



Aula 3 - Heaps, filas de prioridades

PrepTech Google

Roteiro da Aula

1 - Problema

Possível cenário cotidiano

2 - Dificuldades do Problema


Reflexões sobre como resolver

3 - Solução

Estado da arte para o problema

4 - Prática

Questões do LeetCode



1 - Problema

Inserindo e removendo elementos de uma estrutura de acordo com certa **prioridade**

Fila de Caminhões



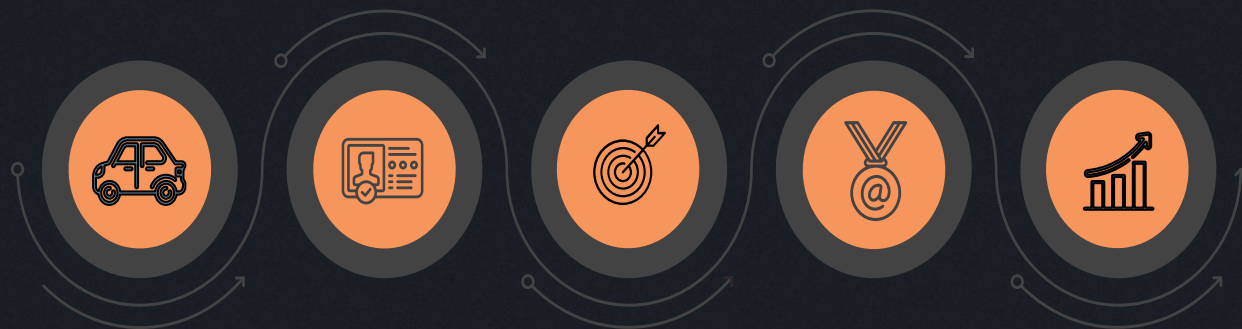
Foto: Rodrigo Leal

PONTUAÇÃO

Cada caminhão recebe uma **pontuação** (inteiro), que depende do tipo de produto transportado

SAÍDA DOS CAMINHÕES

Depois de atendido, o caminhão vai embora e sai do pátio



CHEGADA DOS CAMINHÕES

Quando os caminhões estacionam no pátio, entram em uma **fila virtual**

ATENDENDO OS CAMINHÕES

O caminhão no pátio com a **maior pontuação** é o próximo a ser atendido

RITMO INTENSO

Caminhões chegam e saem a cada minuto

Fila de Caminhões

Para representar a fila virtual, precisamos de duas primitivas centrais:

- **INSERIR** novo caminhão no pátio com sua pontuação
- **REMOVER** o caminhão de maior pontuação do pátio



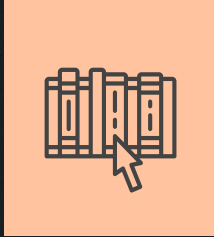
Interface Heap

```
//classe virtual para representar o Heap
class Heap {
public:
    virtual void insert(int val)=0;
    virtual int remove()=0;
};
```

Cenário do Pátio

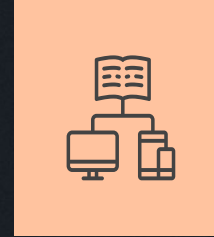
PRIMITIVA	PÁTIO - DEPOIS	CAMINHÃO ATENDIDO
ESTADO INICIAL	[12, 5, 76, 2, 48]	-
REMOVER	[12, 5, 2, 48]	Saiu com 76 pontos
INSERIR 34	[12, 5, 2, 48, 34]	Chegou com 34 pontos
REMOVER	[12, 5, 2, 34]	Saiu com 48 pontos
REMOVER	[12, 5, 2]	Saiu com 34 pontos
REMOVER	[5, 2]	Saiu com 12 pontos
INSERIR 8	[5, 2, 8]	Chegou com 8 pontos
REMOVER	[5, 2]	Saiu com 8 pontos

Estratégia 1 - Usando vetores sem ordenação



INSERIR

Basta acrescentar um novo elemento ao final do vetor



REMOVER

- 1) Percorre todo o vetor para procurar o maior elemento e marca sua posição.
- 2) Percorre todo o vetor empurrando uma posição para esquerda todos à direita do maior elemento.
- 3) Remove o último elemento do vetor.

