Algoritmos y estructuras de datos

Ordenamiento por montículo

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

27 de mayo de 2015

Ordenamiento interno

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]`.

Ordenamiento interno

Un problema, mil soluciones

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

- Burbuja
- Inserción directa.
- Selección directa.
- *Quicksort*.
- Ordenamiento por mezcla.
- Ordenamiento por montículo.

Comparaciones

Todos estos algoritmos comparan los datos en el arreglo.
Existen algoritmos que no comparan los datos en el arreglo.

Selección directa

Ejemplo

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

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.

Selección directa

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 máximo elemento.

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 máximo elemento.

Implementación

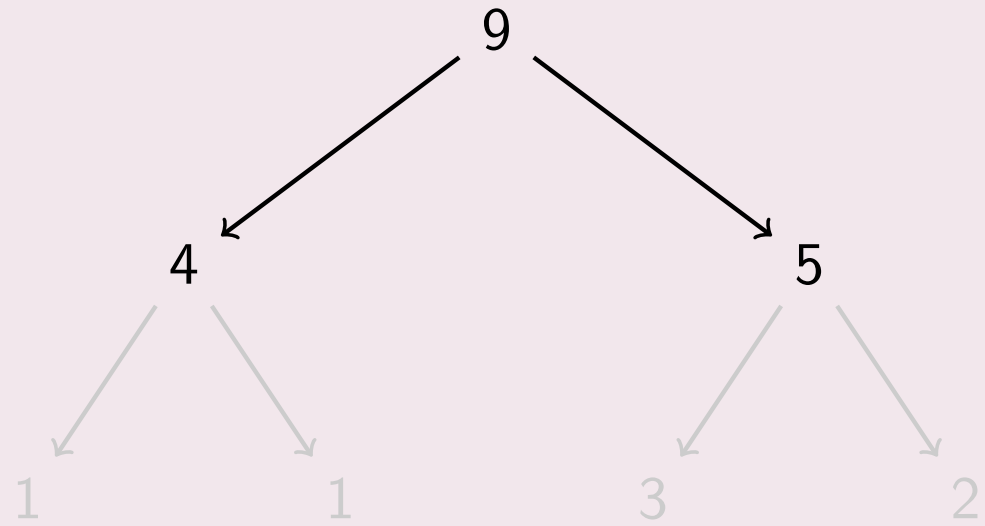
Hay muchas formas de implementar colas de prioridad. Implementaremos una llamada montículo que se representa en un arreglo y donde las operaciones de inserción y eliminación son razonablemente rápidas.

Montículo

Montículo

Podemos representar gráficamente un montículo como un árbol binario que tiene las propiedades de orden y balance.

Ejemplo

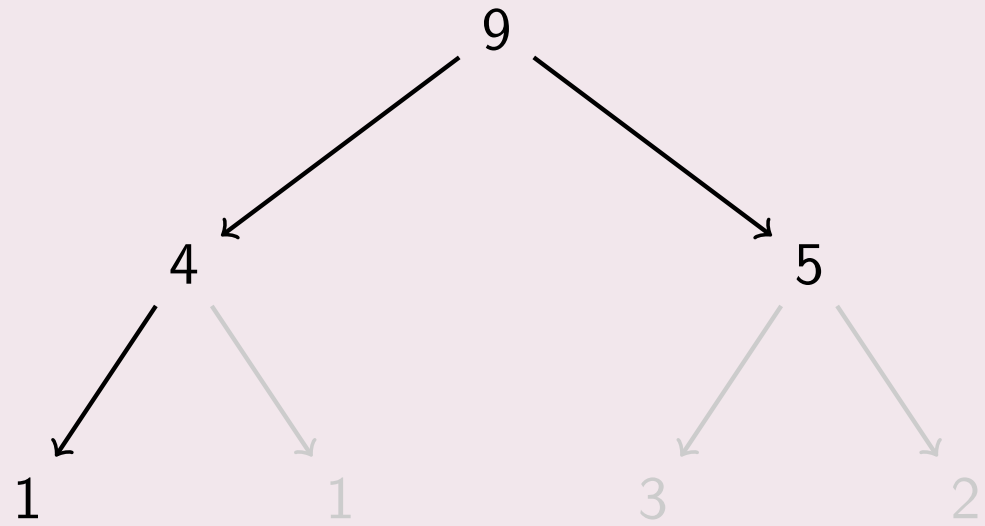


Montículo

Montículo

Podemos representar gráficamente un montículo como un árbol binario que tiene las propiedades de orden y balance.

Ejemplo

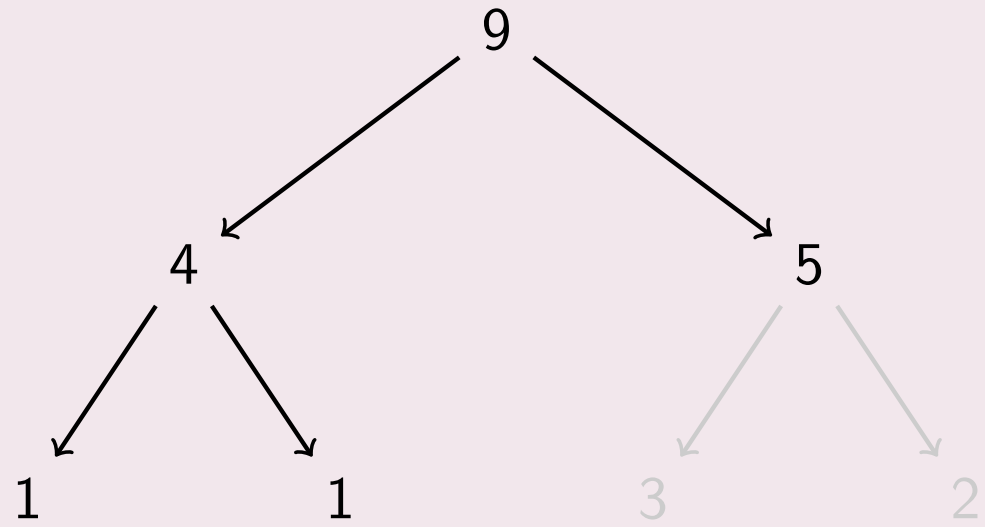


Montículo

Montículo

Podemos representar gráficamente un montículo como un árbol binario que tiene las propiedades de orden y balance.

Ejemplo

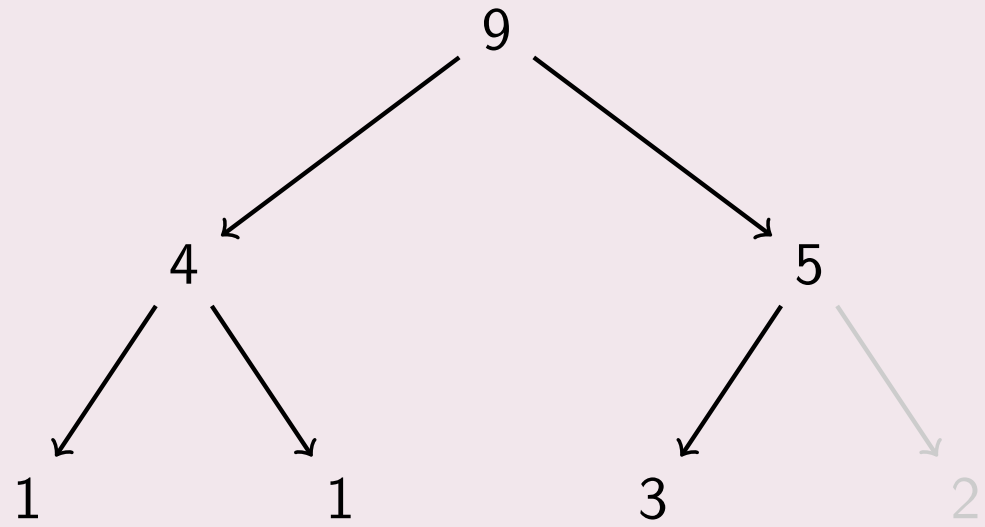


Montículo

Montículo

Podemos representar gráficamente un montículo como un árbol binario que tiene las propiedades de orden y balance.

Ejemplo

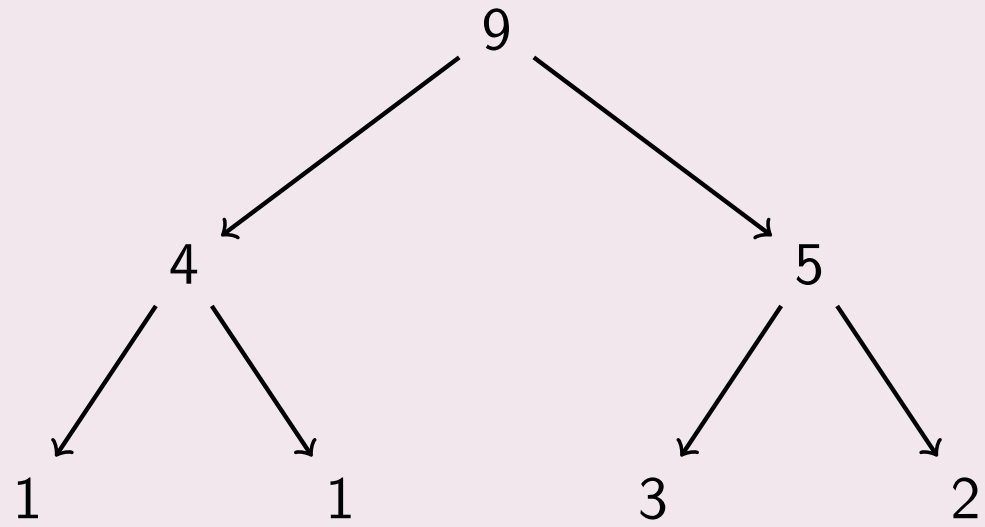


Montículo

Montículo

Podemos representar gráficamente un montículo como un árbol binario que tiene las propiedades de orden y balance.

