3.33pt

Heapsort

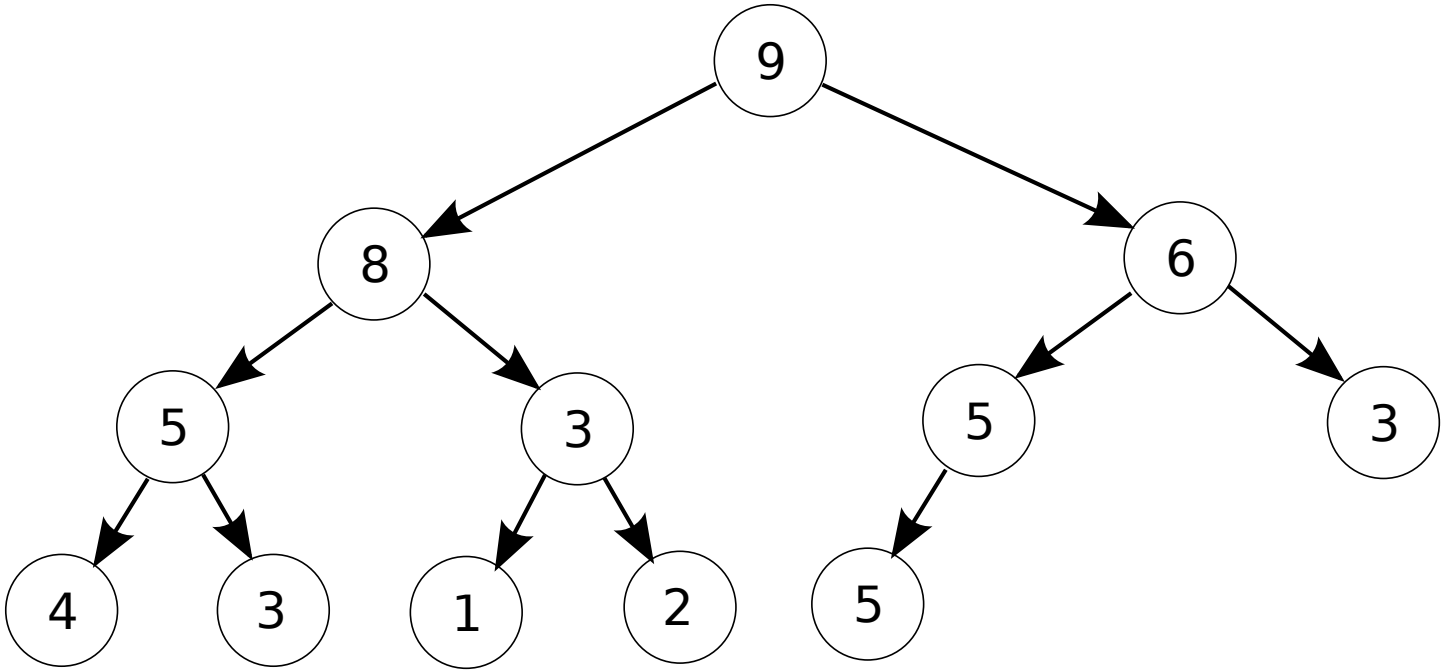
Thiago Martins

PMR2300 / PMR3201
Escola Politécnica da Universidade de São Paulo

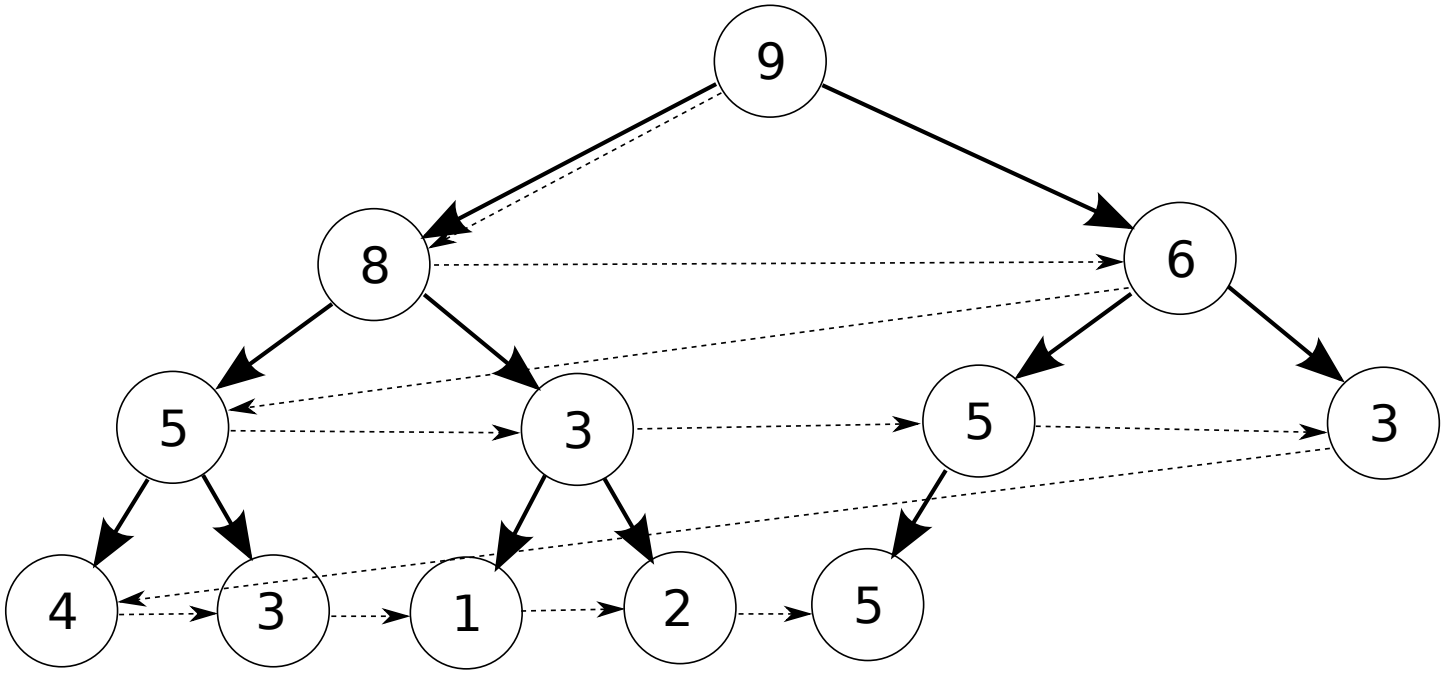
Heap

- Heap binário: árvore binária *completa*: Todos os níveis exceto possivelmente o último estão *cheios*.
- Último nível preenchido *da esquerda para a direita*: Todas as subárvores esquerdas têm altura maior ou igual às subárvores direitas.
- Ordenação: O pai é maior ou igual aos filhos.

Heap



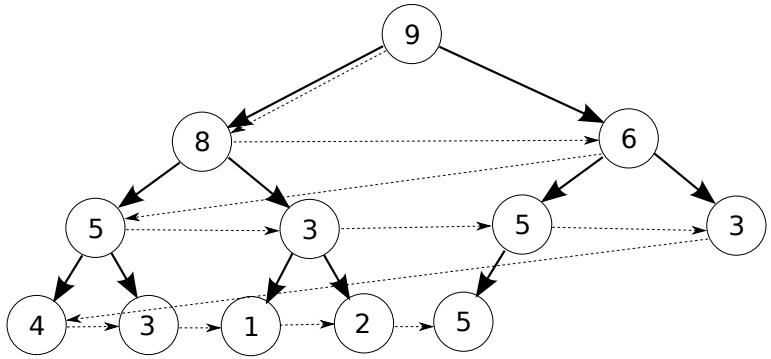
Heap: Armazenamento



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

Heap: Armazenamento

Relação entre nós:



i	0	1	2	3	4	5	6	7	8	9	10	11
	9	8	6	5	3	5	3	4	3	1	2	5

Para índicies baseados em zero (como Java):