Cenário do Pátio - inserindo ao final



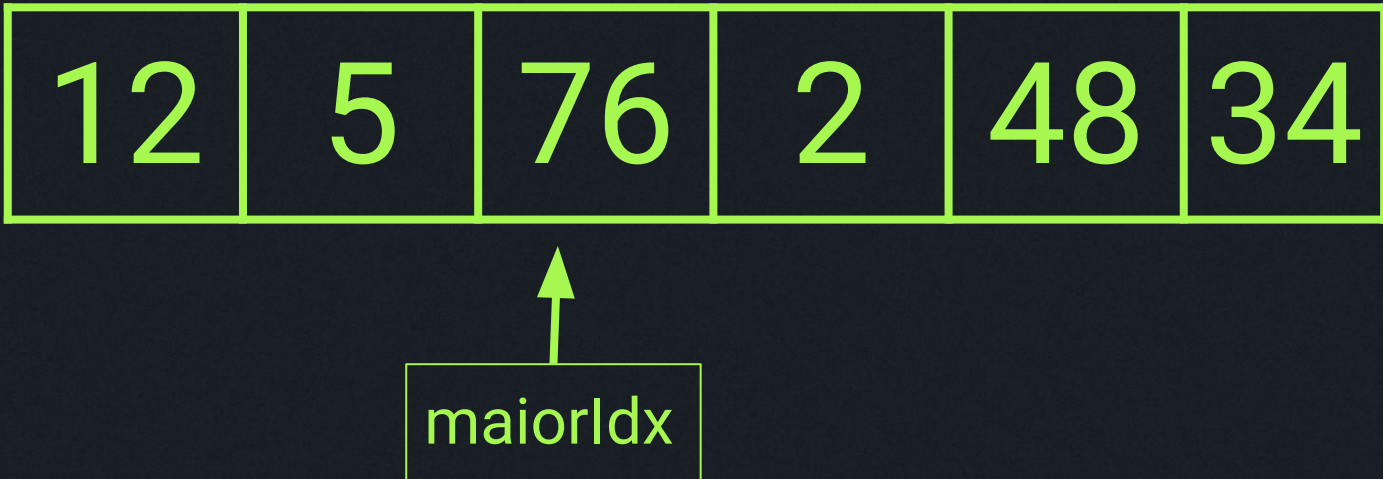
+



HeapAsVector::insert

```
class HeapAsVector : public Heap {  
    private:  
        vector<int> data;  
  
    public:  
        void insert(int val) {  
            data.push_back(val);  
        }  
    ...  
}
```


Passo 1: identificar o índice do maior elemento



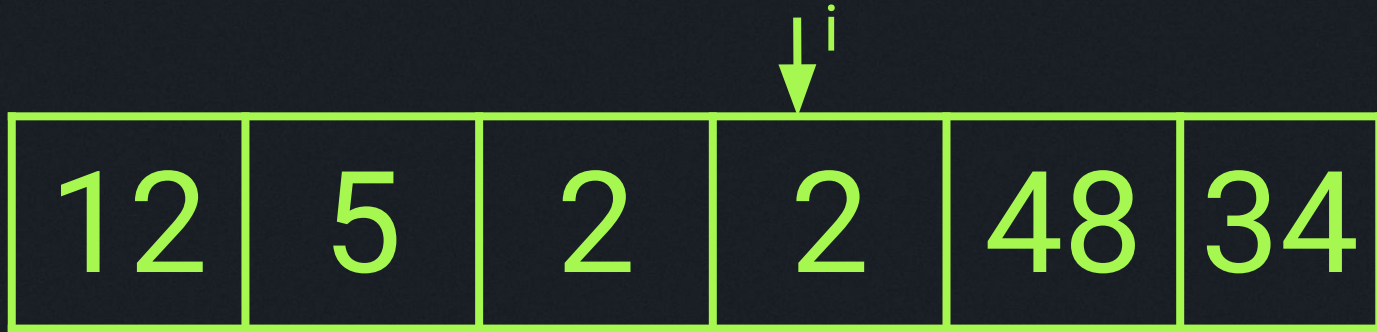
Passo 2: mover uma posição para esquerda do maior



