Ordenamiento rápido y Heap Sort

José Ortiz Beajr

Facultad de Ingeniería Eléctrica
Universidad Michoacana de San Nicolás de Hidalgo

Marzo 16, 2020

Contenido

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	Complejidad			¿En sitio?
	peor	*promedio*	*mejor*	

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	Complejidad			¿En sitio?
	peor	*promedio*	*mejor*	
Insertion-Sort				

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort				

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort				

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort				

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no

Algoritmos de ordenamiento vistos hasta el momento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no
??		$\Theta(n \log n)$		***sí***
??	$\Theta(n \log n)$			***sí***

Algoritmo de partición

- *Paso básico:* partir A en tres partes basado en un valor $v \in A$
  - A_L contiene el conjunto de elementos que son *menores que* v
  - A_v contiene el conjunto de elementos que son *iguales a* v
  - A_R contiene el conjunto de elementos que son *mayores que* v

Algoritmo de partición

- *Paso básico:* partir A en tres partes basado en un valor $v \in A$
  - A_L contiene el conjunto de elementos que son *menores que* v
  - A_v contiene el conjunto de elementos que son *iguales a* v
  - A_R contiene el conjunto de elementos que son *mayores que* v

Por ejemplo, $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$

Algoritmo de partición

- *Paso básico:* partir A en tres partes basado en un valor $v \in A$
  - A_L contiene el conjunto de elementos que son *menores que* v
  - A_v contiene el conjunto de elementos que son *iguales a* v
  - A_R contiene el conjunto de elementos que son *mayores que* v

Por ejemplo, $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
si tomamos $v = 5$

Algoritmo de partición

- *Paso básico:* partir A en tres partes basado en un valor $v \in A$
  - A_L contiene el conjunto de elementos que son *menores que* v
  - A_v contiene el conjunto de elementos que son *iguales a* v
  - A_R contiene el conjunto de elementos que son *mayores que* v

Por ejemplo, $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
si tomamos $v = 5$

$$A_L = \langle 2, 4, 1 \rangle$$

Algoritmo de partición

- *Paso básico:* partir A en tres partes basado en un valor $v \in A$
  - A_L contiene el conjunto de elementos que son *menores que* v
  - A_v contiene el conjunto de elementos que son *iguales a* v
  - A_R contiene el conjunto de elementos que son *mayores que* v

Por ejemplo, $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
si tomamos $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle$$

Algoritmo de partición

- *Paso básico:* partir A en tres partes basado en un valor $v \in A$
  - A_L contiene el conjunto de elementos que son *menores que* v
  - A_v contiene el conjunto de elementos que son *iguales a* v
  - A_R contiene el conjunto de elementos que son *mayores que* v

Por ejemplo, $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
si tomamos $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle \quad A_R = \langle 36, 21, 8, 13, 11, 20 \rangle$$

Algoritmo de partición

- *Paso básico:* partir A en tres partes basado en un valor $v \in A$
  - A_L contiene el conjunto de elementos que son *menores que* v
  - A_v contiene el conjunto de elementos que son *iguales a* v
  - A_R contiene el conjunto de elementos que son *mayores que* v

  Por ejemplo, $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
  si tomamos $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle \quad A_R = \langle 36, 21, 8, 13, 11, 20 \rangle$$

- *¿Podemos usar está idea para ordenar A?*

Algoritmo de partición

- *Paso básico:* partir A en tres partes basado en un valor $v \in A$
  - A_L contiene el conjunto de elementos que son *menores que* v
  - A_v contiene el conjunto de elementos que son *iguales a* v
  - A_R contiene el conjunto de elementos que son *mayores que* v

  Por ejemplo, $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
  si tomamos $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle \quad A_R = \langle 36, 21, 8, 13, 11, 20 \rangle$$

- *¿Podemos usar está idea para ordenar A?*
- *¿Podemos ordenar A* ***en sitio****?*

Otra estrategia de ordenamiento

- *Problema:* ordenar

Otra estrategia de ordenamiento

- *Problema:* ordenar

- *Idea:* reasignar la secuencia $A[1 \ldots n]$ en tres partes basados en el *"pivote"* elegido $v \in A$
  - $A[1 \ldots q-1]$ contiene los elementos que son *menores o iguales que* v
  - $A[q] = v$
  - $A[q+1 \ldots n]$ contiene los elementos que son *mayores que* v

Otra estrategia de ordenamiento

- *Problema:* ordenar

- *Idea:* reasignar la secuencia $A[1 \ldots n]$ en tres partes basados en el *"pivote"* elegido $v \in A$
  - $A[1 \ldots q-1]$ contiene los elementos que son *menores o iguales que* v
  - $A[q] = v$
  - $A[q+1 \ldots n]$ contiene los elementos que son *mayores que* v

2	36	5	21	8	13	11	20	5	4	1

$v = 8$

Otra estrategia de ordenamiento

- *Problema:* ordenar

- *Idea:* reasignar la secuencia $A[1 \ldots n]$ en tres partes basados en el *"pivote"* elegido $v \in A$
  - $A[1 \ldots q-1]$ contiene los elementos que son *menores o iguales que* v
  - $A[q] = v$
  - $A[q+1 \ldots n]$ contiene los elementos que son *mayores que* v

2	36	5	21	8	13	11	20	5	4	1

$v = 8$

2	4	1	5	5						

Otra estrategia de ordenamiento

- *Problema:* ordenar

- *Idea:* reasignar la secuencia $A[1 \ldots n]$ en tres partes basados en el *"pivote"* elegido $v \in A$
  - $A[1 \ldots q-1]$ contiene los elementos que son *menores o iguales que* v
  - $A[q] = v$
  - $A[q+1 \ldots n]$ contiene los elementos que son *mayores que* v

2	36	5	21	8	13	11	20	5	4	1

$v = 8$

2	4	1	5	5	8					

Otra estrategia de ordenamiento

- *Problema:* ordenar

- *Idea:* reasignar la secuencia $A[1 \ldots n]$ en tres partes basados en el *"pivote"* elegido $v \in A$
  - $A[1 \ldots q-1]$ contiene los elementos que son *menores o iguales que* v
  - $A[q] = v$
  - $A[q+1 \ldots n]$ contiene los elementos que son *mayores que* v

2	36	5	21	8	13	11	20	5	4	1

$v = 8$

2	4	1	5	5	8	11	20	13	36	21

Otra estrategia de ordenamiento

- *Problema:* ordenar

- *Idea:* reasignar la secuencia $A[1 \ldots n]$ en tres partes basados en el *"pivote"* elegido $v \in A$
  - $A[1 \ldots q-1]$ contiene los elementos que son *menores o iguales que* v
  - $A[q] = v$
  - $A[q+1 \ldots n]$ contiene los elementos que son *mayores que* v

2	36	5	21	8	13	11	20	5	4	1

$v = 8$

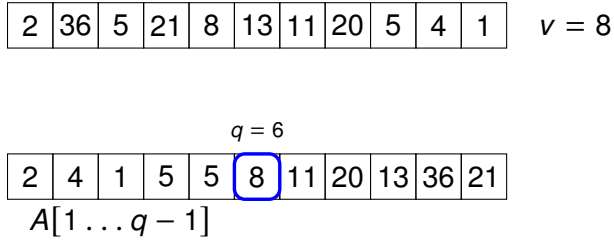
$q = 6$

2	4	1	5	5	8	11	20	13	36	21

Otra estrategia de ordenamiento

- *Problema:* ordenar

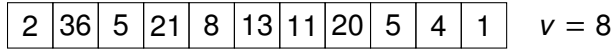
- *Idea:* reasignar la secuencia $A[1 \ldots n]$ en tres partes basados en el *"pivote"* elegido $v \in A$
  - $A[1 \ldots q-1]$ contiene los elementos que son *menores o iguales que* v
  - $A[q] = v$
  - $A[q+1 \ldots n]$ contiene los elementos que son *mayores que* v