- Pai: $(i-1)/2$
- Filho esquerdo: $2i+1$
- Filho direito: $2i+2$

Heap: operações

Operações clássicas sobre um heap:

- Remover o *maior* elemento.
- Adicionar um novo elemento.
- Montar o heap a partir de dados desordenados.

Heap: operações

Remoção do maior elemento:

- Em um heap, o maior elemento é o nó raiz (o primeiro elemento).
- Em um vetor, o único elemento que pode ser removido sem movimentação é o último!

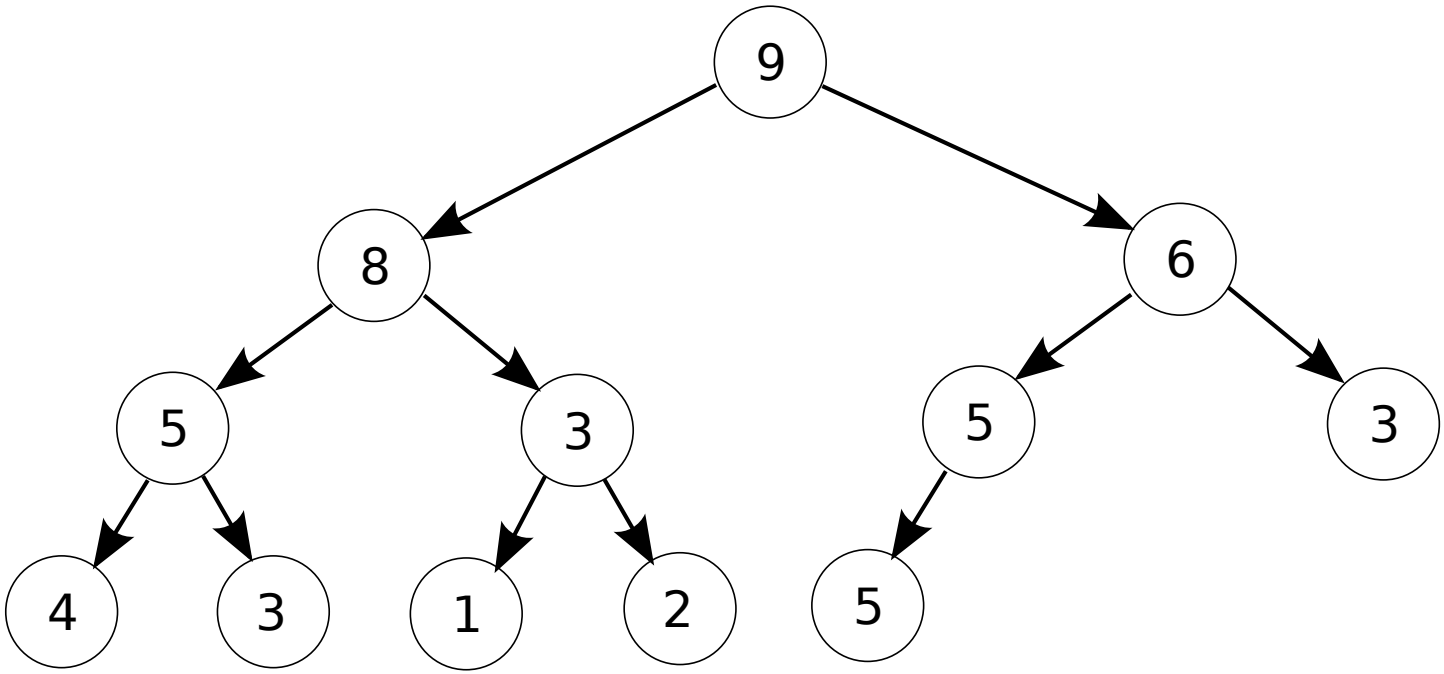
Heap: operações

Remoção do maior elemento: FixHeapDown

- Troque o primeiro e o último elemento de lugar $\Rightarrow$ Heap *inválido* (momentaneamente).
- Remova o último elemento do vetor.
- “Conserte” o heap, começando pelo nó raiz e descendo pela sub-árvore (eventualmente) modificada.