$\text{data}[i] = \text{data}[i+1]$



Passo 2: mover uma posição para esquerda do maior



$\text{data}[i] = \text{data}[i+1]$



Passo 2: mover uma posição para esquerda do maior



$\text{data}[i] = \text{data}[i+1]$



Passo 3: remover último elemento

12	5	2	48	34	34
----	---	---	----	----	----

```
data.pop_back()
```

12	5	2	48	34
----	---	---	----	----

HeapAsVector::remove

```
int HeapAsVector::remove() {
```

```
    int maxIdx = 0;
```

```
    for(int i=1; i<data.size(); i++) {
```

```
        if( data[i]>data[maxIdx])
```

```
            maxIdx = i;
```

```
    }
```

Passo 1: identificar o índice do maior elemento

```
    int ret = data[maxIdx];
```

```
    for(int i=maxIdx; i+1<data.size(); i++)
```

```
        data[i] = data[i+1];
```

```
    }
```

Passo 2: mover uma posição para esquerda do maior

```
    data.pop_back();
```

```
    return ret;
```

```
}
```

Passo 3: remover último elemento

Testando as estratégias com 100 mil inserções ou remoções aleatórias

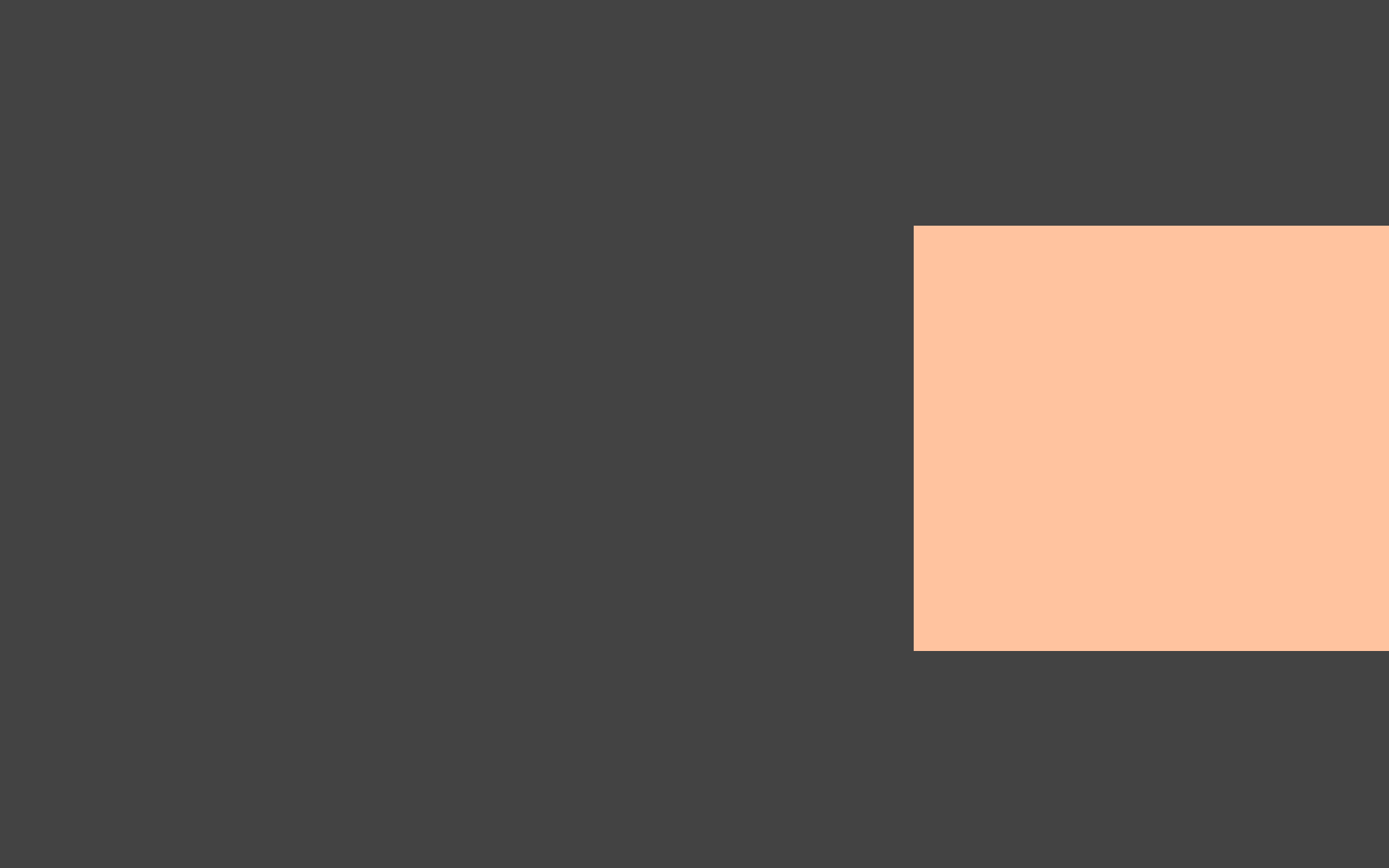
```
double testStrategy(Heap *myHeap) {  
    clock_t begin = clock();  
    int szHeap = 0;  
    srand( -time(NULL) );  
    for( int i=0; i<100000; i++) {  
        if( szHeap==0 || rand()%3 ) { // insere se o heap estiver vazio ou 2/3 das vezes  
            myHeap->insert( rand()%50000 );  
            szHeap++;  
        }  
        else { // 1/3 das vezes remove (quando dá 0 no rand()), se o heap não estiver vazio  
            myHeap->remove();  
            szHeap--;  
        }  
    }  
    clock_t end = clock();  
    double processingTime = double(end - begin) / CLOCKS_PER_SEC;  
    return processingTime;  
}
```