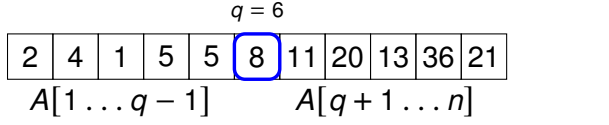


Otra estrategia de ordenamiento

- *Problema:* ordenar

- *Idea:* reasignar la secuencia $A[1 \ldots n]$ en tres partes basados en el *"pivote"* elegido $v \in A$
  - $A[1 \ldots q-1]$ contiene los elementos que son *menores o iguales que* v
  - $A[q] = v$
  - $A[q+1 \ldots n]$ contiene los elementos que son *mayores que* v





Otra estrategia de ordenamiento Divide y Vencerás

- ***Divide:***

Otra estrategia de ordenamiento Divide y Vencerás

- ***Divide:*** particionar A en $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$ tal que

$$1 \leq i < q < j \leq n \Rightarrow A[i] \leq A[q] \leq A[j]$$

Otra estrategia de ordenamiento Divide y Vencerás

- ***Divide:*** particionar A en $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$ tal que

$$1 \leq i < q < j \leq n \Rightarrow A[i] \leq A[q] \leq A[j]$$

- ***Conquista:***

Otra estrategia de ordenamiento Divide y Vencerás

- ***Divide:*** particionar A en $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$ tal que

$$1 \leq i < q < j \leq n \Rightarrow A[i] \leq A[q] \leq A[j]$$

- ***Conquista:*** ordenar $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$

Otra estrategia de ordenamiento Divide y Vencerás

- ***Divide:*** particionar A en $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$ tal que

$$1 \leq i < q < j \leq n \Rightarrow A[i] \leq A[q] \leq A[j]$$

- ***Conquista:*** ordenar $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$
- ***Combinar:***

Otra estrategia de ordenamiento Divide y Vencerás

- ***Divide:*** particionar A en $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$ tal que

$$1 \leq i < q < j \leq n \Rightarrow A[i] \leq A[q] \leq A[j]$$

- ***Conquista:*** ordenar $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$
- ***Combinar:*** no hay paso de combinar
  - note la diferencia con **MergeSort**

Otra estrategia de ordenamiento Divide y Vencerás

- ***Divide:*** particionar A en $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$ tal que

$$1 \leq i < q < j \leq n \Rightarrow A[i] \leq A[q] \leq A[j]$$

- ***Conquista:*** ordenar $A[1 \ldots q-1]$ y $A[q+1 \ldots n]$
- ***Combinar:*** no hay paso de combinar
  - note la diferencia con **MergeSort**