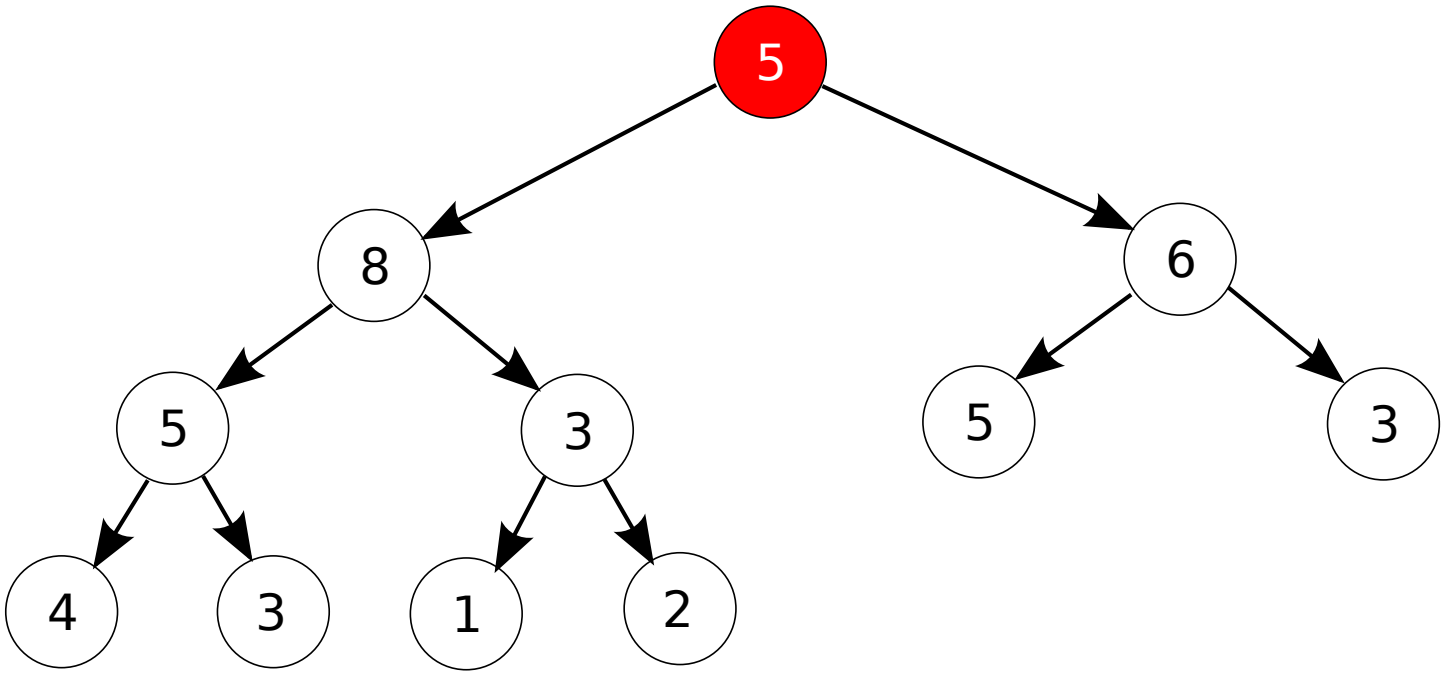
Heap: operações (*FixHeapDown*)

FixHeapDown



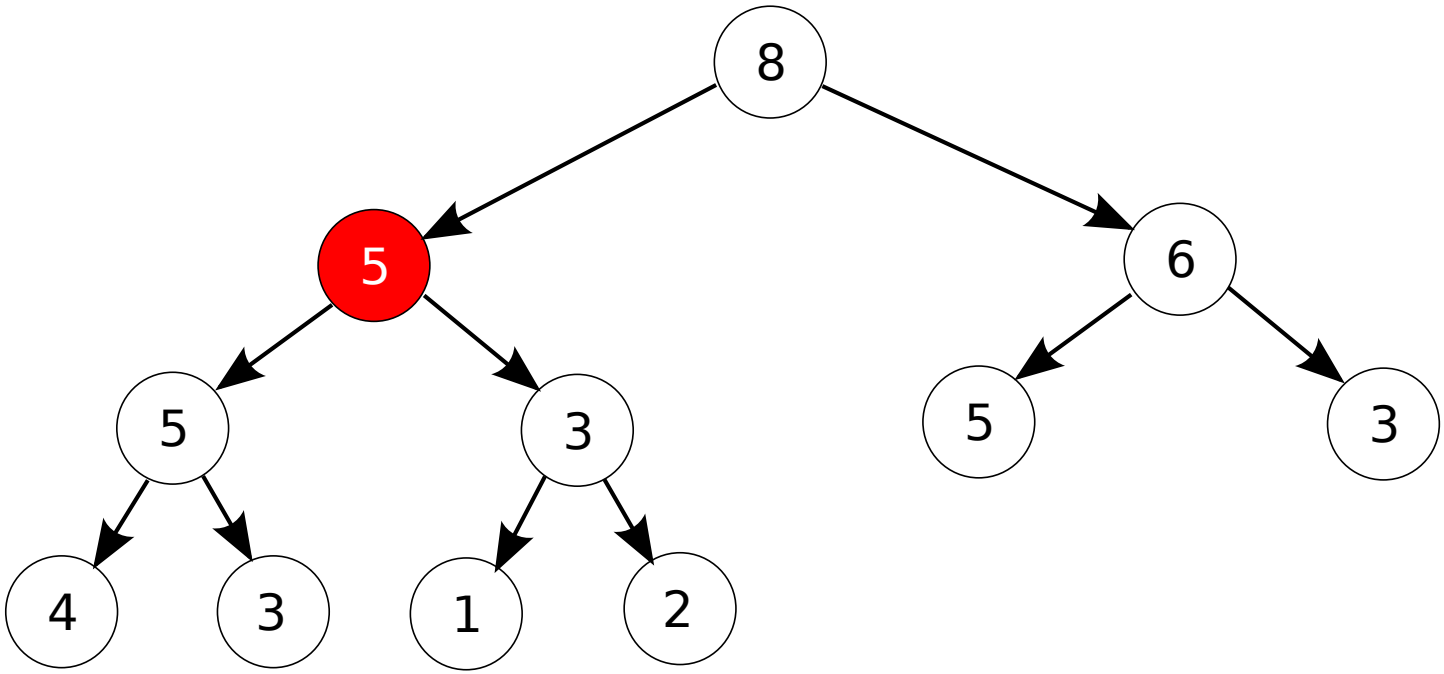
Heap: operações (*FixHeapDown*)