Estratégia 1

Heap como um vetor
não-ordenado

0.23s



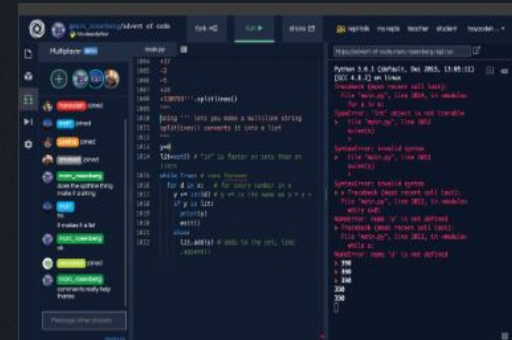
2 - Dificuldades do Problema

Reflexões sobre essa solução proposta e como melhorar

Solução escalável?

Estratégia 1 Heap como vetor sem ordenação	
No. Operações	Tempo (segundos)
1K	0.00065595
10K	0.0212126
100K	1.85968
1M	184.859
10M	

Código-fonte:

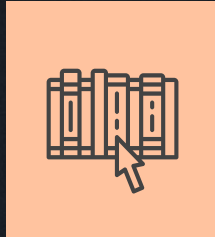


```
def HeapSort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        _sift(arr, n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        _sift(arr, i, 0)
    return arr

def _sift(arr, n, i):
    while True:
        left = 2 * i + 1
        right = 2 * i + 2
        if left > n:
            break
        if right > n:
            if arr[i] < arr[left]:
                arr[i], arr[left] = arr[left], arr[i]
            break
        if arr[i] < arr[left] and arr[i] < arr[right]:
            arr[i], arr[left] = arr[left], arr[i]
        elif arr[i] < arr[right]:
            arr[i], arr[right] = arr[right], arr[i]
        i = left if left < right else right

if __name__ == '__main__':
    n = int(input())
    arr = list(map(int, input().split()))
    sorted_arr = HeapSort(arr)
    print(*sorted_arr)
```

Estratégia 2 - Usando vetores com ordenação

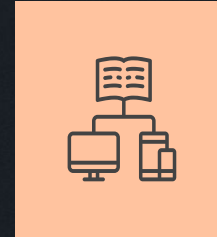


INSERIR

- 1) Inserir o elemento ao final do vetor
- 2) Ordenar o vetor

OU

- 1) Busca binária para posição
- 2) Move todos para direita, para criar espaço



REMOVER

- 1) Retira o último elemento do vetor, que será o maior

Com vetor ordenado, maior está ao final

2	5	12	34	48	76
---	---	----	----	----	----

`data.pop_back()`

2	5	12	34	48
---	---	----	----	----

76



DIFICULDADE

COMO MANTER O VETOR
ORDENADO APÓS
INSERIR UM NOVO
ELEMENTO?