QuickSort$(A, \mathit{begin}, \mathit{end})$

```
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

Partición

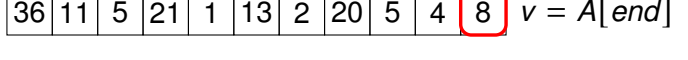
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha

Partición

- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$

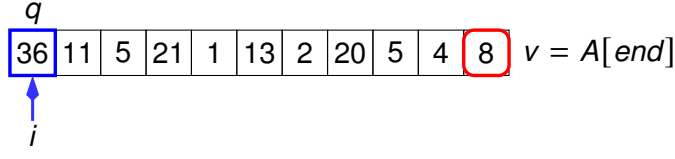
Partición

- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



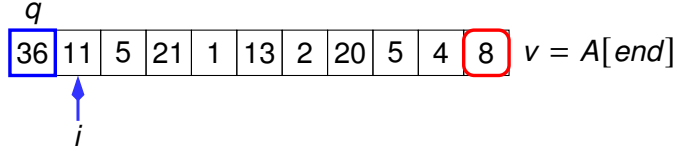
Partición

- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$

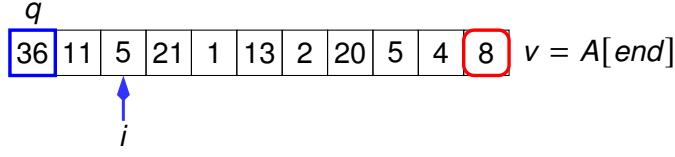


Partición

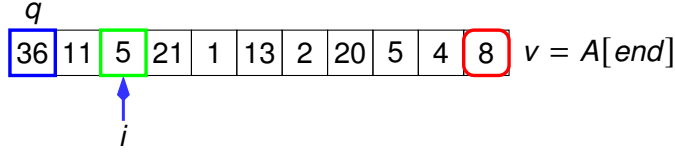
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



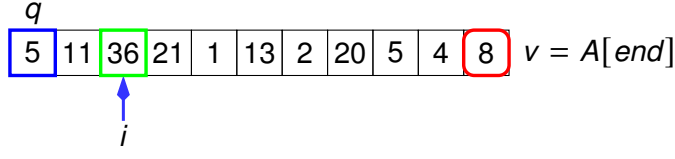
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



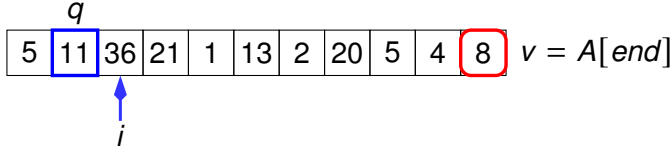
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



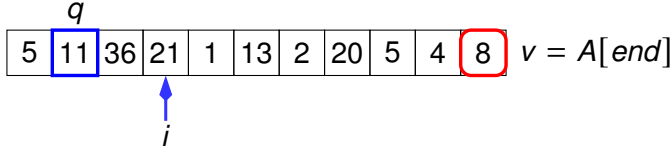
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$

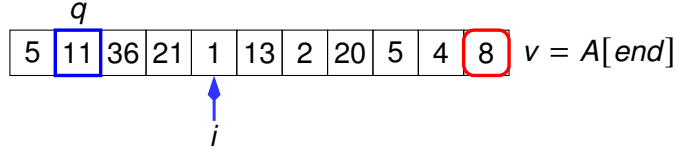


- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



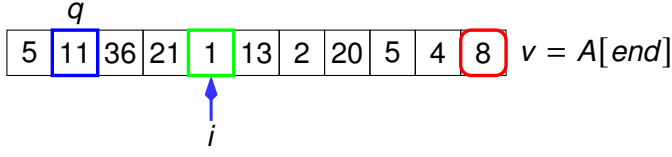
Partición

- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



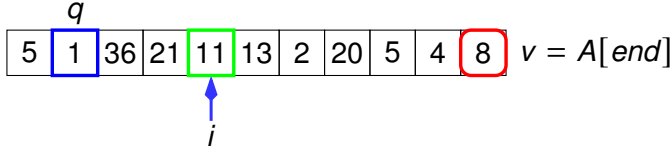
Partición

- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$

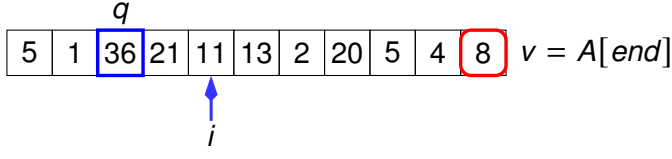


Partición

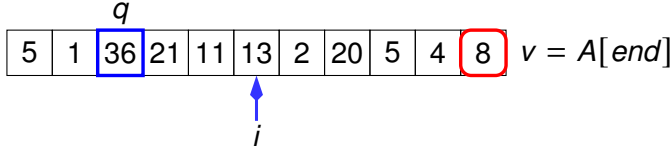
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



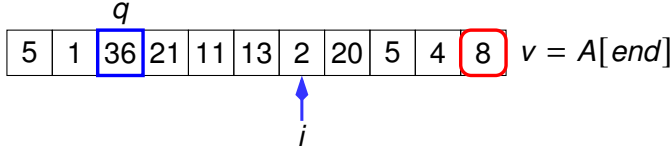
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



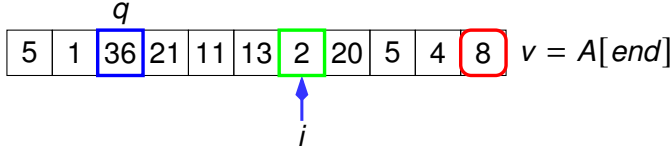
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



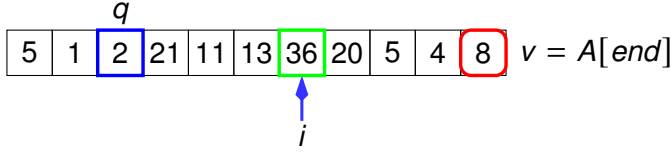
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



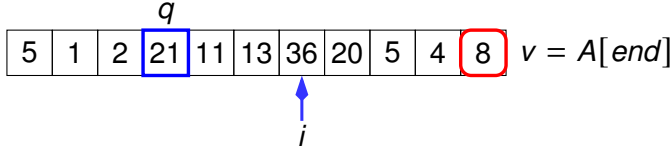
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



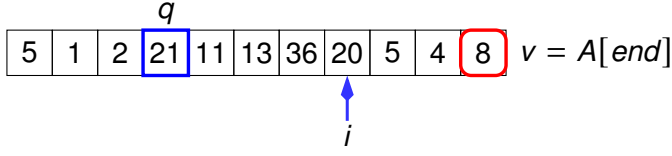
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



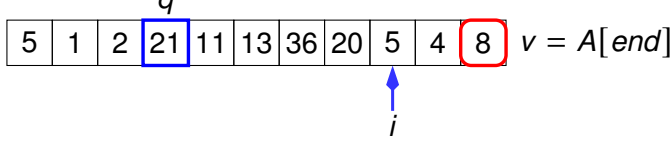
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



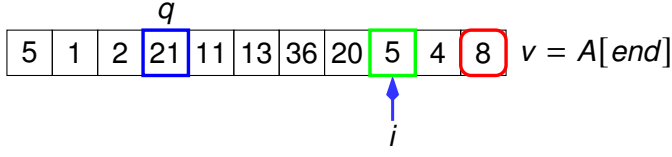
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



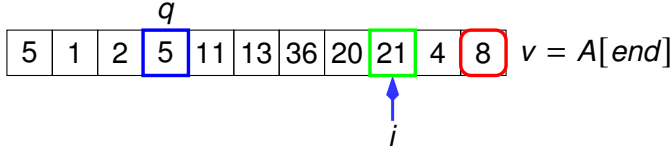
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



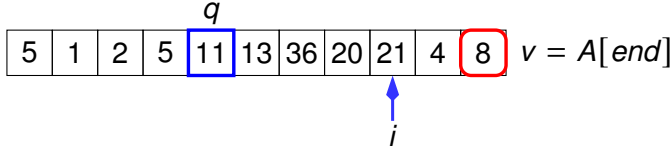
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



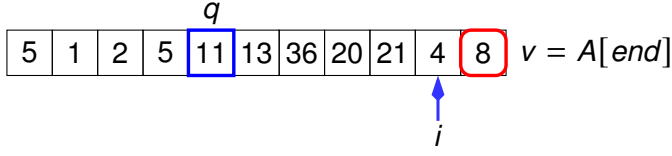
- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$

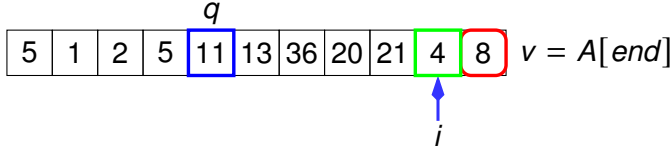


- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$

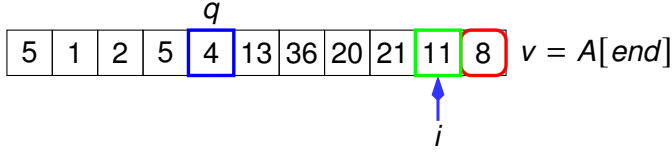


Partición

- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$

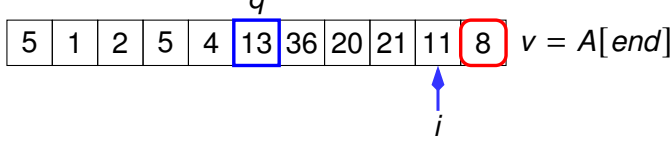


- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



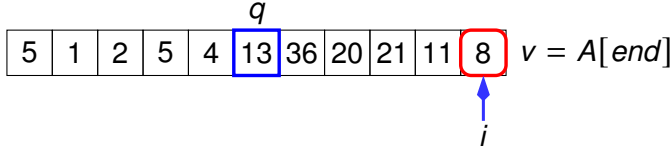
Partición

- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



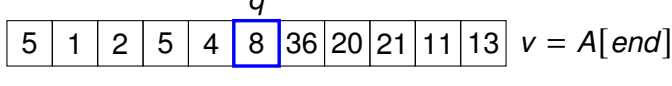
Partición

- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



Partición

- Comenzar con $q = 1$
  - e.i., *asumir que todos los elementos son mayores que el pivote*
- Leer el arreglo de izquierda a derecha, comenzando en la posición 2
- Si un elemento $A[i]$ es menor o igual que el pivote, entoces intercambiar con la posición actual de q y desplazar q a la derecha
- *Invariante de ciclo*
  - $begin \leq k < q \Rightarrow A[k] \leq v$
  - $q < k < i \Rightarrow A[k] > v$



Algoritmo QuickSort

```
Partition(A, begin, end)
1  q = begin
2  v = A[end]
3  for i = begin to end
4       if A[i] ≤ v
5            swap(A[i], A[q])
6            q = q + 1
7  return q − 1
```

```
QuickSort(A, begin, end)
1  if begin < end
2       q = Partition(A, begin, end)
3       QuickSort(A, begin, q − 1)
4       QuickSort(A, q + 1, end)
```

Complejidad de **Partition**

```
Partition(A, begin, end)
1  q = begin
2  v = A[end]
3  for i = begin to end
4       if A[i] ≤ v
5            swap (A[i], A[q])
6            q = q + 1
7  return q − 1
```

Complejidad de **Partition**

Partition$(A, begin, end)$