FixHeapDown



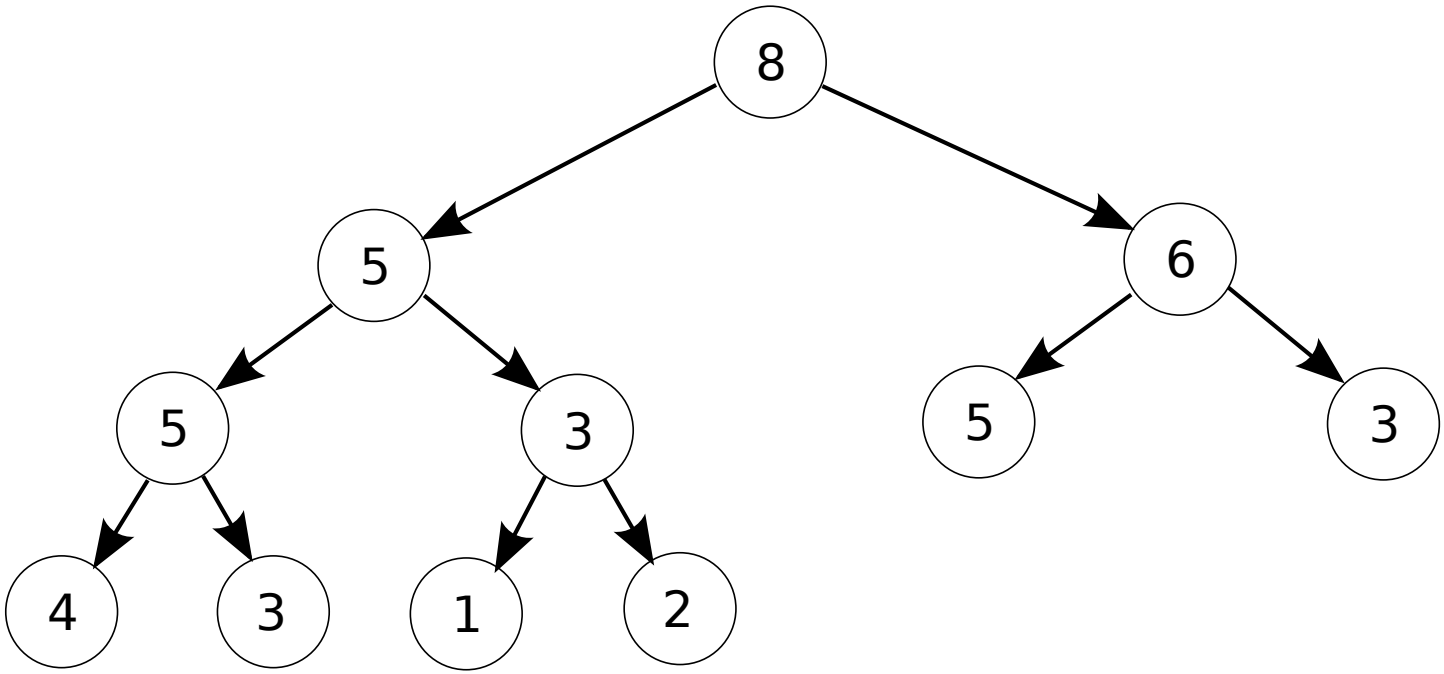
Heap: operações (*FixHeapDown*)

FixHeapDown



Heap: operações (*FixHeapDown*)

FixHeapDown



Heap: operações (*FixHeapDown*)

FixHeapDown

- O heap está correto, *a menos de um nó*, que deve ser movido *para baixo*.
- Considere o nó “problemático” e seus dois filhos imediatos: Escolha o maior nó.
- Troque o nó “problemático” pelo maior nó $\Rightarrow$ Se não há troca a ser feita o algoritmo termina.
- Prossiga o algoritmo pelo nó filho que recebeu o nó pai.

Heap: operações (*FixHeapDown*)

FixHeapDown

- O heap está correto, *a menos de um nó*, que deve ser movido *para baixo*.
- Considere o nó “problemático” e seus dois filhos imediatos: Escolha o maior nó.
- Troque o nó “problemático” pelo maior nó $\Rightarrow$ Se não há troca a ser feita o algoritmo termina.
- Prossiga o algoritmo pelo nó filho que recebeu o nó pai.

Complexidade de FixHeapDown?

Heap: operações (*FixHeapDown*)

FixHeapDown

- O heap está correto, *a menos de um nó*, que deve ser movido *para baixo*.
- Considere o nó "problemático" e seus dois filhos imediatos: Escolha o maior nó.
- Troque o nó "problemático" pelo maior nó $\Rightarrow$ Se não há troca a ser feita o algoritmo termina.
- Prossiga o algoritmo pelo nó filho que recebeu o nó pai.

Complexidade de FixHeapDown?
$\mathcal{O}(H)$ onde H é a altura da árvore.

