción de un montículo



Montículos

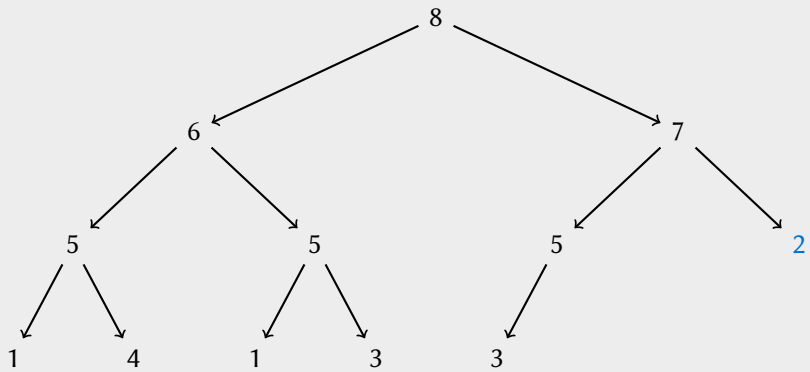
Ejemplo de destrucción de un montículo



Montículos

Ejemplo de destrucción de un montículo

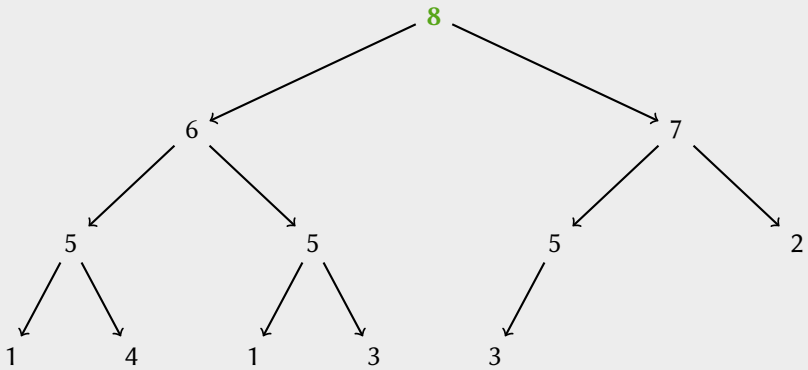
8 6 7 5 5 5 2 1 4 1 3 3 **9** **9** **9**



Montículos

Ejemplo de destrucción de un montículo

8 6 7 5 5 5 2 1 4 1 3 3 **9** **9** **9**



Montículos

Ejemplo de destrucción de un montículo

1 1 2 3 3 4 5 5 5 6 7 8 9 9 9

1

Implementación de la eliminación

Se tiene un montículo en `a[0], ..., a[j]` y se quiere eliminar `a[0]`.

```
void eliminaMonMax(int j, int a[]) {
  int p = 0, s;
  intercambio(a, 0, j);              // el mayor
  while (2*p+1 < j) {                // con sucesor
    s = 2*p+1;                       // izquierdo
    if (s+1 < j && a[s] < a[s+1])    // y derecho
      s = s+1;                       // cambia
    if (a[p] < a[s]) {               // mal puesto
      intercambio(a, p, s);          // cambialos
      p = s;                         // y avanza
    } else break;
  }
}
```

Eliminamos todos los elementos del montículo uno por uno.

```
void destruyeMonMax(int n, int a[]) {
  for (int j = n-1; j > 0; j--)
    eliminaMonMax(j, a);
}
```

Tiempo de ejecución

Cada eliminación tarda $\leq \log_2 n$ pasos. En total se hacen $\leq n \log_2 n$ pasos.

Ordenamiento por montículo

Consiste simplemente en construir un montículo y luego destruirlo:

```
void monticulo(int n, int a[]) {
  construyeMonMax(n, a);
  destruyeMonMax(n, a);
}
```

Tiempo de ejecución

En total se hacen $\leq n \log_2 n$ pasos.

Ordenamiento por montículo

Comparación con otros algoritmos de ordenamiento

Propiedad	Burbuja	Inserción	Selección	Mezcla	Quicksort	Montículo
Memoria	n	n	n	$2n$	n	n
Peor tiempo	n^2	n^2	n^2	$n \log_2 n$	n^2	$n \log_2 n$
Mejor tiempo	n	n	n^2	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$
Promedio	n^2	n^2	n^2	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$

Tiempo promedio para ordenar n datos si cada comparación tarda 1 ns.

n	Burbuja	Inserción	Selección	Mezcla	Quicksort	Montículo
10^3	0.5 ms	0.5 ms	0.5 ms	0.01 ms	0.01 ms	0.01 ms
10^4	50 ms	50 ms	50 ms	0.1 ms	0.1 ms	0.1 ms
10^5	5 s	5 s	5 s	1 ms	1 ms	1 ms
10^6	500 s	500 s	500 s	20 ms	20 ms	20 ms

1. Si se quisiera ordenar de mayor a menor se usaría un montículo mínimo con la propiedad de orden invertida: el precursor debe ser menor o igual que sus sucesores. Reescribe todas las funciones vistas de esta manera. Pista: Sólo se deben cambiar las líneas que tienen el comentario `// mal puesto`.
2. Aunque no es la convención de C, un montículo también se suele representar con la raíz en `a[1]`. En este caso, los sucesores de `a[p]` son `a[2*p]` y `a[2*p+1]`, mientras que el precursor de `a[s]` está en `a[s/2]`. Reescribe todas las funciones vistas de esta manera.