```
1  q = begin
2  v = A[end]
3  for i = begin to end
4       if A[i] ≤ v
5            swap (A[i], A[q])
6            q = q + 1
7  return q − 1
```

$$T(n) = \Theta(n)$$

Complejidad de QuickSort

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

Complejidad de QuickSort

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

- Peor caso

Complejidad de QuickSort

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

- Peor caso
  - $q = begin$ o $q = end$

Complejidad de QuickSort

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

- Peor caso
  - $q = begin$ o $q = end$
  - el procedimiento *Partition* transforma P de tamaño n en P de tamaño $n - 1$

Complejidad de QuickSort

```
QuickSort(A, begin, end) 1  if begin < end
                         2      q = Partition(A, begin, end)
                         3      QuickSort(A, begin, q − 1)
                         4      QuickSort(A, q + 1, end)
```

- Peor caso
  - $q = begin$ o $q = end$
  - el procedimiento *Partition* transforma P de tamaño n en P de tamaño $n - 1$

$$T(n) = T(n-1) + \Theta(n)$$

Complejidad de QuickSort

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

- Peor caso
  - $q = begin$ o $q = end$
  - el procedimiento *Partition* transforma P de tamaño n en P de tamaño $n - 1$

$$T(n) = T(n-1) + \Theta(n)$$

$$T(n) = \Theta(n^2)$$

Complejidad de QuickSort (2)

```
QuickSort(A, begin, end)
1   if begin < end
2       q = Partition(A, begin, end)
3       QuickSort(A, begin, q − 1)
4       QuickSort(A, q + 1, end)
```

Complejidad de QuickSort (2)

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

- Mejor caso

Complejidad de QuickSort (2)

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

- Mejor caso
  - $q = \lceil n/2 \rceil$

Complejidad de QuickSort (2)

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

- Mejor caso
  - $q = \lceil n/2 \rceil$
  - *Partition* transforma P de tamaño n en *dos* problemas P de tamaño $\lfloor n/2 \rfloor$ y $\lceil n/2 \rceil - 1$, respectivamente

Complejidad de QuickSort (2)

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

- Mejor caso
  - $q = \lceil n/2 \rceil$
  - *Partition* transforma P de tamaño n en *dos* problemas P de tamaño $\lfloor n/2 \rfloor$ y $\lceil n/2 \rceil - 1$, respectivamente

$$T(n) = 2T(n/2) + \Theta(n)$$

Complejidad de QuickSort (2)