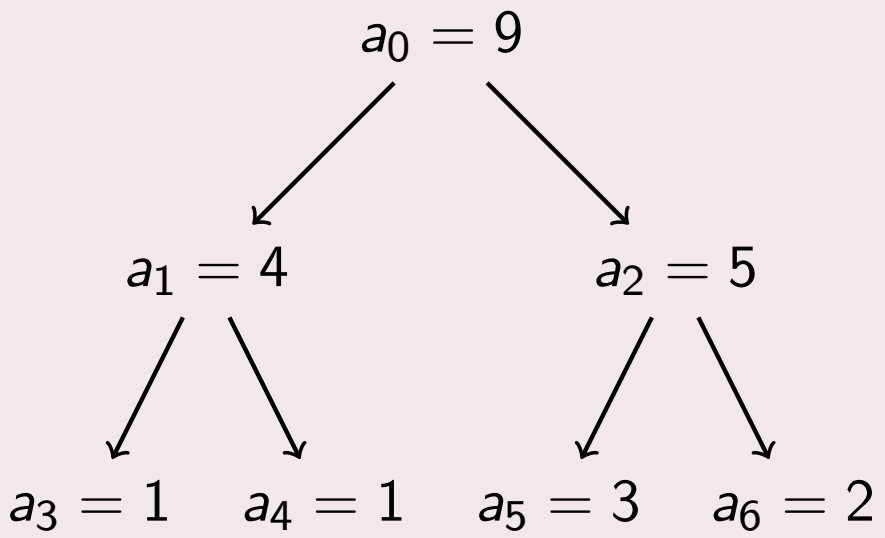
Ejemplo



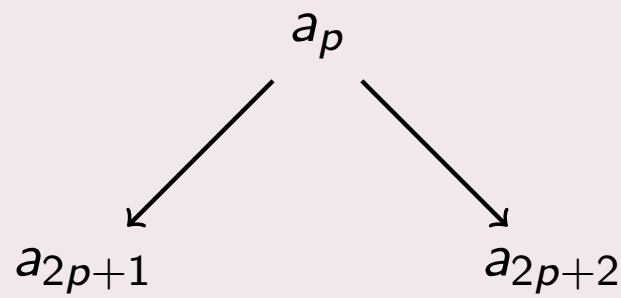
Montículo

Montículo en un arreglo

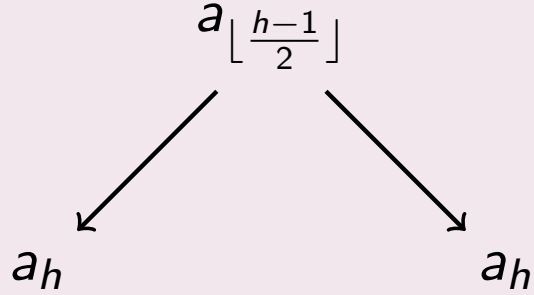
Los elementos del arreglo a representan los nodos del árbol, nivel por nivel, a partir de la raíz a_0, sus hijos a_1, a_2, etc.