Heap: operações (*FixHeapDown*)

FixHeapDown

- O heap está correto, *a menos de um nó*, que deve ser movido *para baixo*.
- Considere o nó "problemático" e seus dois filhos imediatos: Escolha o maior nó.
- Troque o nó "problemático" pelo maior nó $\Rightarrow$ Se não há troca a ser feita o algoritmo termina.
- Prossiga o algoritmo pelo nó filho que recebeu o nó pai.

Complexidade de FixHeapDown?
$\mathcal{O}(H)$ onde H é a altura da árvore. A altura da árvore é $\mathcal{O}(\log N)$, onde N é o número de elementos da árvore.

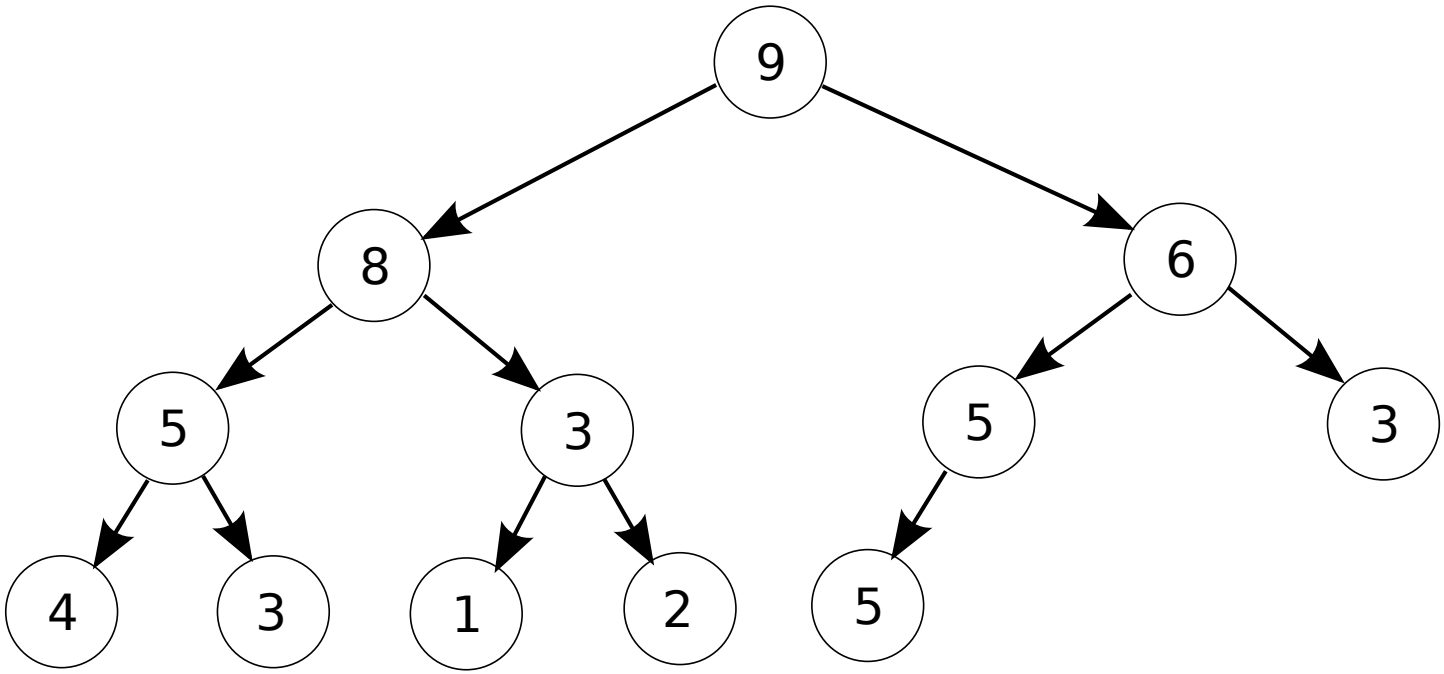
Heap: operações

Adicionar um novo elemento:

- Novamente, em um vetor, a única posição em que se pode facilmente adicionar um elemento é a última.
- Mas em um heap, a posição final depende do valor do elemento!

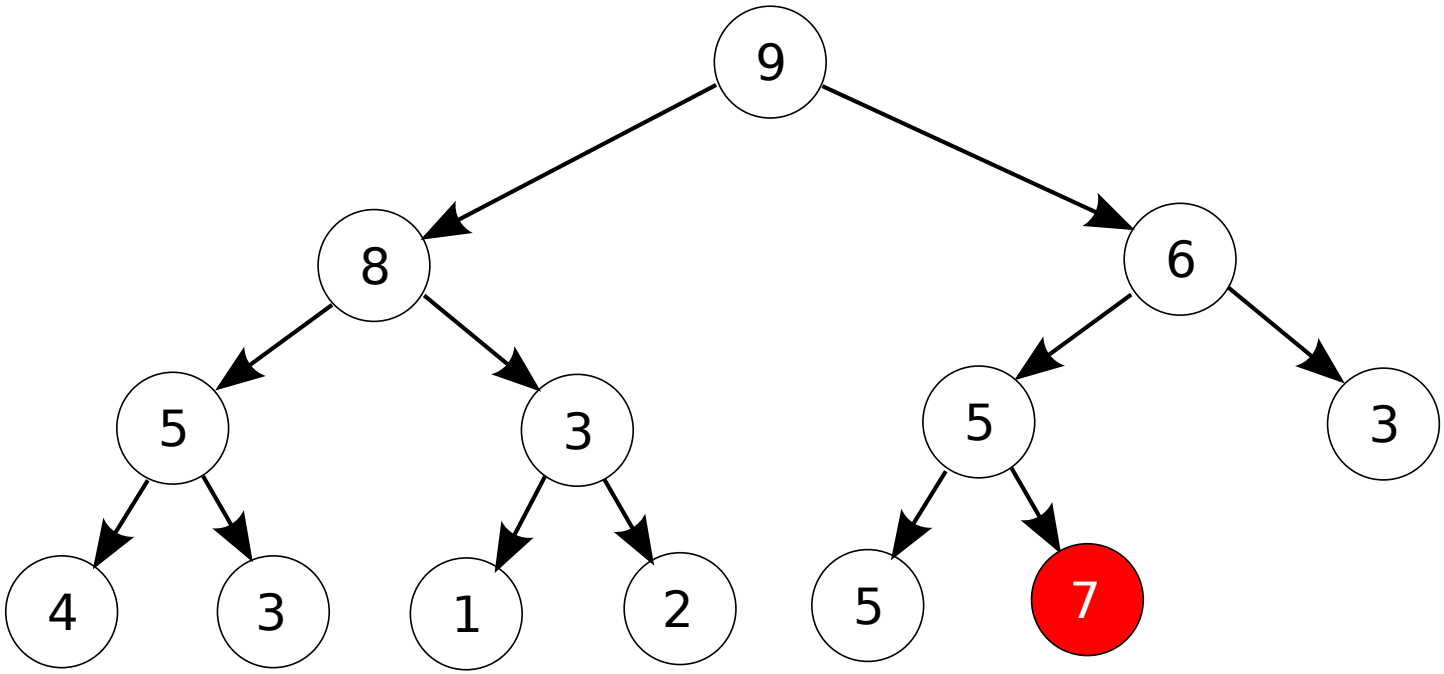
Heap: operações (*FixHeapUp*)