```
QuickSort(A, begin, end)
1  if begin < end
2      q = Partition(A, begin, end)
3      QuickSort(A, begin, q − 1)
4      QuickSort(A, q + 1, end)
```

- Mejor caso
  - $q = \lceil n/2 \rceil$
  - *Partition* transforma P de tamaño n en *dos* problemas P de tamaño $\lfloor n/2 \rfloor$ y $\lceil n/2 \rceil - 1$, respectivamente

$$T(n) = 2T(n/2) + \Theta(n)$$

$$T(n) = \Theta(n \log n)$$

Algoritmos de ordenamiento vistos hasta ahora

Algoritmos de ordenamiento vistos hasta ahora

Algorithm	**Complejidad**			**¿En sitio?**
	pero	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no

Algoritmos de ordenamiento vistos hasta ahora

Algorithm	**Complejidad**			**¿En sitio?**
	pero	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no
QuickSort				

Algoritmos de ordenamiento vistos hasta ahora

Algorithm	**Complejidad**			**¿En sitio?**
	pero	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no
QuickSort	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$	sí

Algoritmos de ordenamiento vistos hasta ahora

Algorithm	**Complejidad**			**¿En sitio?**
	pero	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no
QuickSort	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$	sí
??	$\Theta(n \log n)$			***sí***

Montículo (Heap) binario

- La primera ***estructura de datos*** que vamos a ver

Montículo (Heap) binario

- La primera ***estructura de datos*** que vamos a ver
- Interfaz

Montículo (Heap) binario

- La primera ***estructura de datos*** que vamos a ver
- Interfaz
  - **Build-Max-Heap**(A) convertir A en un max-heap
  - **Heap-Insert**(H, key) insertar *key* en el heap
  - **Heap-Extract-Max**(H) extrae la clave con el valor máximo en el heap
  - $H.heap\text{-}size$ es el número de elementos en el heap H

Montículo (Heap) binario

- La primera ***estructura de datos*** que vamos a ver
- Interfaz
  - **Build-Max-Heap**(A) convertir A en un max-heap
  - **Heap-Insert**(H, key) insertar *key* en el heap
  - **Heap-Extract-Max**(H) extrae la clave con el valor máximo en el heap
  - $H.heap\text{-}size$ es el número de elementos en el heap H
- Dos clases de Heaps binarios

Montículo (Heap) binario

- La primera ***estructura de datos*** que vamos a ver
- Interfaz
  - **Build-Max-Heap**(A) convertir A en un max-heap
  - **Heap-Insert**(H, key) insertar *key* en el heap
  - **Heap-Extract-Max**(H) extrae la clave con el valor máximo en el heap
  - $H.heap\text{-}size$ es el número de elementos en el heap H
- Dos clases de Heaps binarios
  - max-heaps

Montículo (Heap) binario

- La primera ***estructura de datos*** que vamos a ver
- Interfaz
  - **Build-Max-Heap**(A) convertir A en un max-heap
  - **Heap-Insert**(H, key) insertar *key* en el heap
  - **Heap-Extract-Max**(H) extrae la clave con el valor máximo en el heap
  - $H.heap\text{-}size$ es el número de elementos en el heap H
- Dos clases de Heaps binarios
  - max-heaps
  - min-heaps

Montículo (Heap) binario

- La primera ***estructura de datos*** que vamos a ver
- Interfaz
  - **Build-Max-Heap**(A) convertir A en un max-heap
  - **Heap-Insert**(H, key) insertar *key* en el heap
  - **Heap-Extract-Max**(H) extrae la clave con el valor máximo en el heap
  - $H.heap\text{-}size$ es el número de elementos en el heap H
- Dos clases de Heaps binarios
  - max-heaps
  - min-heaps
- Aplicaciones

Montículo (Heap) binario

- La primera ***estructura de datos*** que vamos a ver
- Interfaz
  - **Build-Max-Heap**(A) convertir A en un max-heap
  - **Heap-Insert**(H, key) insertar *key* en el heap
  - **Heap-Extract-Max**(H) extrae la clave con el valor máximo en el heap
  - $H.heap\text{-}size$ es el número de elementos en el heap H
- Dos clases de Heaps binarios
  - max-heaps
  - min-heaps
- Aplicaciones
  - Ordenamiento

Montículo (Heap) binario

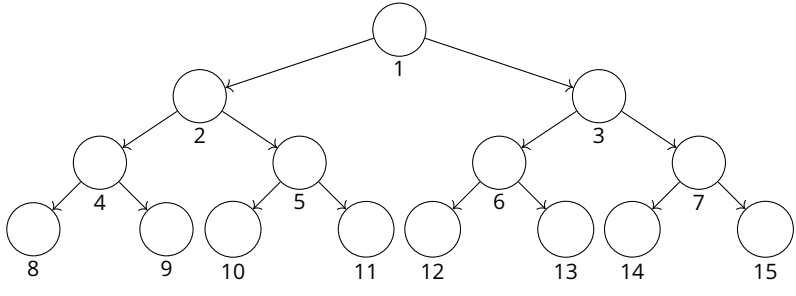
- La primera ***estructura de datos*** que vamos a ver
- Interfaz
  - **Build-Max-Heap**(A) convertir A en un max-heap
  - **Heap-Insert**(H, key) insertar *key* en el heap
  - **Heap-Extract-Max**(H) extrae la clave con el valor máximo en el heap
  - $H.heap\text{-}size$ es el número de elementos en el heap H
- Dos clases de Heaps binarios
  - max-heaps
  - min-heaps