Hijos del padre



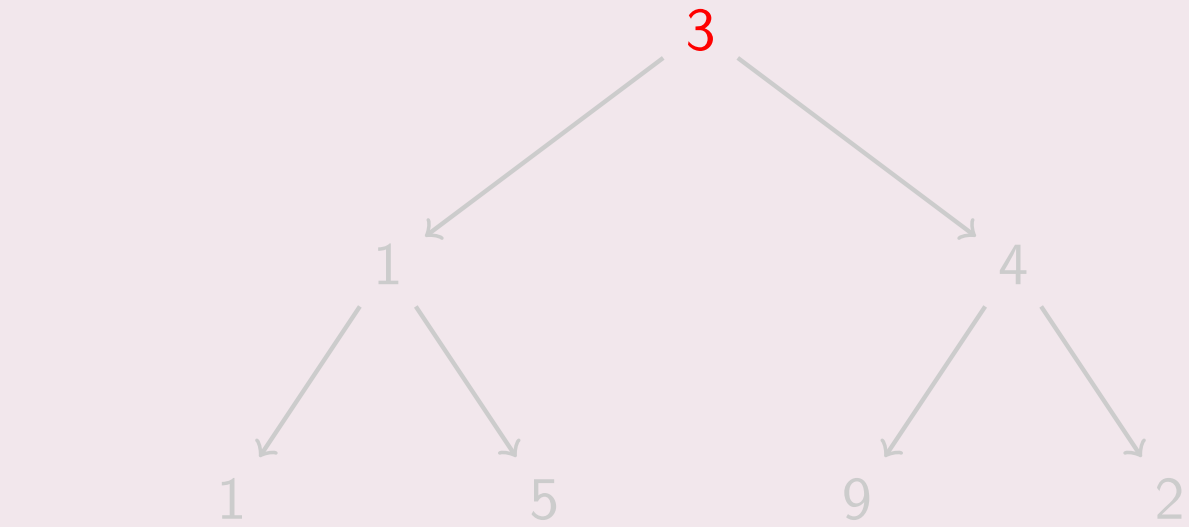
Padre de los hijos



Montículos

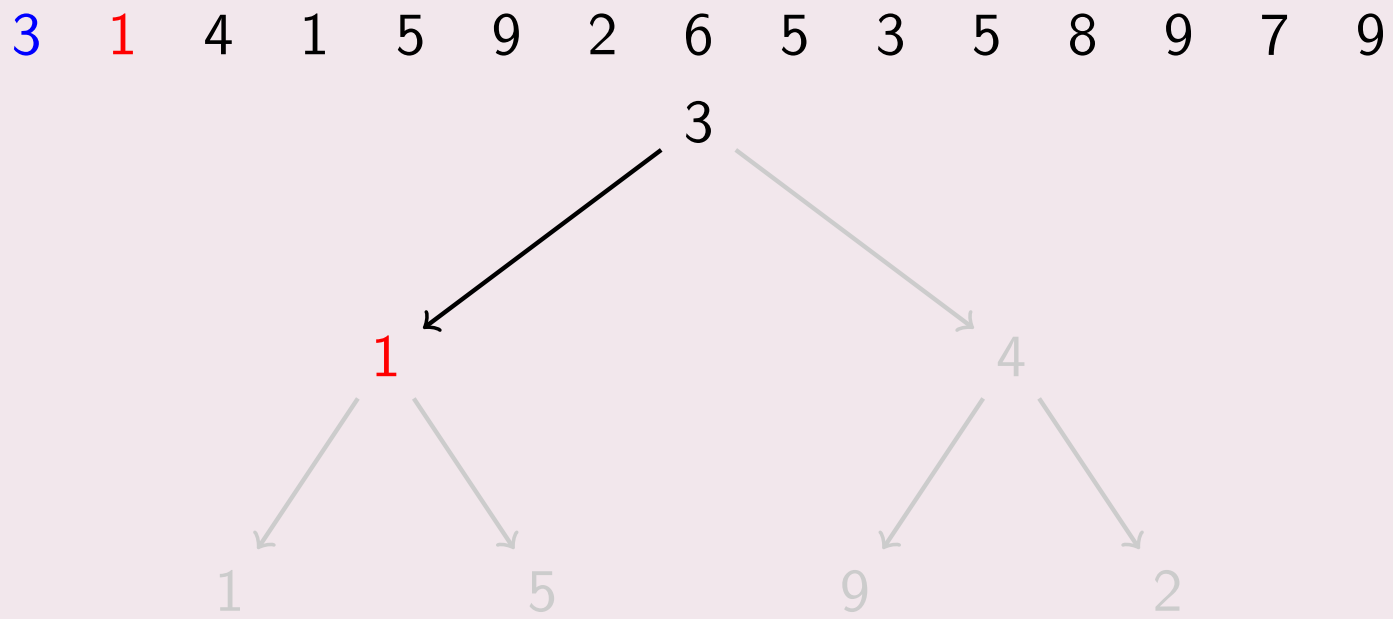
Inserción

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



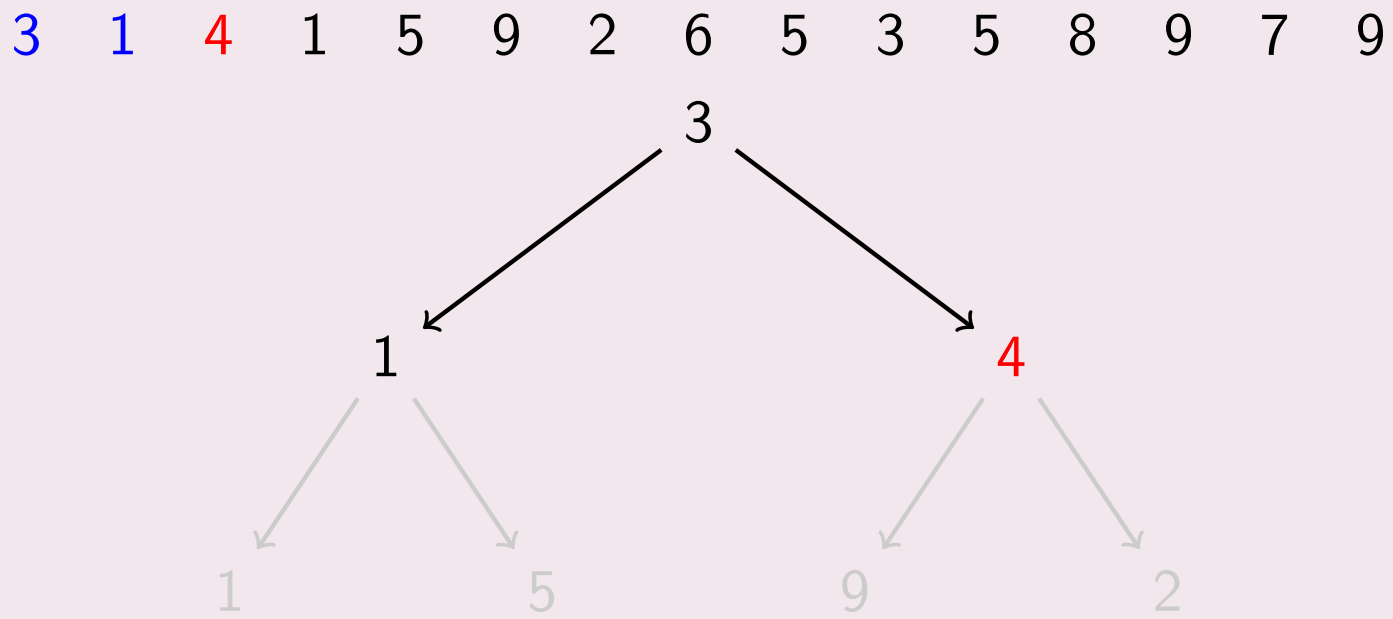
Montículos

Inserción



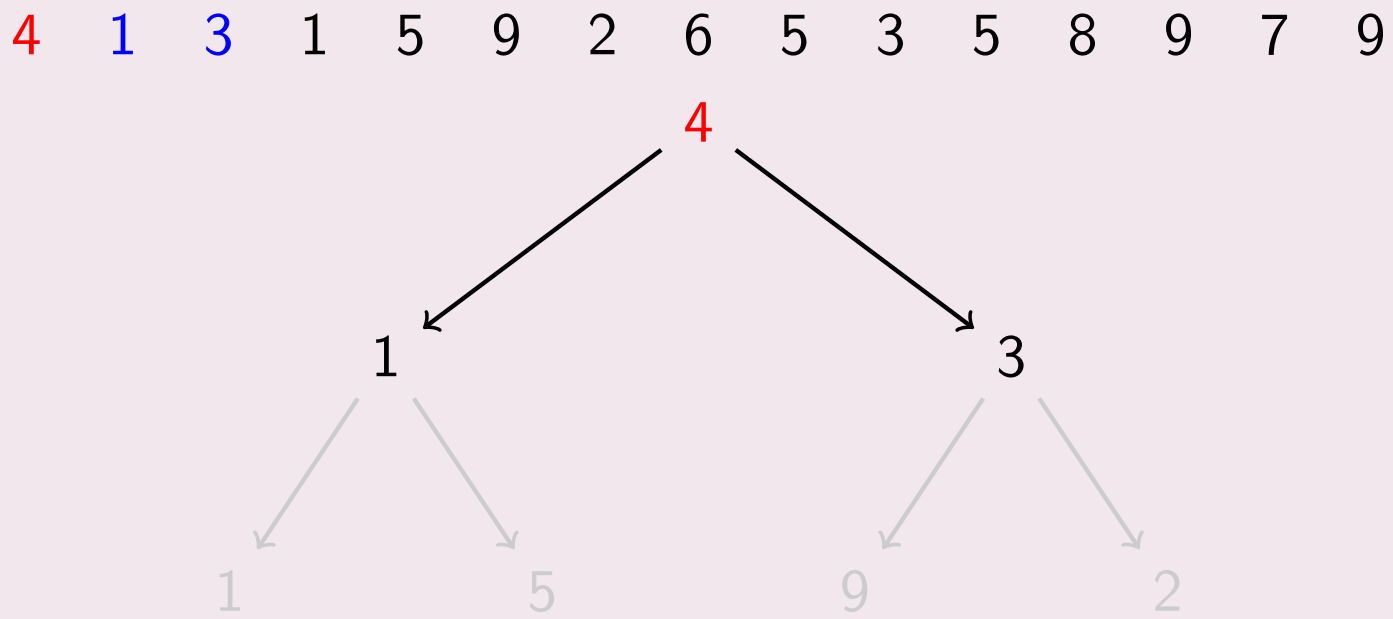
Montículos

Inserción

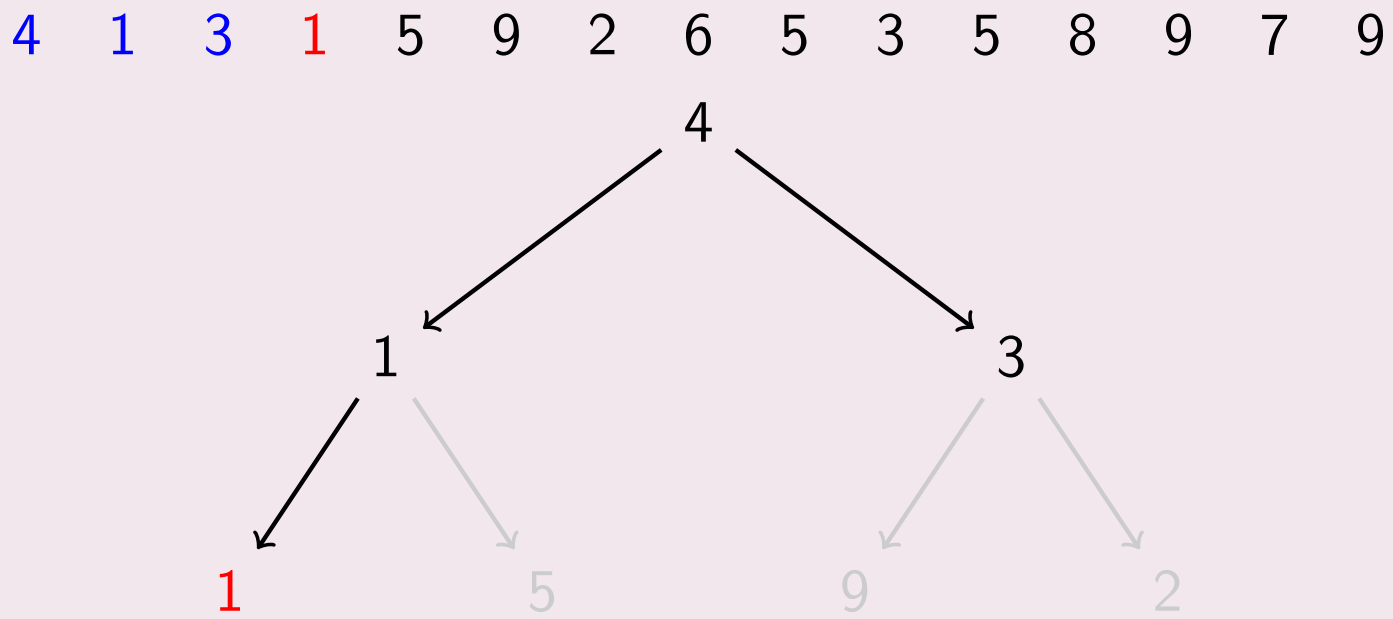


Montículos

Inserción

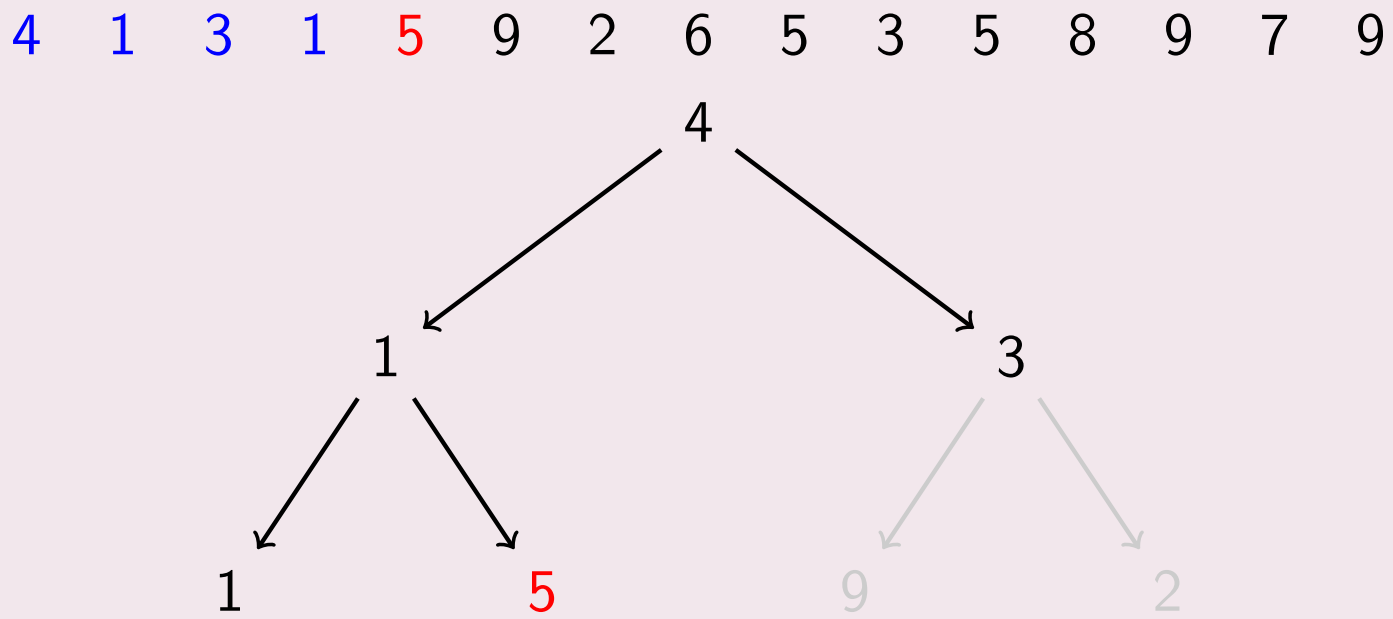


Inserción



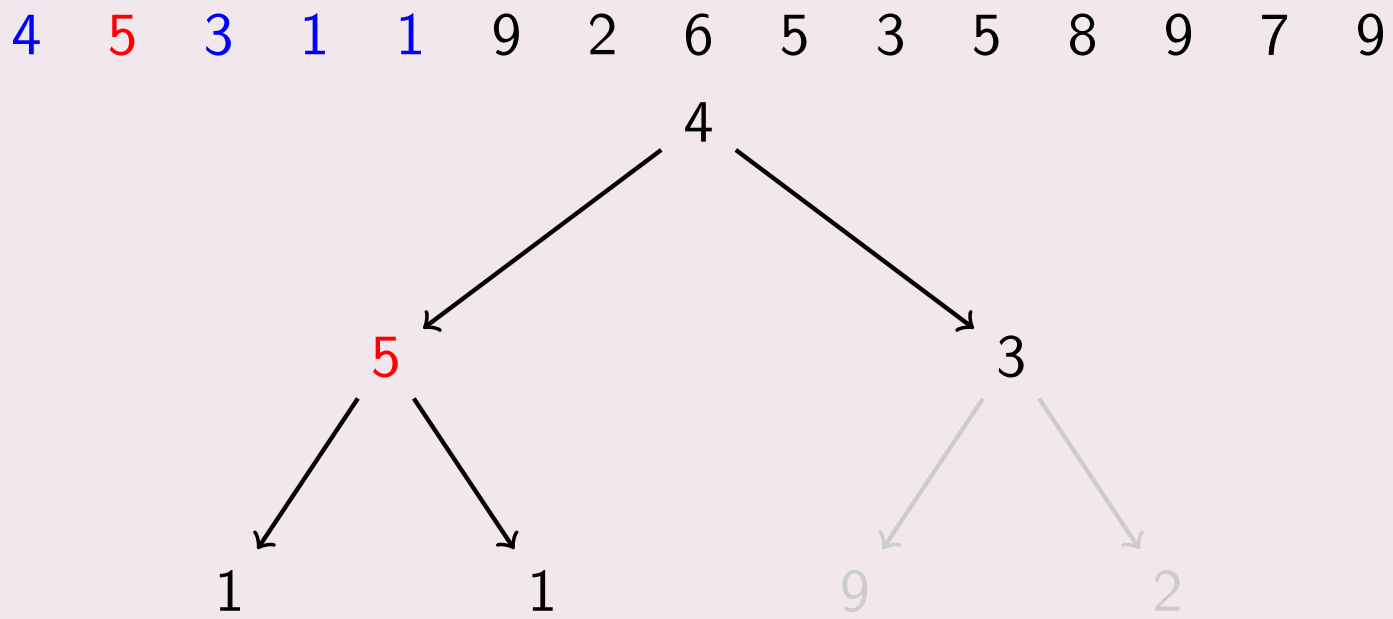
Montículos

Inserción



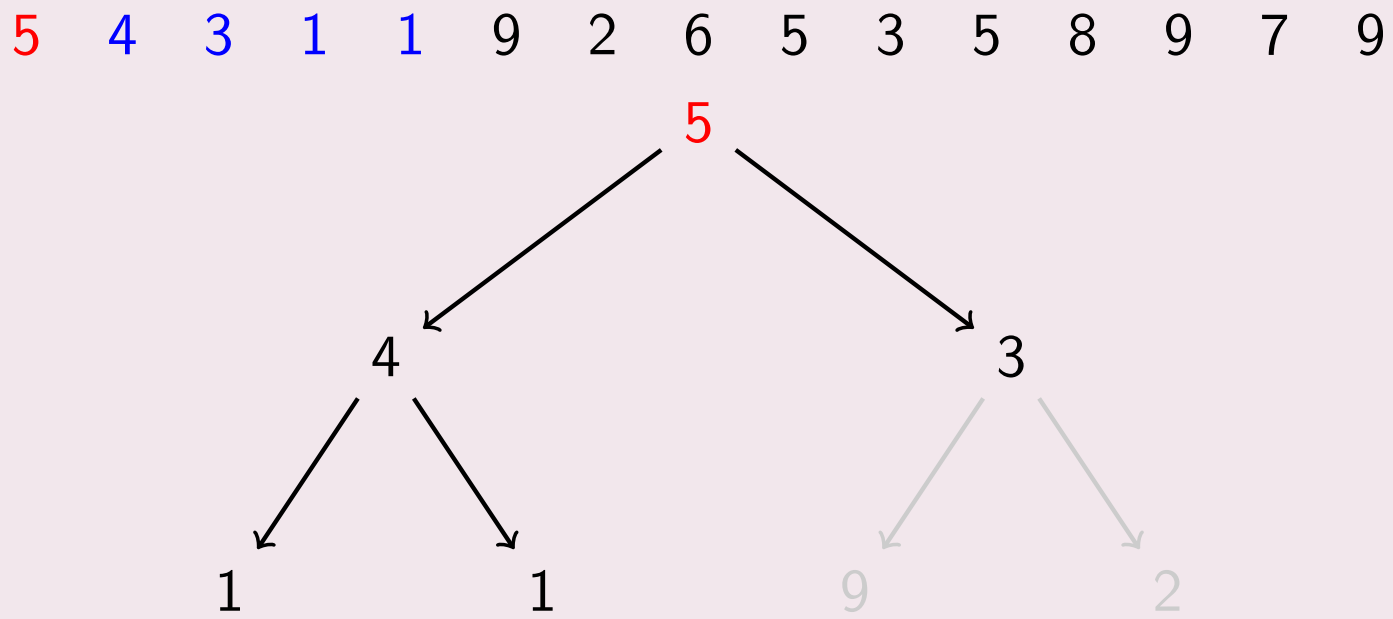