FixHeapUp



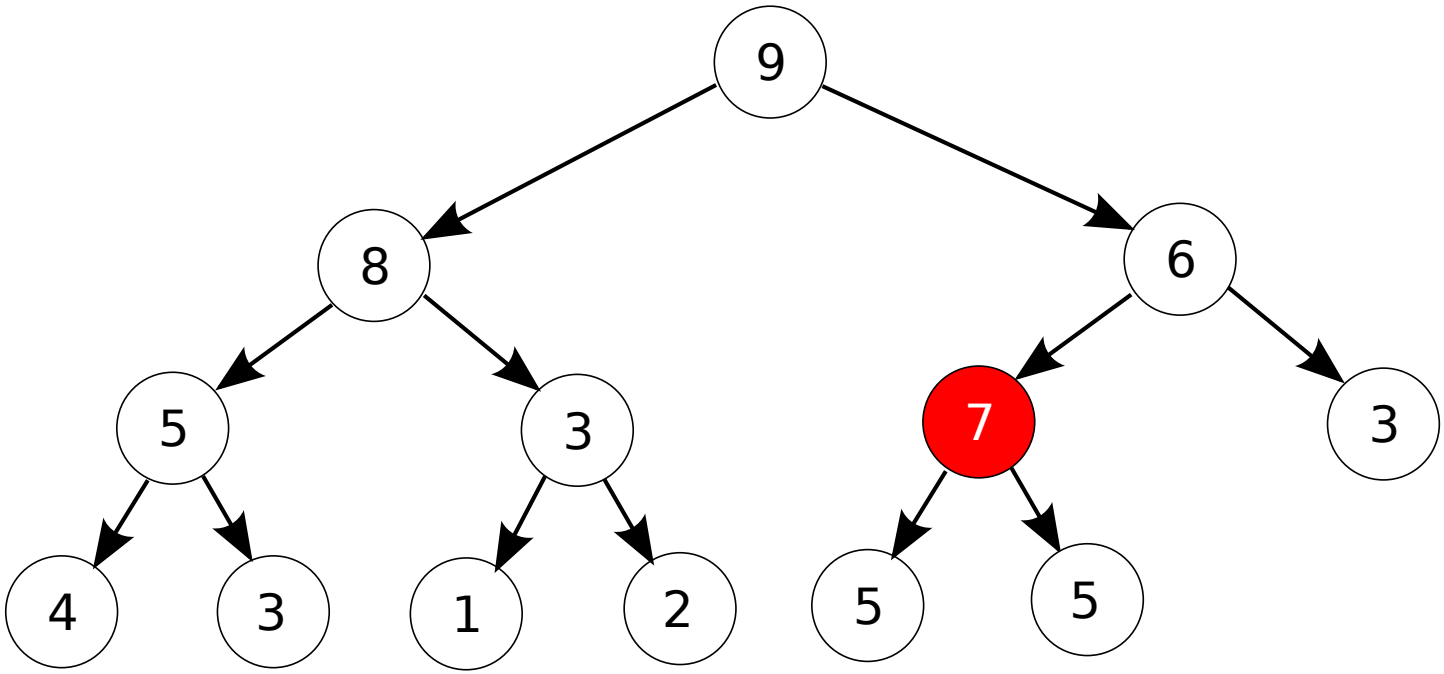
Heap: operações (*FixHeapUp*)