- Aplicaciones
  - Ordenamiento
  - Cola de prioridad

Estructura de un Heap binario

- Árbol binario completo

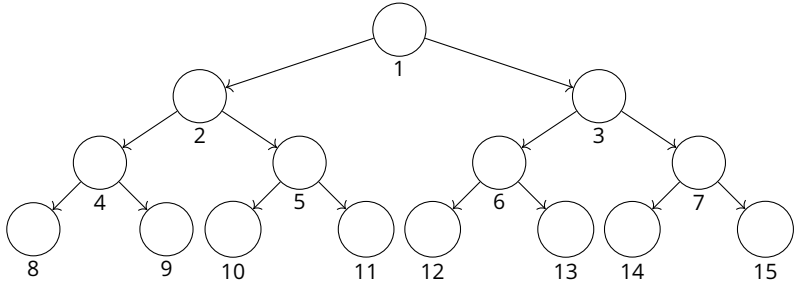
Estructura de un Heap binario

- Árbol binario completo

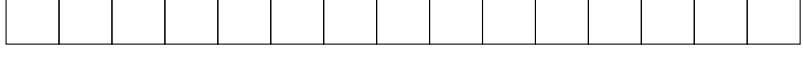


Estructura de un Heap binario

- Árbol binario completo

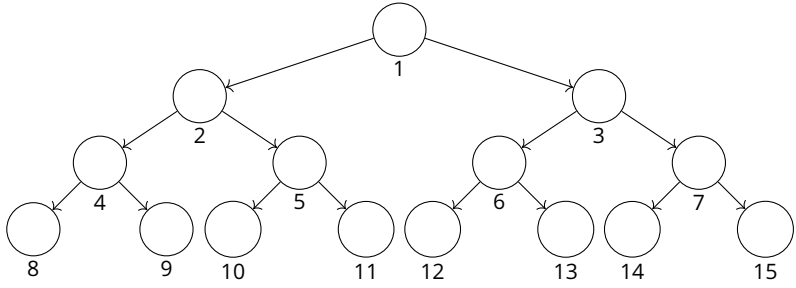


- Implementado como un array

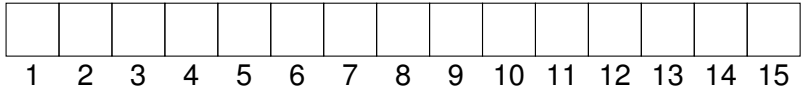


Estructura de un Heap binario

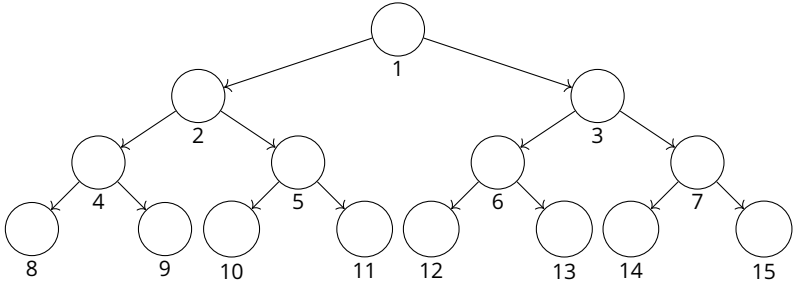
- Árbol binario completo



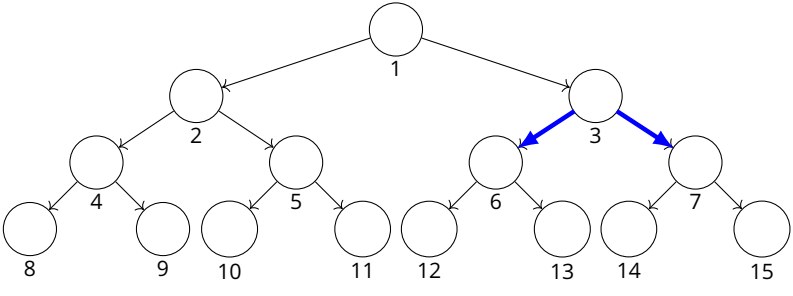
- Implementado como un array



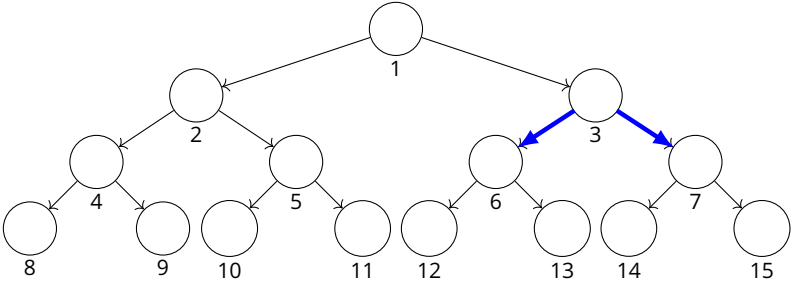
Propiedades del Heap binario



Propiedades del Heap binario

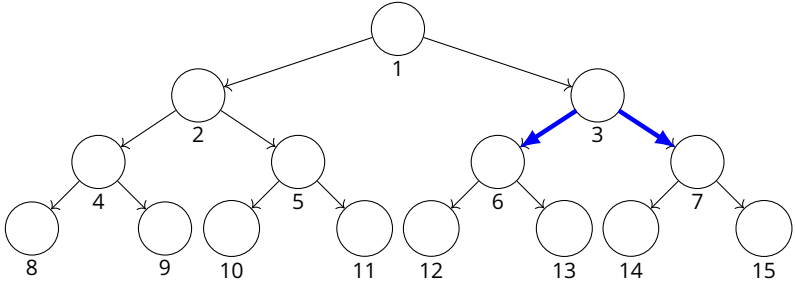


Propiedades del Heap binario

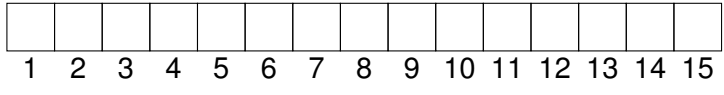


Parent(i)
    return $\lfloor \frac{i}{2} \rfloor$
Left(i)
    return $2i$
Right(i)
    return $2i + 1$

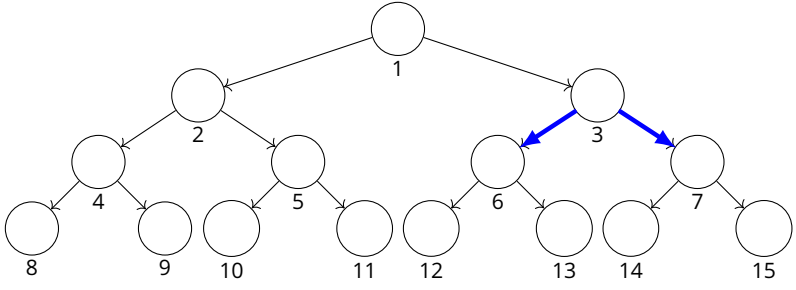
Propiedades del Heap binario



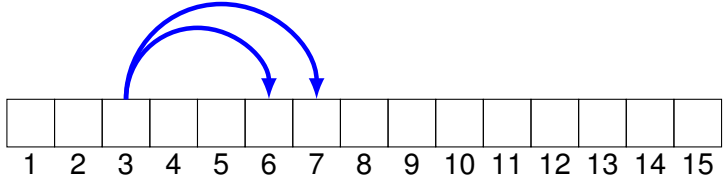
Parent(i)
    return $\lfloor \frac{i}{2} \rfloor$
Left(i)
    return $2i$
Right(i)
    return $2i + 1$



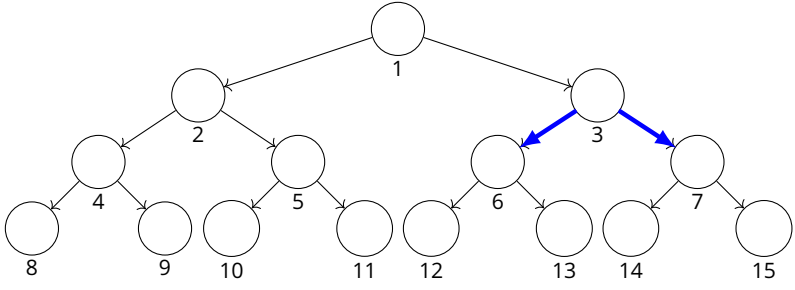
Propiedades del Heap binario



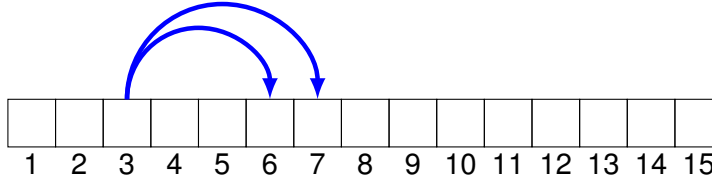
Parent(i)
    return $\lfloor \frac{i}{2} \rfloor$
Left(i)
    return $2i$
Right(i)
    return $2i + 1$



Propiedades del Heap binario



Parent(i)
  return $\lfloor \frac{i}{2} \rfloor$
Left(i)
  return $2i$
Right(i)
  return $2i + 1$



- ***propiedad del Max-heap:*** para todo $i > 1$, $A[\textbf{Parent}(i)] \geq A[i]$

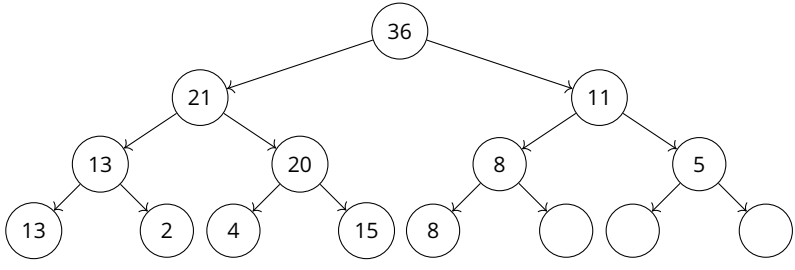
Ejemplo

- *propiedad del Max-heap* para todo $i > 1$, $A[\textbf{Parent}(i)] \geq A[i]$

Ejemplo

- *propiedad del Max-heap* para todo $i > 1$, $A[\textbf{Parent}(i)] \geq A[i]$

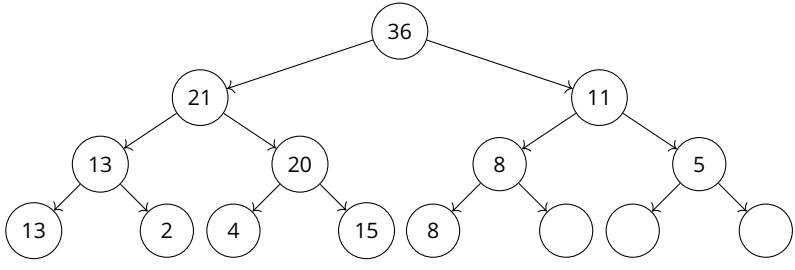
  Ej.,



Ejemplo

- *propiedad del Max-heap* para todo $i > 1$, $A[\textbf{Parent}(i)] \geq A[i]$

  Ej.,

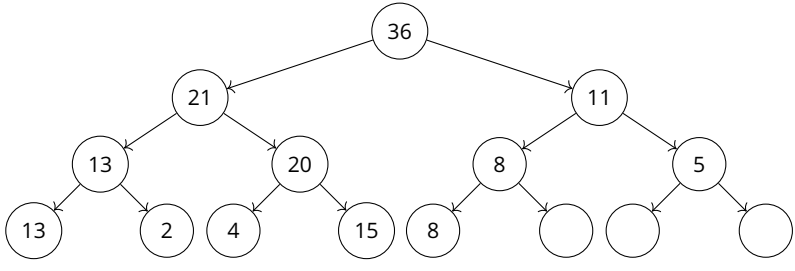


- ¿Dónde está el elemento con el valor máximo?

Ejemplo

- *propiedad del Max-heap* para todo $i > 1$, $A[\textbf{Parent}(i)] \geq A[i]$

  Ej.,



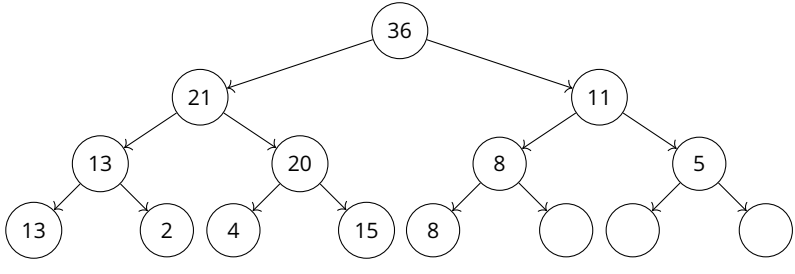
- ¿Dónde está el elemento con el valor máximo?
- ¿Cómo implementamos **Heap-Extract-Max**?

Heap-Extract-Max

- procedimiento **Heap-Extract-Max**
  - extraer el elemento con el valor máximo
  - reordenar el heap para mantener la *propiedad max-heap*

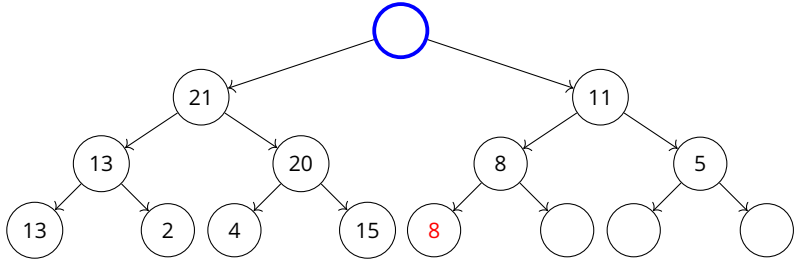
Heap-Extract-Max

- procedimiento **Heap-Extract-Max**
  - extraer el elemento con el valor máximo
  - reordenar el heap para mantener la *propiedad max-heap*

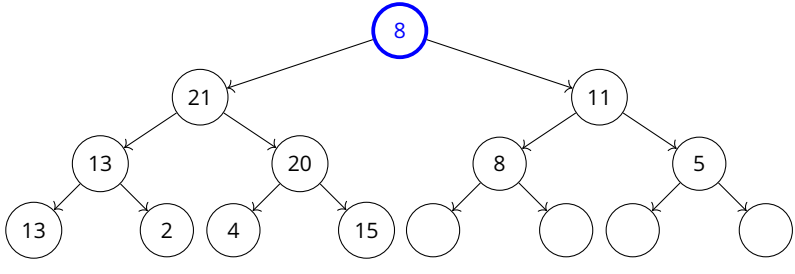


Heap-Extract-Max

- procedimiento **Heap-Extract-Max**
  - extraer el elemento con el valor máximo
  - reordenar el heap para mantener la *propiedad max-heap*



- procedimiento **Heap-Extract-Max**
  - extraer el elemento con el valor máximo
  - reordenar el heap para mantener la *propiedad max-heap*

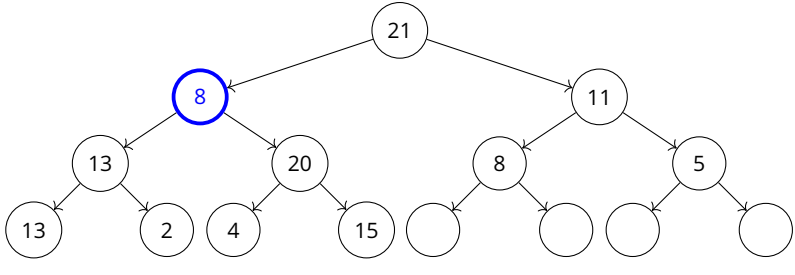


- Ahora se tienen dos sub-árboles donde la *propiedad max-heap* se mantiene

Max-Heapify

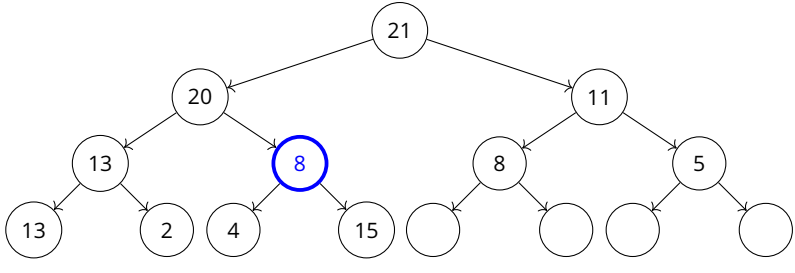
- Procedimiento **Max-Heapify**(A, i)
  - *asumir:* que la *propiedad max-heap* se mantiene en los sub-árboles del nodo i
  - *objetivo:* reordenar el heap para mantener la *propiedad max-heap*

- Procedimiento **Max-Heapify**(A, i)
  - *asumir:* que la *propiedad max-heap* se mantiene en los sub-árboles del nodo i
  - *objetivo:* reordenar el heap para mantener la *propiedad max-heap*



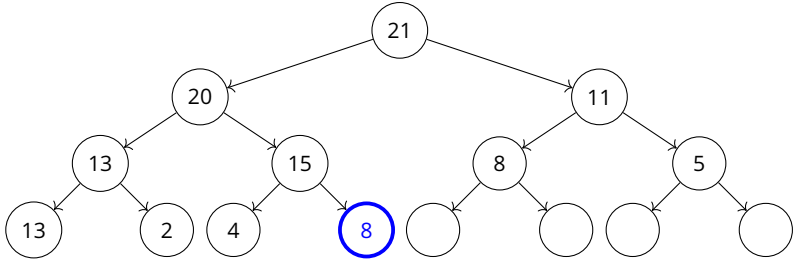
intercambiamos con el mayor de sus hijos mientras se viole la condición del heap

- Procedimiento **Max-Heapify**(A, i)
  - *asumir:* que la *propiedad max-heap* se mantiene en los sub-árboles del nodo i
  - *objetivo:* reordenar el heap para mantener la *propiedad max-heap*



intercambiamos con el mayor de sus hijos mientras se viole la condición del heap

- Procedimiento **Max-Heapify**(A, i)
  - *asumir:* que la *propiedad max-heap* se mantiene en los sub-árboles del nodo i
  - *objetivo:* reordenar el heap para mantener la *propiedad max-heap*



Terminamos!

Max-Heapify