Montículos

Inserción



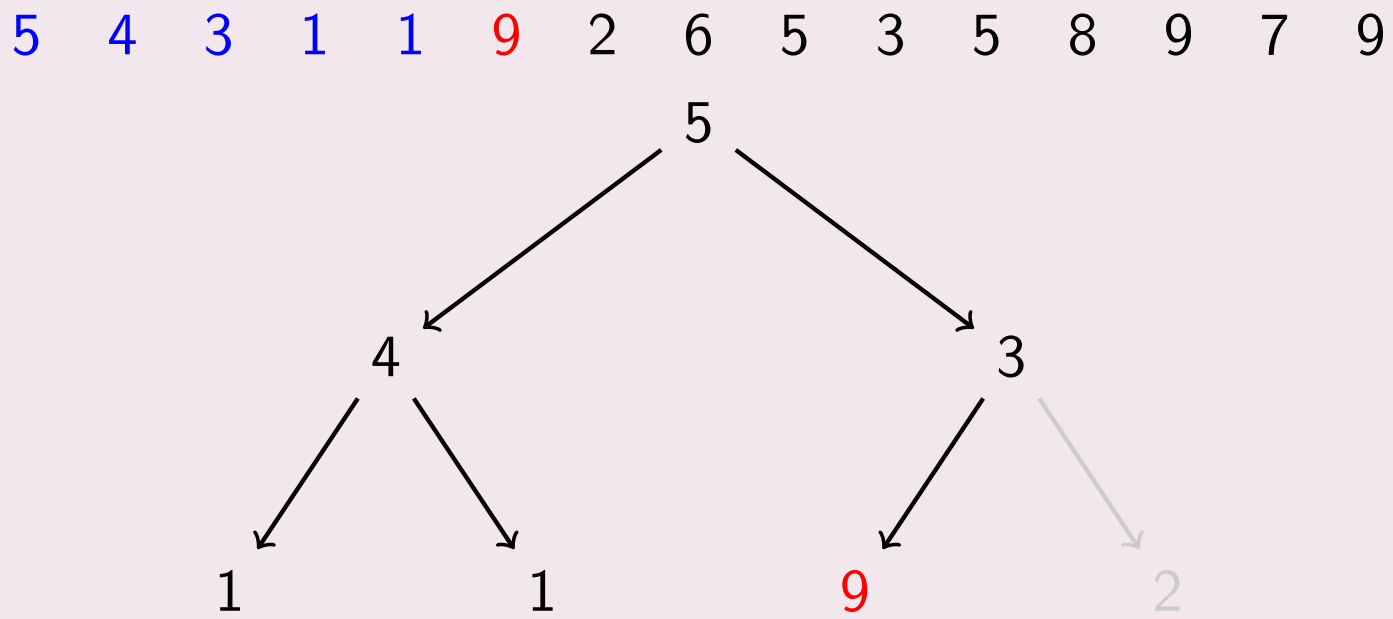
Montículos

Inserción



Montículos

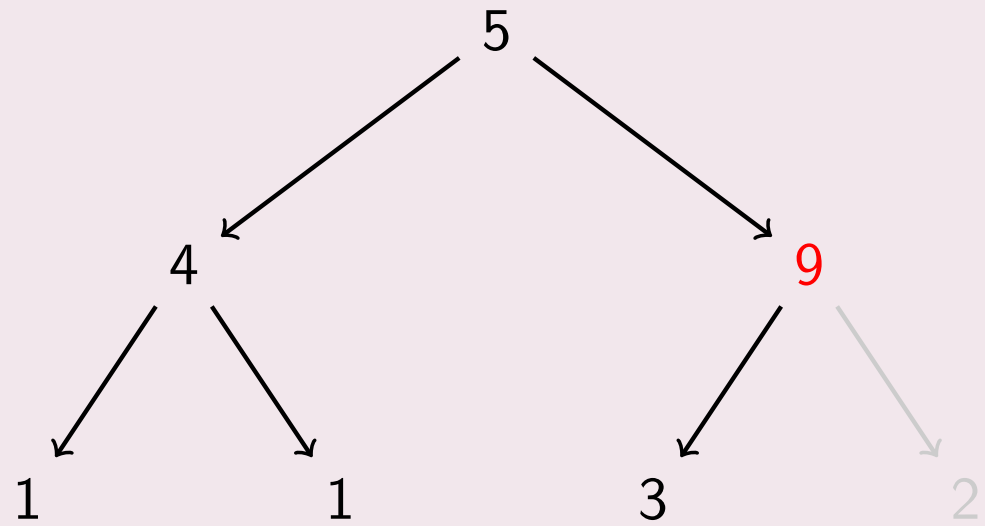
Inserción



Montículos

Inserción

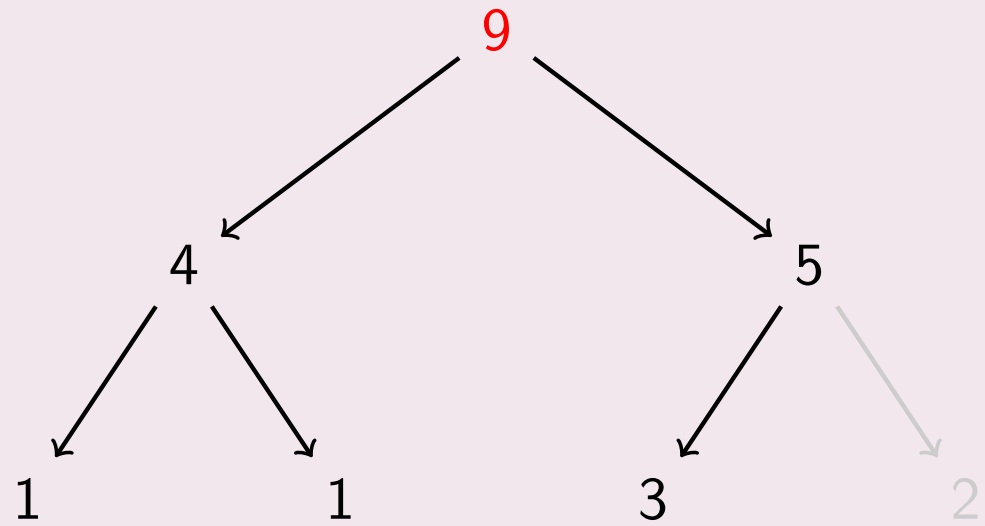
5 4 9 1 1 3 2 6 5 3 5 8 9 7 9



Montículos

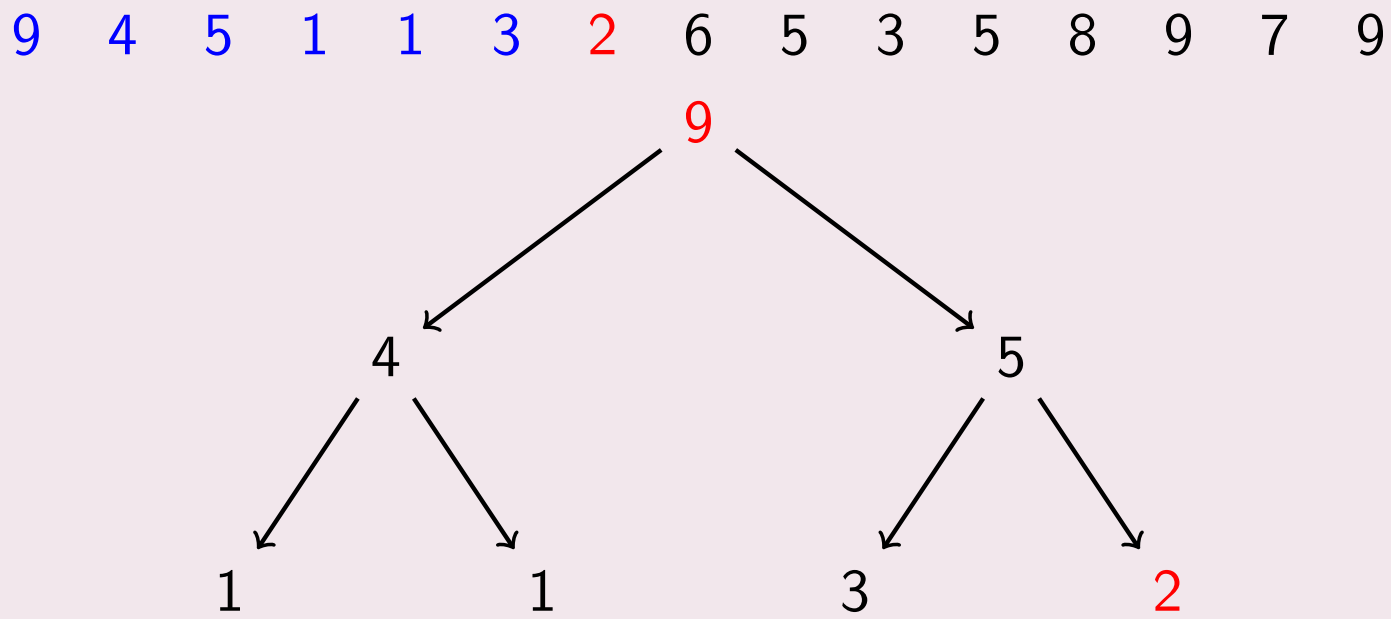
Inserción

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



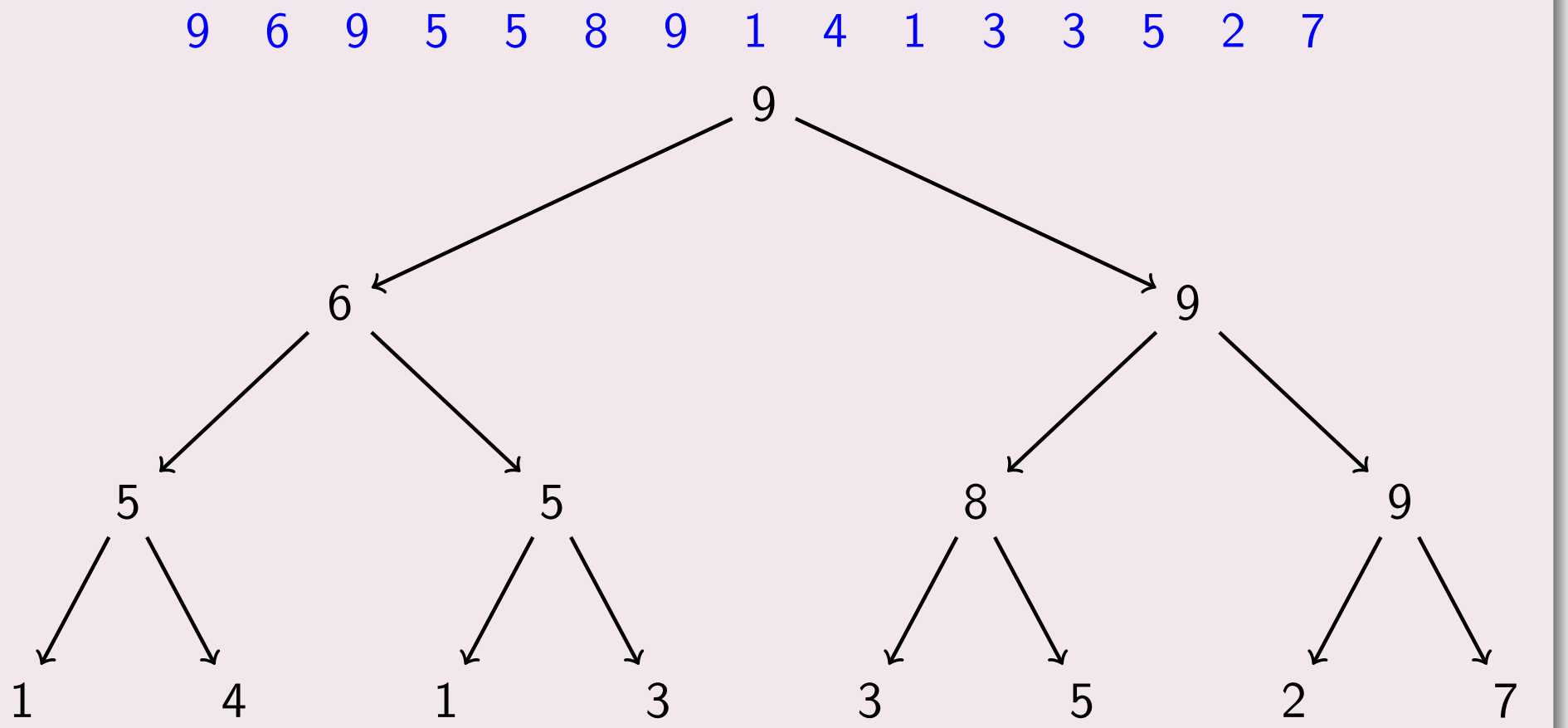
Montículos

Inserción



Montículos

El montículo completo



Montículos

Inserción

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 inserta(int j, int a[])