FixHeapUp



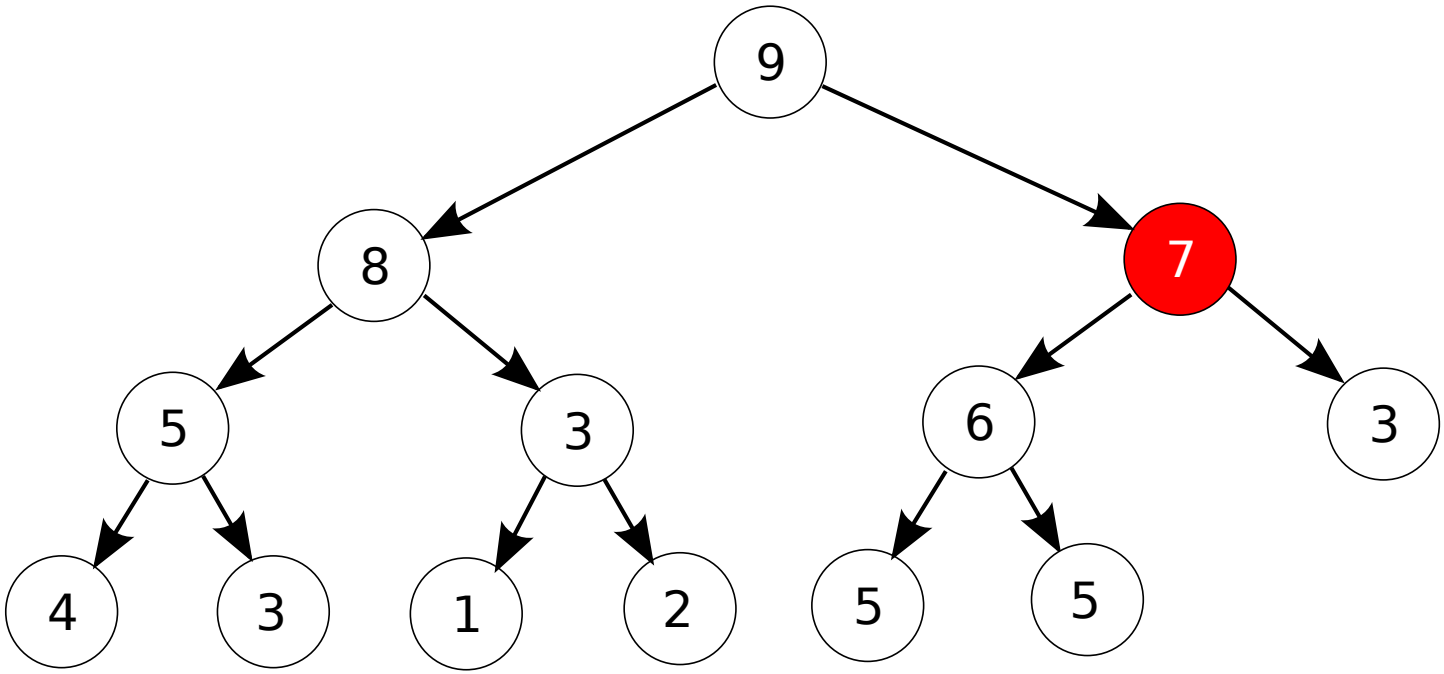
Heap: operações (*FixHeapUp*)

FixHeapUp



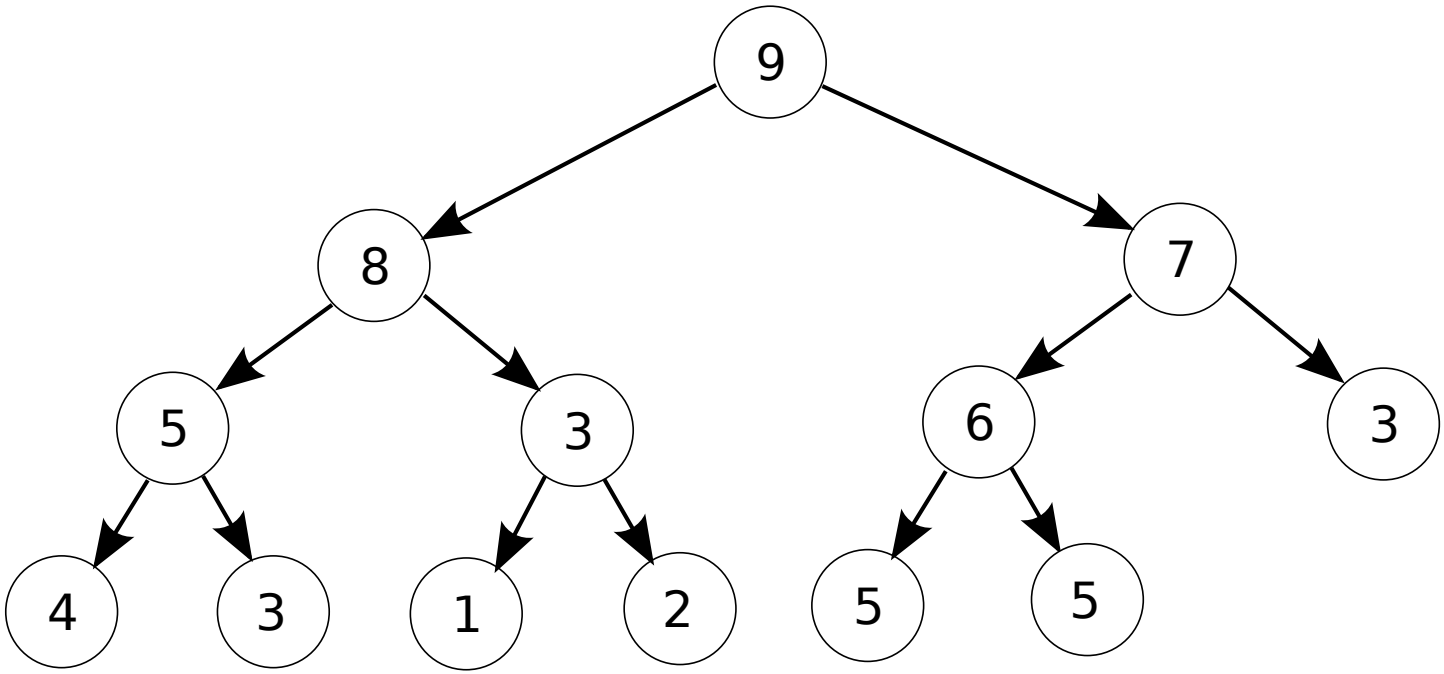
Heap: operações (*FixHeapUp*)

FixHeapUp



Heap: operações (*FixHeapUp*)

FixHeapUp



Heap: operações (*FixHeapUp*)

FixHeapUp

- O heap está correto, *a menos de um nó*, que deve ser movido *para cima*.
- Considere o nó “problemático” e o seu pai.
- Caso o nó “problemático” seja maior que o pai, troque-os de posição.
- Prossiga o algoritmo pelo nó pai.