```
Max-Heapify(A, i)  1  l = Left(i)
                   2  r = Right(i)
                   3  if l ≤ A.heap-size and A[l] > A[i]
                   4        largest = l
                   5  else largest = i
                   6  if r ≤ A.heap-size and A[r] > A[largest]
                   7        largest = r
                   8  if largest ≠ i
                   9        swap A[i] and A[largest]
                  10        Max-Heapify(A, largest)
```

Max-Heapify

```
Max-Heapify(A, i)  1  l = Left(i)
                   2  r = Right(i)
                   3  if l ≤ A.heap-size and A[l] > A[i]
                   4        largest = l
                   5  else largest = i
                   6  if r ≤ A.heap-size and A[r] > A[largest]
                   7        largest = r
                   8  if largest ≠ i
                   9        swap A[i] and A[largest]
                  10        Max-Heapify(A, largest)
```

- ¿Complejidad de **Max-Heapify**?

Max-Heapify

```
Max-Heapify(A, i)  1  l = Left(i)
                   2  r = Right(i)
                   3  if l ≤ A.heap-size and A[l] > A[i]
                   4        largest = l
                   5  else largest = i
                   6  if r ≤ A.heap-size and A[r] > A[largest]
                   7        largest = r
                   8  if largest ≠ i
                   9        swap A[i] and A[largest]
                  10        Max-Heapify(A, largest)
```

- ¿Complejidad de **Max-Heapify**? ¡La altura del árbol!

Max-Heapify

```
Max-Heapify(A, i)  1  l = Left(i)
                   2  r = Right(i)
                   3  if l ≤ A.heap-size and A[l] > A[i]
                   4        largest = l
                   5  else largest = i
                   6  if r ≤ A.heap-size and A[r] > A[largest]
                   7        largest = r
                   8  if largest ≠ i
                   9        swap A[i] and A[largest]
                  10        Max-Heapify(A, largest)
```

- ¿Complejidad de **Max-Heapify**? ¡La altura del árbol!

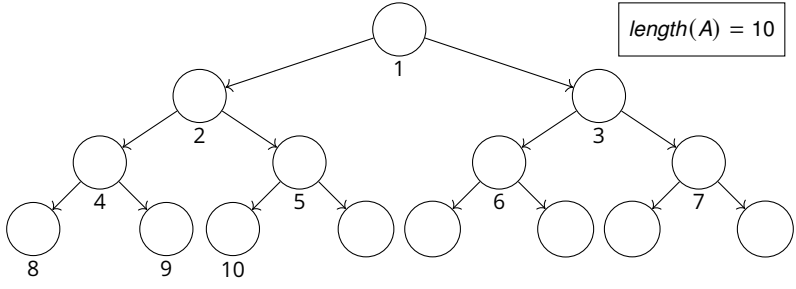
$$T(n) = \Theta(\log n)$$

Construcción del Heap