{
  int p;

  while (j > 0) {
    p = (j-1)/2;                    /* padre de j */
    if (a[p] < a[j]) {              /* mal puesto */
      intercambia(&a[p], &a[j]);    /* cambialos  */
      j = p;                        /* y avanza   */
    } else break;
  }
}
```

Construcción de un montículo

```
void construye(int n, int a[])
{
  int j;

  for (j = 1; j < n; j++)
    inserta(j, a);
}
```

Montículos

Construcción de un montículo

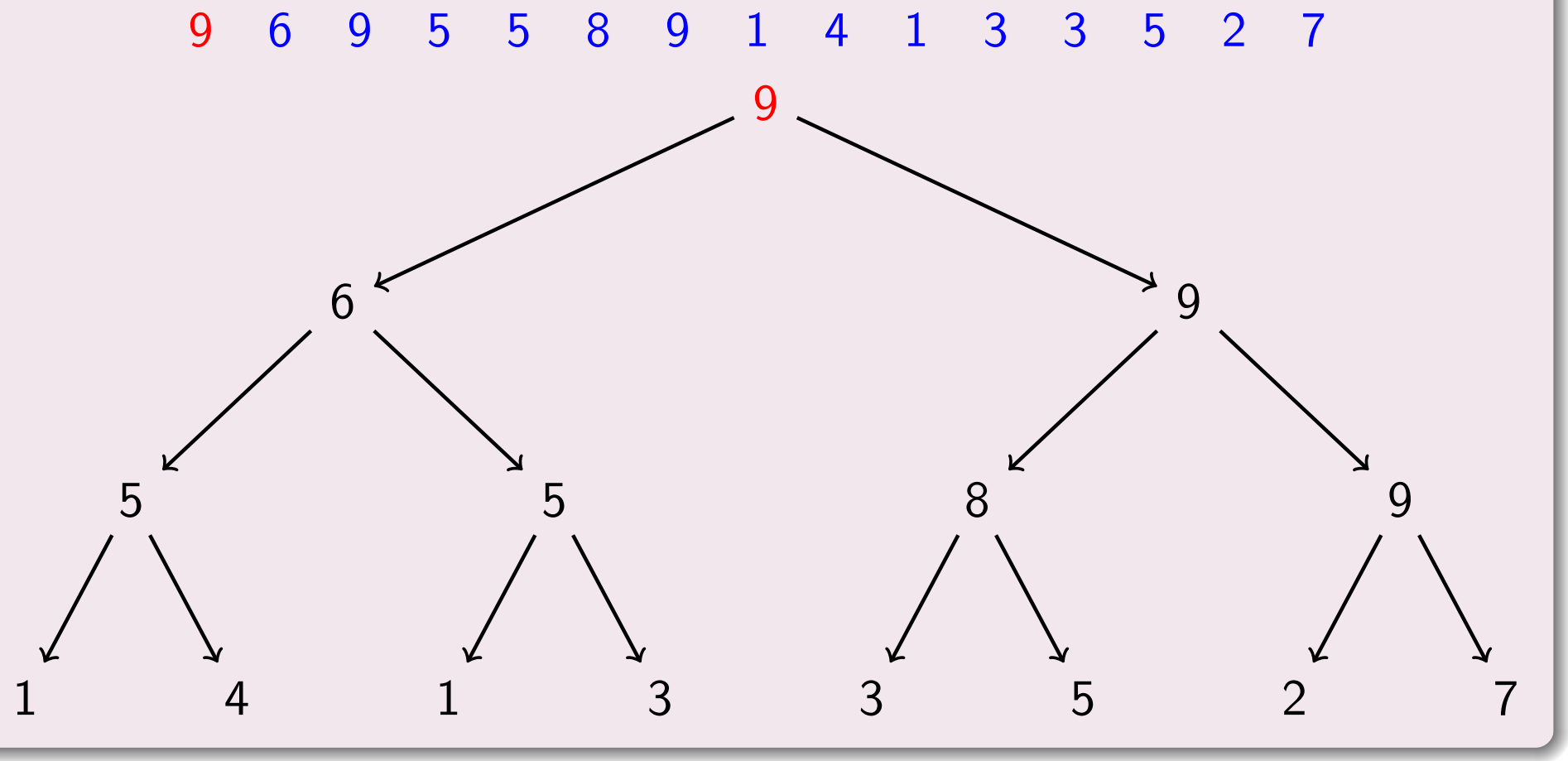
```
void construye(int n, int a[])
{
  int j;