Heap: operações (*FixHeapUp*)

FixHeapUp

- O heap está correto, *a menos de um nó*, que deve ser movido *para cima*.
- Considere o nó “problemático” e o seu pai.
- Caso o nó “problemático” seja maior que o pai, troque-os de posição.
- Prossiga o algoritmo pelo nó pai.

Complexidade de FixHeapUp?

Heap: operações (*FixHeapUp*)

FixHeapUp

- O heap está correto, *a menos de um nó*, que deve ser movido *para cima*.
- Considere o nó "problemático" e o seu pai.
- Caso o nó "problemático" seja maior que o pai, troque-os de posição.
- Prossiga o algoritmo pelo nó pai.

Complexidade de FixHeapUp?
$\mathcal{O}(H)$ onde H é a profundidade do elemento problemático.

Heap: operações (*FixHeapUp*)

FixHeapUp

- O heap está correto, *a menos de um nó*, que deve ser movido *para cima*.
- Considere o nó "problemático" e o seu pai.
- Caso o nó "problemático" seja maior que o pai, troque-os de posição.
- Prossiga o algoritmo pelo nó pai.

Complexidade de FixHeapUp?
$\mathcal{O}(H)$ onde H é a profundidade do elemento problemático. A profundidade do último elemento é $\mathcal{O}(\log N)$, onde N é o número de elementos da árvore.

Heap: operações

Montar o heap a partir de dados desordenados:
Uma possibilidade é adicionar dados a um heap “da esquerda para a direita”, com FixHeapUp.

Heap: operações

Montar o heap a partir de dados desordenados:
Uma possibilidade é adicionar dados a um heap “da esquerda para a direita”, com FixHeapUp.
Complexidade?

Heap: operações

Montar o heap a partir de dados desordenados:
Uma possibilidade é adicionar dados a um heap “da esquerda para a direita”, com FixHeapUp.
Complexidade?

$$\sum_{n} \log n = \mathcal{O}(n \log n)$$

Heap: operações

Montar o heap a partir de dados desordenados:
Mas e se considerarmos o vetor como um heap *inteiramente* quebrado, e consertarmos “de baixo para cima”, com FixHeapDown?

Heap: operações

Montar o heap a partir de dados desordenados:
Mas e se considerarmos o vetor como um heap *inteiramente* quebrado, e consertarmos “de baixo para cima”, com FixHeapDown?
Complexidade?

Heap: operações

Montar o heap a partir de dados desordenados:
Mas e se considerarmos o vetor como um heap *inteiramente* quebrado, e consertarmos “de baixo para cima”, com FixHeapDown?
Complexidade?
Cada chamada de FixHeapDown tem complexidade $(O)(H)$

Heap: operações

Montar o heap a partir de dados desordenados:
Mas e se considerarmos o vetor como um heap *inteiramente* quebrado, e consertarmos “de baixo para cima”, com FixHeapDown?
Complexidade?
Cada chamada de FixHeapDown tem complexidade $(O)(H)$
Cada chamada de FixHeapDown tem complexidade $(O)(H)$
No nível h da árvore, há $O(N/2^h)$ elementos, onde N é o número total de elementos da árvore.

Heap: operações

Montar o heap a partir de dados desordenados:
Mas e se considerarmos o vetor como um heap *inteiramente* quebrado, e consertarmos “de baixo para cima”, com FixHeapDown?
Complexidade?
Cada chamada de FixHeapDown tem complexidade $(O)(H)$
Cada chamada de FixHeapDown tem complexidade $(O)(H)$
No nível h da árvore, há $O(N/2^h)$ elementos, onde N é o número total de elementos da árvore.

$$\mathcal{O}\left(\sum_{h=0}^{\log N} \frac{hN}{2^h}\right)$$

Heap: operações

Montar o heap a partir de dados desordenados:
Mas e se considerarmos o vetor como um heap *inteiramente* quebrado, e consertarmos “de baixo para cima”, com FixHeapDown?
Complexidade?
Cada chamada de FixHeapDown tem complexidade $(O)(H)$
Cada chamada de FixHeapDown tem complexidade $(O)(H)$
No nível h da árvore, há $O(N/2^h)$ elementos, onde N é o número total de elementos da árvore.

$$\mathcal{O}\left(N\sum_{h=0}^{\log N}\frac{h}{2^h}\right) = \mathcal{O}(N)$$

Usar FixHeapDown é (surpreendentemente) mais rápido!

Heapsort

- Monte o heap a partir de dados desordenados com FixHeapDown.
- Remova (e posicione ao final) sucessivamente o maior elemento do heap até que o mesmo esteja vazio.
- O resultado é o vetor ordenado em ordem crescente!.

Heapsort

- Monte o heap a partir de dados desordenados com FixHeapDown.
- Remova (e posicione ao final) sucessivamente o maior elemento do heap até que o mesmo esteja vazio.
- O resultado é o vetor ordenado em ordem crescente!.

Complexidade?

Heapsort

- Monte o heap a partir de dados desordenados com FixHeapDown.
- Remova (e posicione ao final) sucessivamente o maior elemento do heap até que o mesmo esteja vazio.
- O resultado é o vetor ordenado em ordem crescente!.

Complexidade?
$\mathcal{O}(N)$ para a montagem do heap, $\mathcal{O}(N \log N)$ para a desmontagem.

Heapsort

- Monte o heap a partir de dados desordenados com FixHeapDown.
- Remova (e posicione ao final) sucessivamente o maior elemento do heap até que o mesmo esteja vazio.
- O resultado é o vetor ordenado em ordem crescente!.

Complexidade?
$\mathcal{O}(N)$ para a montagem do heap, $\mathcal{O}(N \log N)$ para a desmontagem.
$\mathcal{O}(N \log N)$, *sem* uso de armazenamento auxiliar!