```
Build-Max-Heap(A) 1  A.heap-size = length(A)
                  2  for i = ⌊length(A)/2⌋ downto 1
                  3      Max-Heapify(A, i)
```

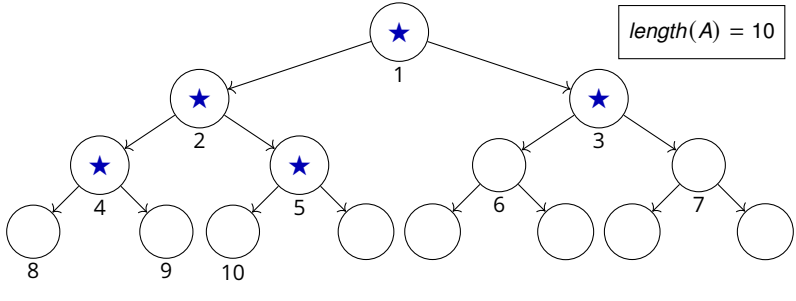
Construcción del Heap

Build-Max-Heap(A) 1 $A.heap\text{-}size = length(A)$
2 **for** $i = \lfloor length(A)/2 \rfloor$ **downto** 1
3 **Max-Heapify**(A, i)



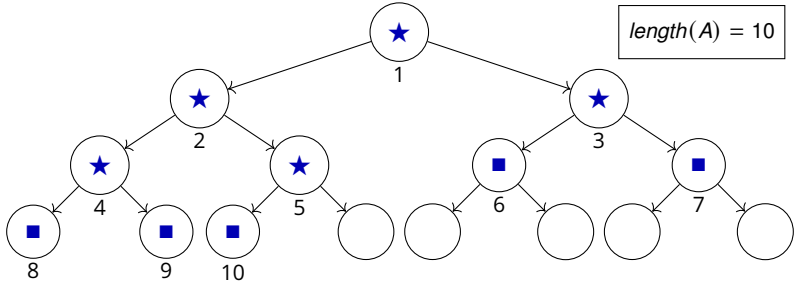
Construcción del Heap

```
Build-Max-Heap(A) 1  A.heap-size = length(A)
                  2  for i = ⌊length(A)/2⌋ downto 1
                  3      Max-Heapify(A, i)
```



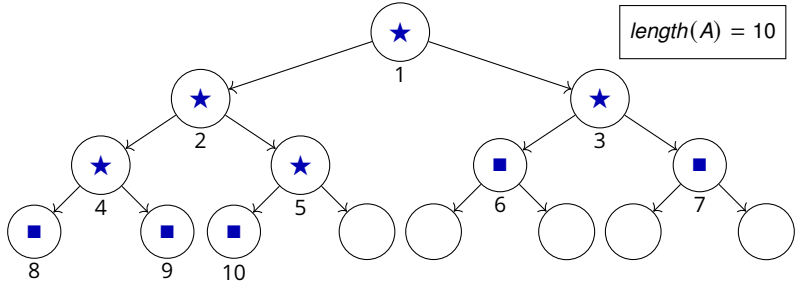
Construcción del Heap

```
Build-Max-Heap(A) 1   A.heap-size = length(A)
                  2   for i = ⌊length(A)/2⌋ downto 1
                  3       Max-Heapify(A, i)
```



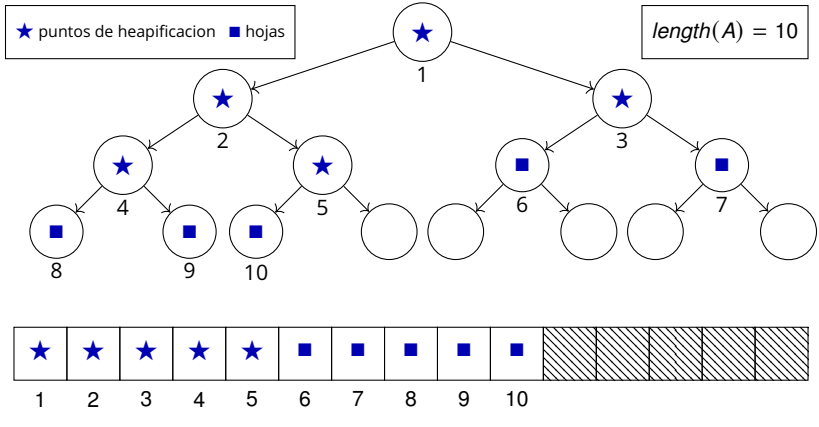
Construcción del Heap

```
Build-Max-Heap(A) 1   A.heap-size = length(A)
                  2   for i = ⌊length(A)/2⌋ downto 1
                  3       Max-Heapify(A, i)
```



Construcción del Heap

Build-Max-Heap(A) 1 $A.heap\text{-}size = length(A)$
2 **for** $i = \lfloor length(A)/2 \rfloor$ **downto** 1
3 **Max-Heapify**(A, i)



Heap Sort

- Idea: podemos utilizar el heap para ordenar el arreglo

```
Heap-Sort(A) 1  Build-Max-Heap(A)
             2  for i = length(A) downto 1
             3      swap (A[i],A[1])
             4      A.heap-size = A.heap-size − 1
             5      Max-Heapify(A, 1)
```

Heap Sort

- Idea: podemos utilizar el heap para ordenar el arreglo

```
Heap-Sort(A) 1  Build-Max-Heap(A)
             2  for i = length(A) downto 1
             3      swap (A[i],A[1])
             4      A.heap-size = A.heap-size − 1
             5      Max-Heapify(A, 1)
```

- ¿Cuál es la complejidad del **Heap-Sort**?

- Idea: podemos utilizar el heap para ordenar el arreglo

```
Heap-Sort(A) 1  Build-Max-Heap(A)
             2  for i = length(A) downto 1
             3      swap (A[i],A[1])
             4      A.heap-size = A.heap-size − 1
             5      Max-Heapify(A, 1)
```

- ¿Cuál es la complejidad del **Heap-Sort**?

$$T(n) = \Theta(n \log n)$$

Heap Sort

- Idea: podemos utilizar el heap para ordenar el arreglo

```
Heap-Sort(A) 1  Build-Max-Heap(A)
             2  for i = length(A) downto 1
             3      swap (A[i],A[1])
             4      A.heap-size = A.heap-size − 1
             5      Max-Heapify(A, 1)
```

- ¿Cuál es la complejidad del **Heap-Sort**?

$$T(n) = \Theta(n \log n)$$

- Beneficios
  - ordenamiento en sitio; peor caso es $\Theta(n \log n)$

Sumario de Algoritmos de Ordenamiento

Sumario de Algoritmos de Ordenamiento

Algoritmo	Complejidad			¿En sitio?
	peor	*promedio*	*mejor*	
Insertion-Sort				

Sumario de Algoritmos de Ordenamiento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort				

Sumario de Algoritmos de Ordenamiento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort				

Sumario de Algoritmos de Ordenamiento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort				

Sumario de Algoritmos de Ordenamiento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no

Sumario de Algoritmos de Ordenamiento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no
Quick-Sort				

Sumario de Algoritmos de Ordenamiento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no
Quick-Sort	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$	sí

Sumario de Algoritmos de Ordenamiento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no
Quick-Sort	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$	sí
Heap-Sort				

Sumario de Algoritmos de Ordenamiento

Algoritmo	**Complejidad**			**¿En sitio?**
	peor	*promedio*	*mejor*	
Insertion-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	sí
Selection-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Bubble-Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	sí
Merge-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	no
Quick-Sort	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$	sí
Heap-Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	sí