  for (j = 1; j < n; j++)
    inserta(j, a);
}
```

Tiempo de ejecución

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

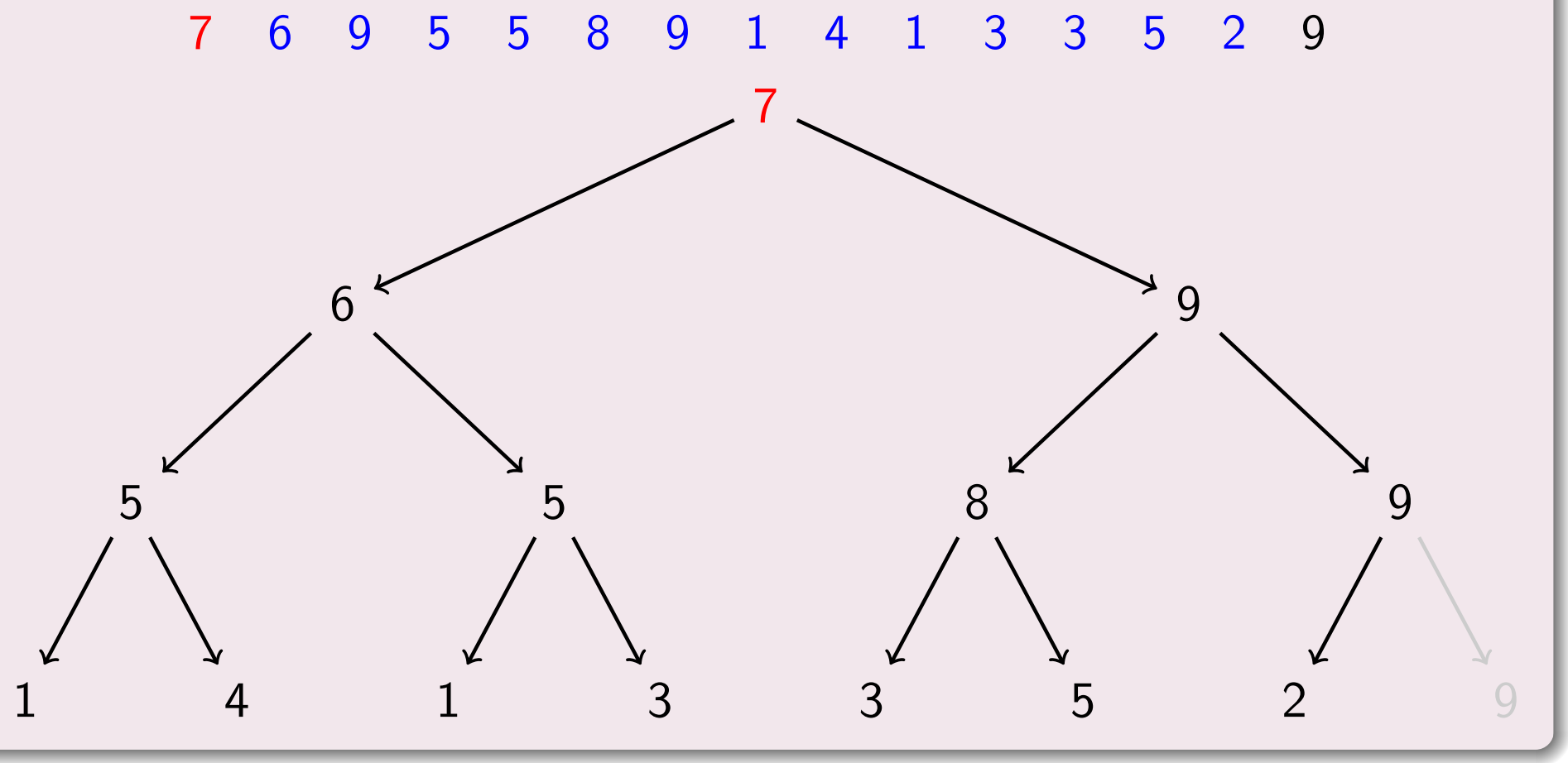
Montículos

Eliminación



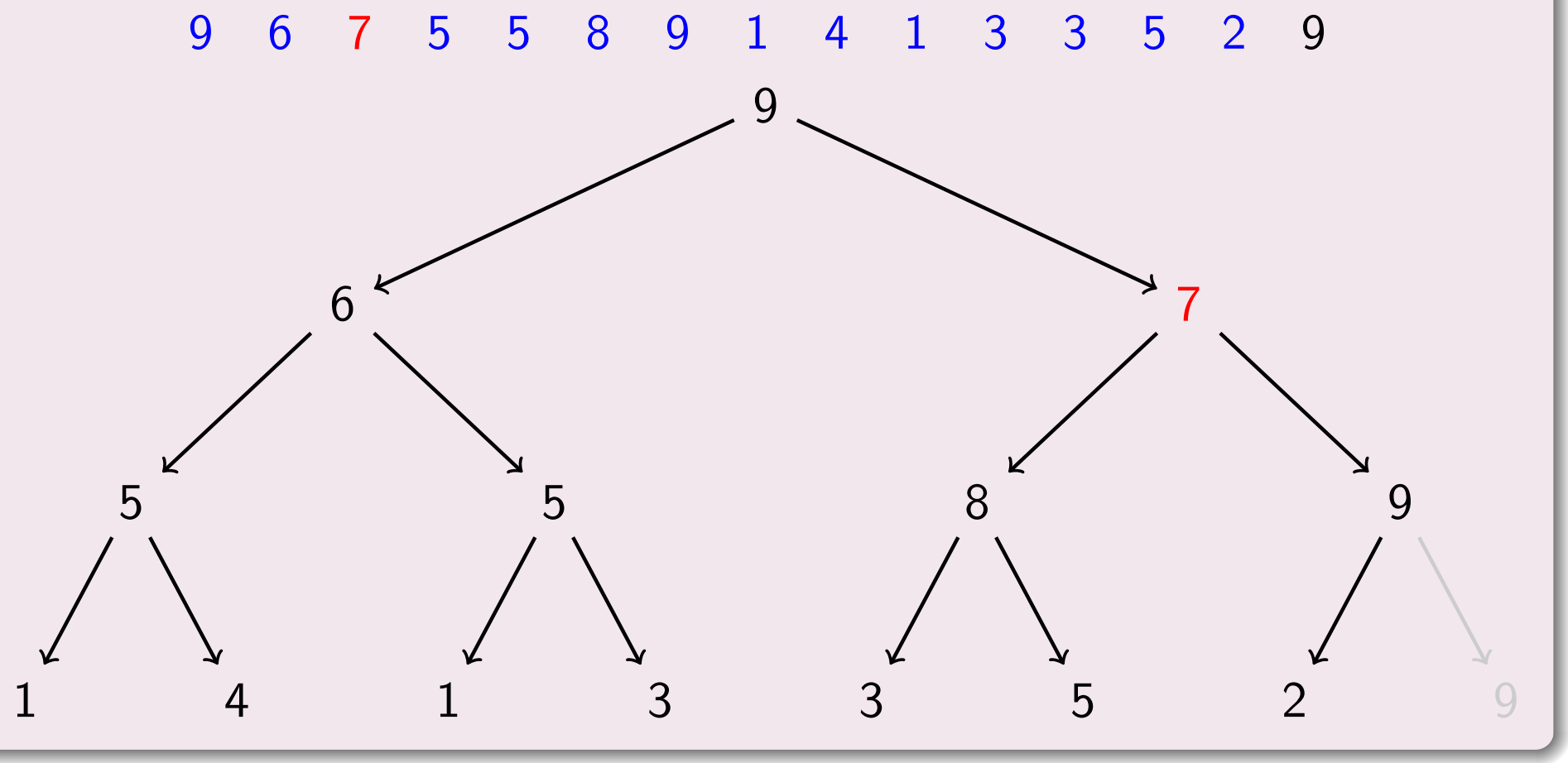
Montículos

Eliminación



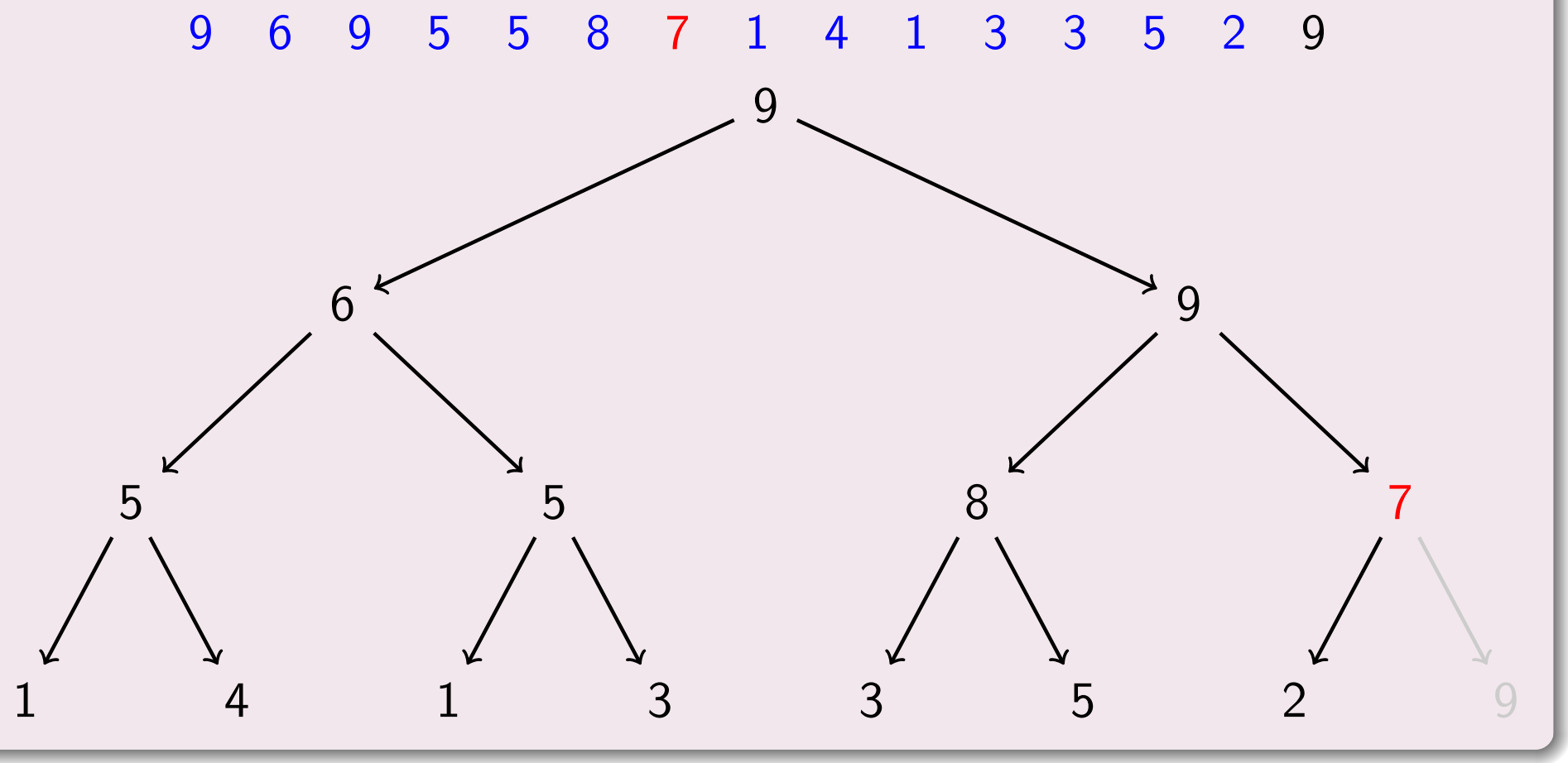
Montículos

Eliminación



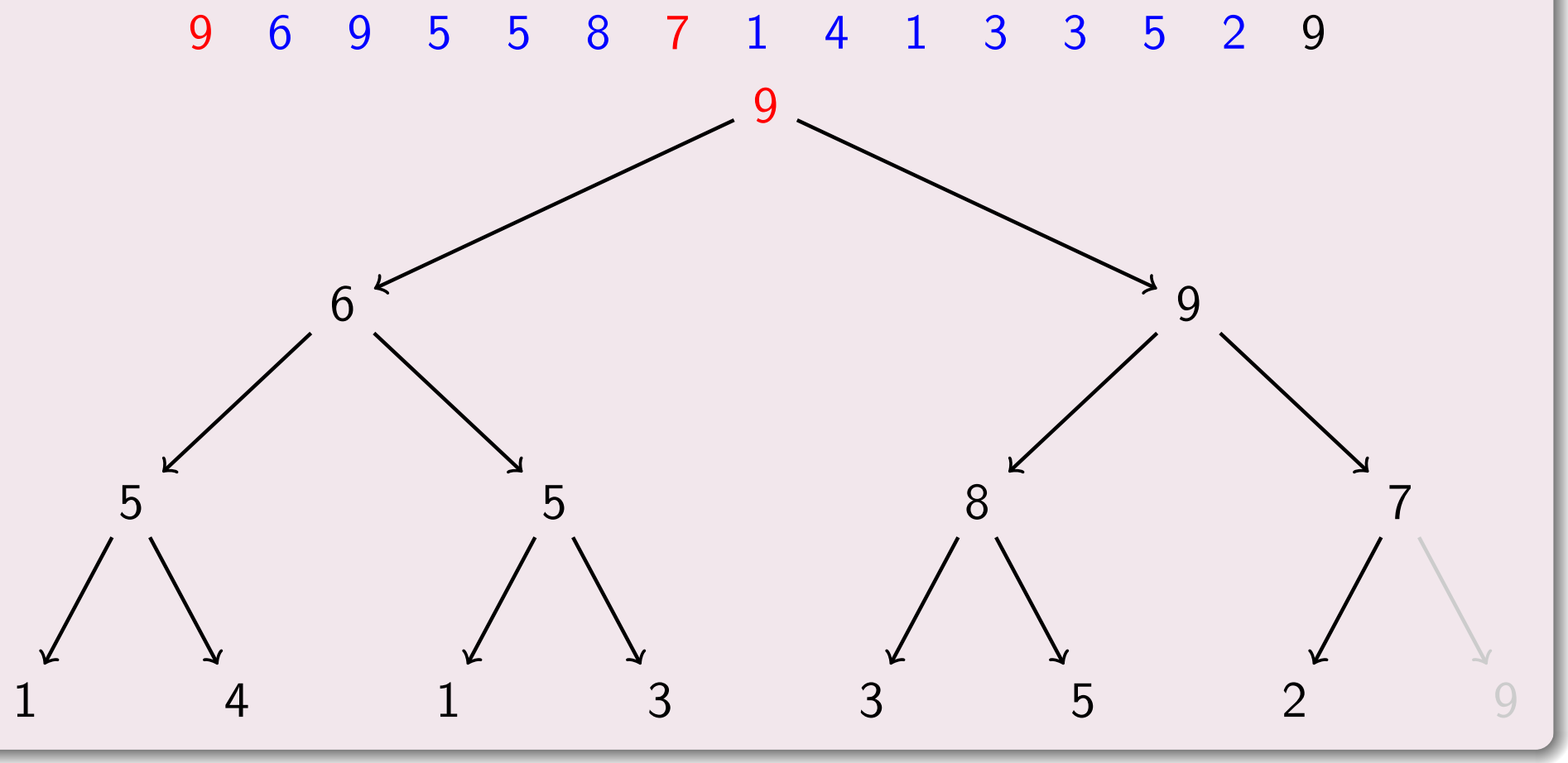
Montículos

Eliminación



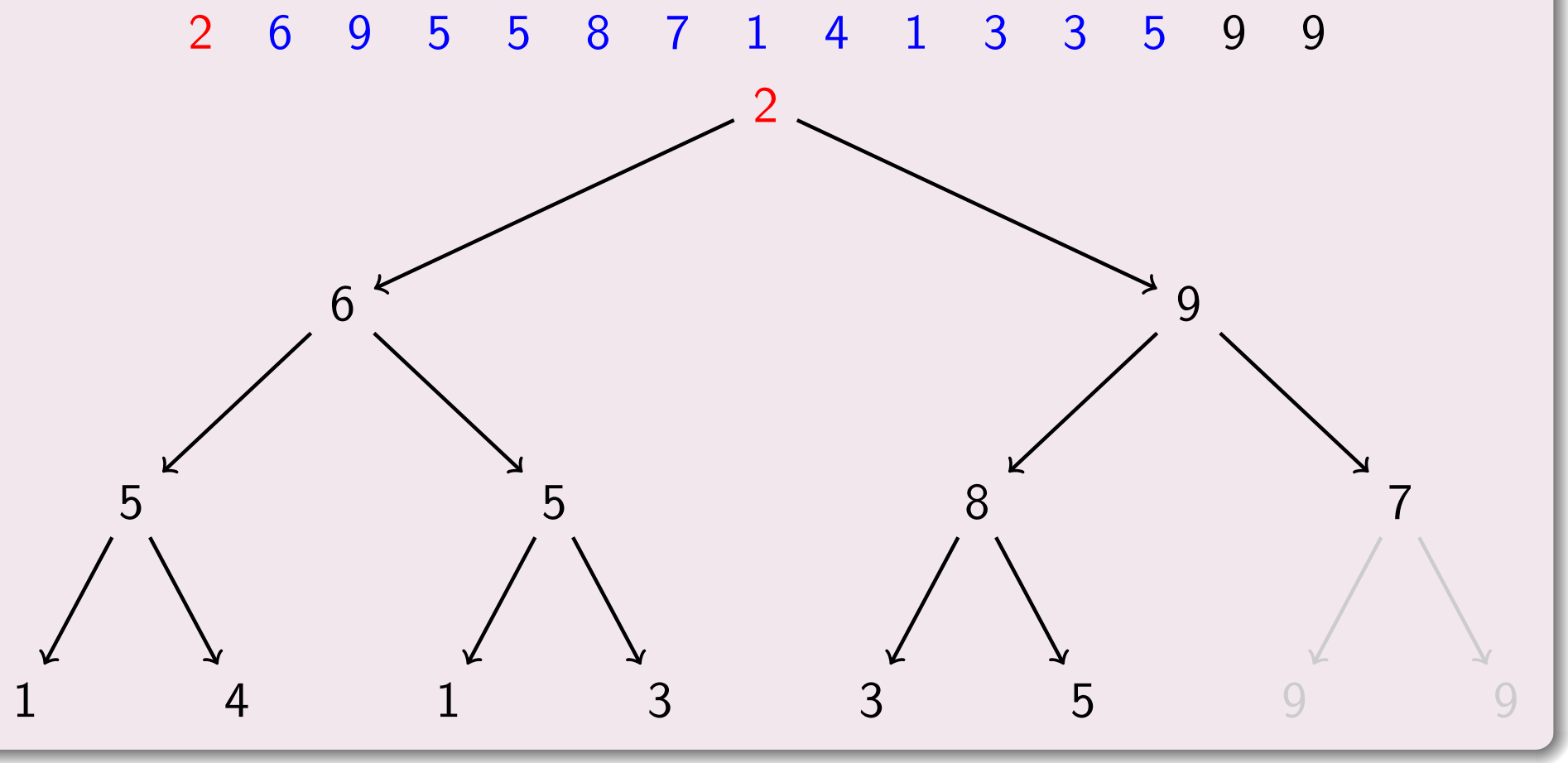
Montículos

Eliminación



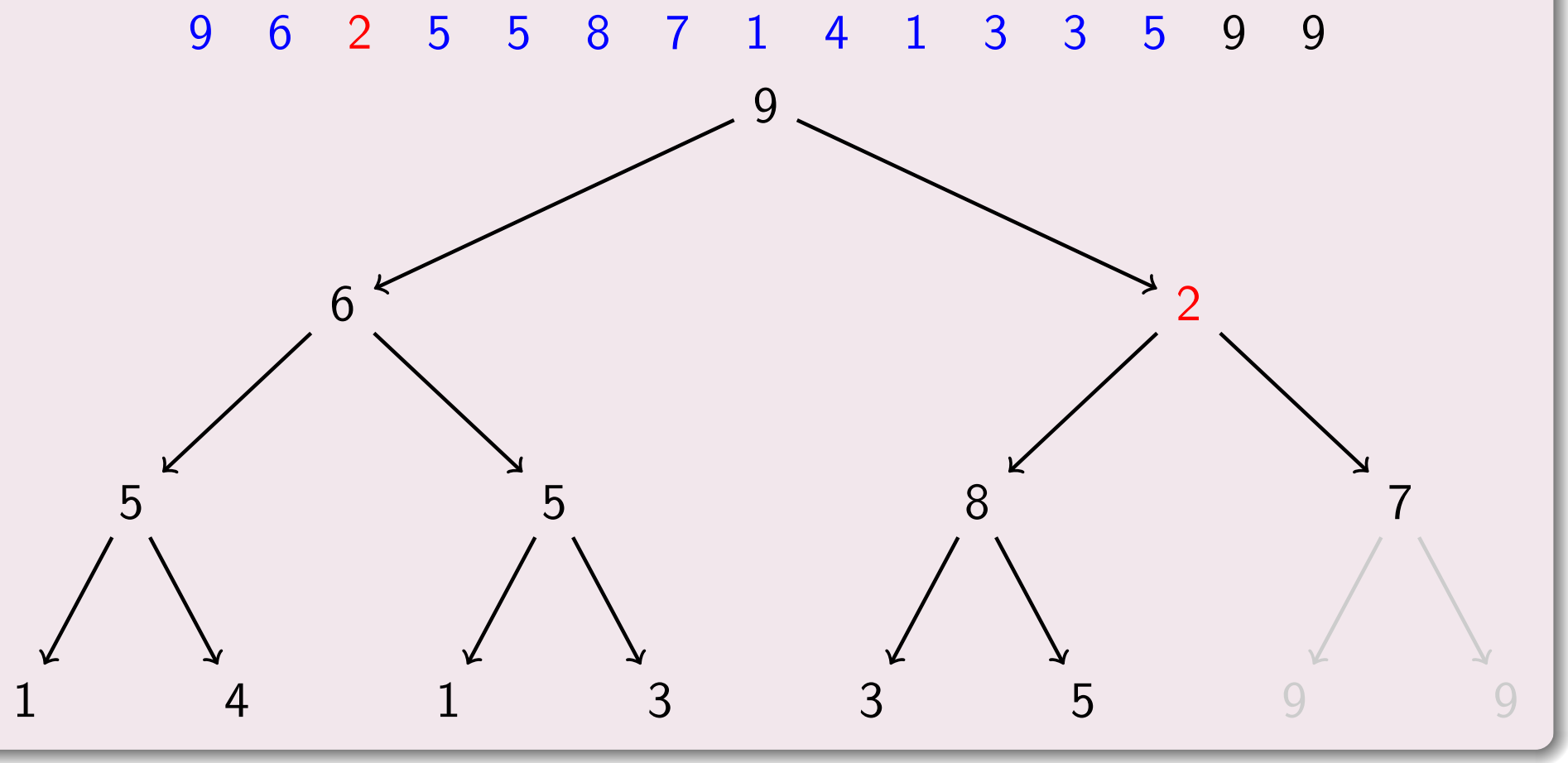
Montículos

Eliminación



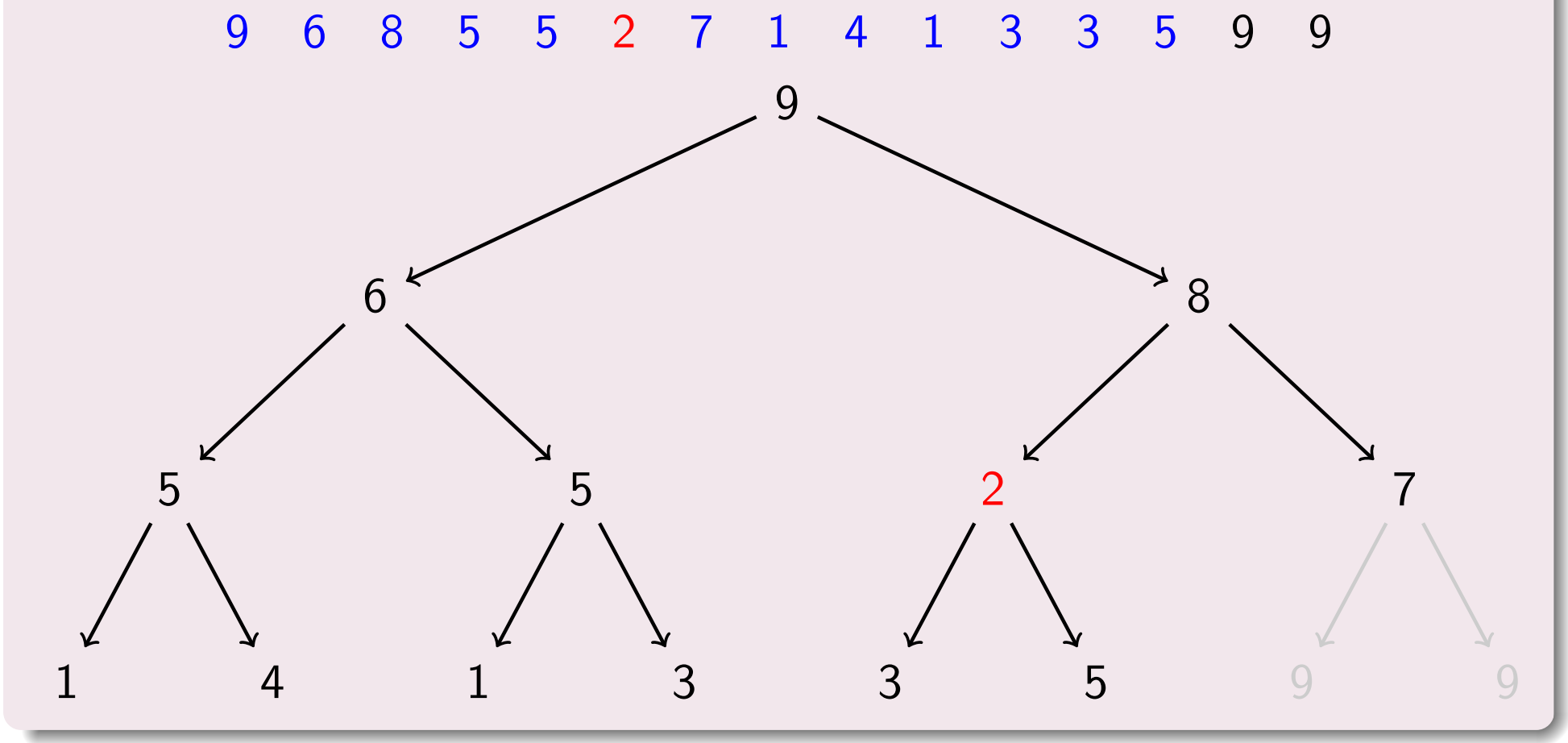
Montículos

Eliminación



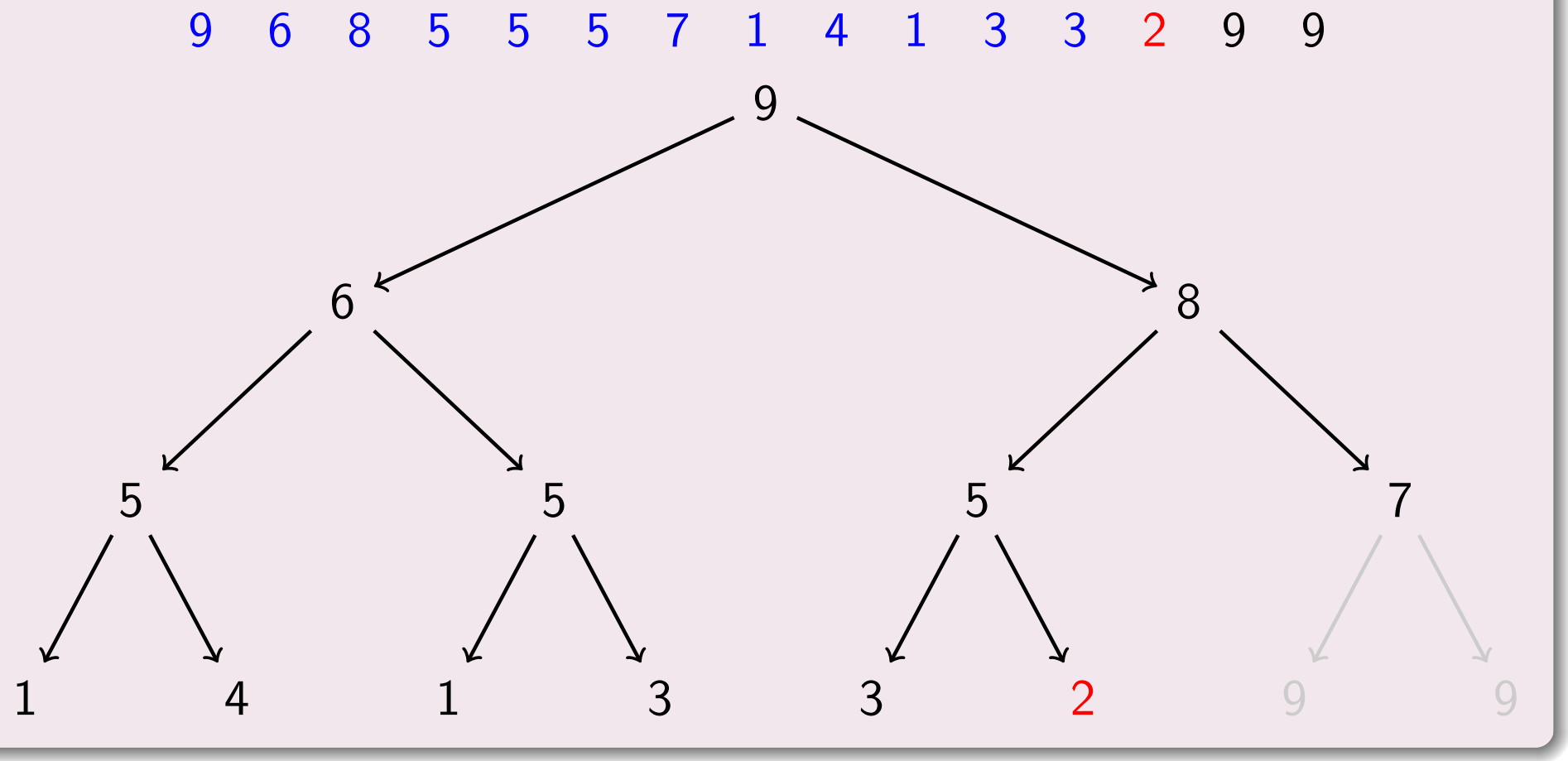
Montículos

Eliminación



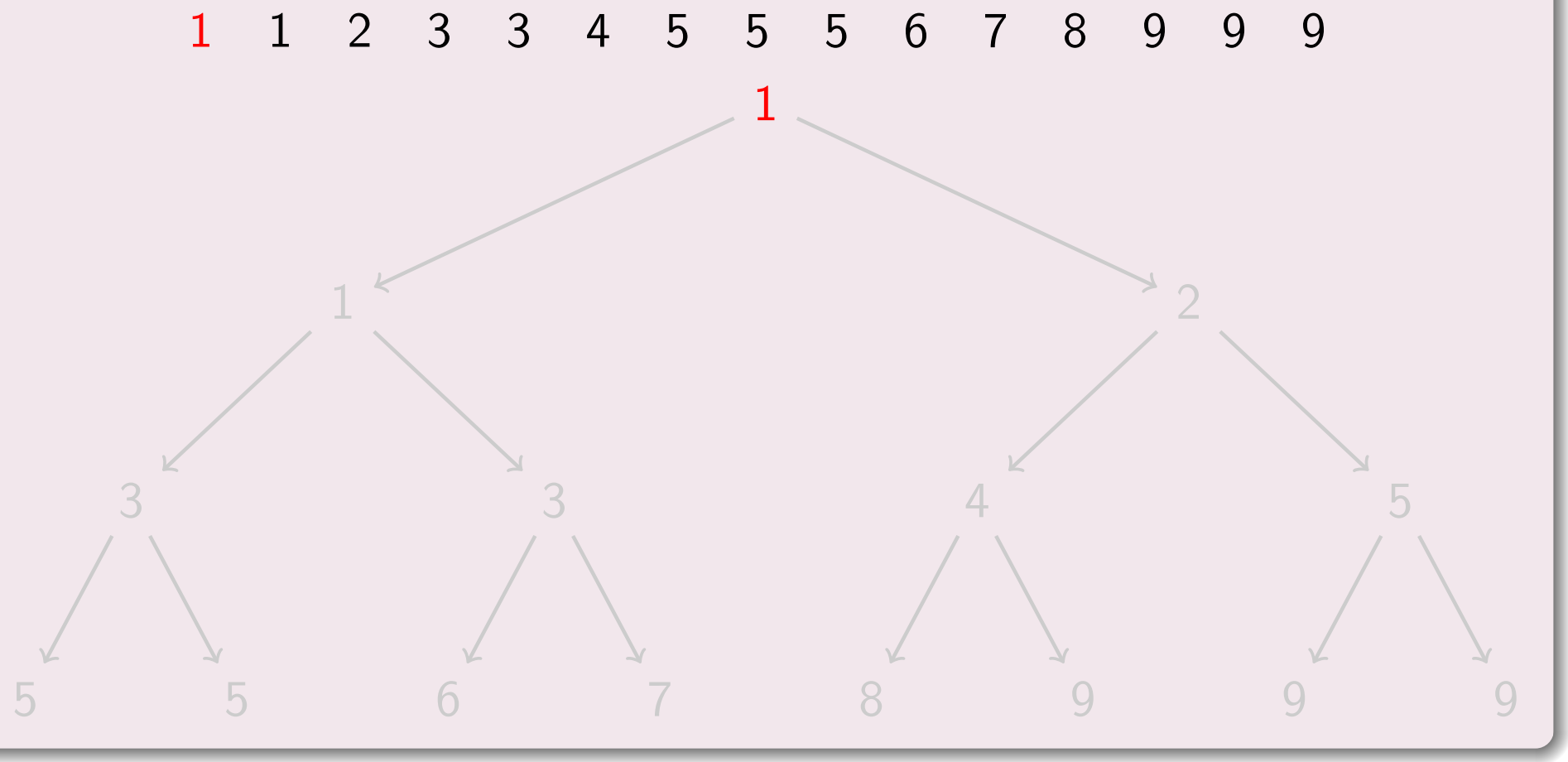
Montículos

Eliminación



Montículos

El arreglo al final



Eliminación

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

```
void elimina(int j, int a[])
{
  int p = 0, h;

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

Destrucción de un montículo

```
void destruye(int n, int a[])
{
  int j;

  for (j = n-1; j > 0; j--)
    elimina(j, a);
}
```

Montículos

Destrucción de un montículo

```
void destruye(int n, int a[])
{
  int j;

  for (j = n-1; j > 0; j--)
    elimina(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

Comparación

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	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$
Promedio	n^2	n^2	n^2	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$

Ordenamiento por montículo

Comparación

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	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$
Promedio	n^2	n^2	n^2	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$

Otra comparación

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

Ordenamiento por montículo

Ejercicios

1. Si se quisiera ordenar de mayor a menor se usaría un montículo con la propiedad de orden invertida: el padre no debe ser mayor a sus hijos. Reescribe todas las funciones vistas de esta manera.
2. Aunque no es la convención de C, un montículo también se suele representar con la raíz en `a[1]`. En este caso, los hijos de `a[p]` son `a[2*p]` y `a[2*p+1]`, mientras que el padre de `a[h]` está en `a[h/2]`. Reescribe todas las funciones vistas de esta manera.