HeapVectorSort

```
class HeapVectorSort : public Heap {  
    private:  
        vector<int> data;  
    public:  
        void insert(int val) {  
            data.push_back(val);  
            sort(data.begin(), data.end());  
        }  
        int remove(){  
            if(data.size()==0) return -1; //não pode remover de heap vazio  
            int ret = data.back();  
            data.pop_back();  
            return ret;  
        }  
};
```

← Insere ao final e ordena o vetor inteiro depois



Estratégia 2

Heap ordenando o vetor

27 s

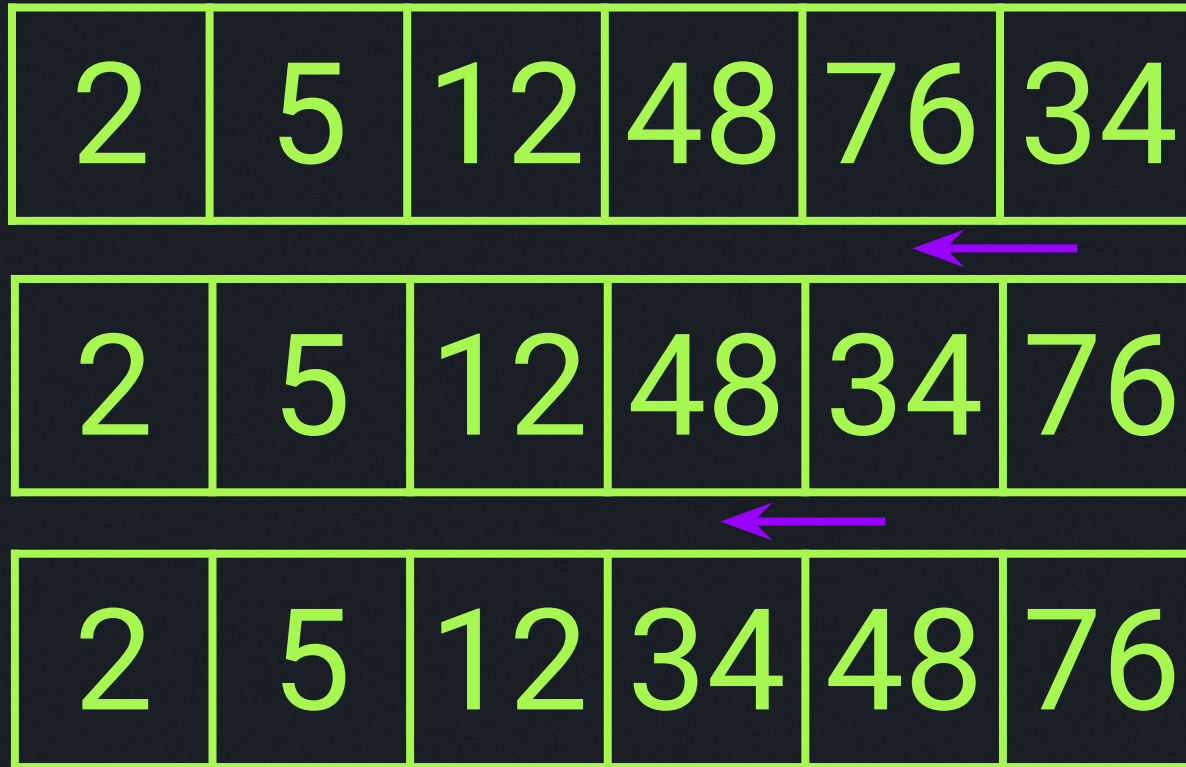
Passo 1: Acrescenta novo elemento ao final



`data.push_back(34)`



Passo 2: Move para esquerda, até achar posição ordenada



HeapVectorPosition

```
void HeapVectorPosition::insert(int val) {  
    data.push_back(val);
```

Passo 1: Acrescenta novo
elemento ao final

```
    for(int i=data.size()-2; i>=0; i--) {  
        if(data[i]>val) {  
            data[i+1] = data[i];  
            data[i] = val;  
        }  
        else  
            break;  
    }  
}
```

Passo 2: Move para esquerda, até
achar posição ordenada



Estratégia 3

Heap movendo na inserção, para manter ordenado

0.19s

3 - Estado da arte

Qual a melhor forma de resolver esse problema?

Fila de Prioridade



HeapPQueue

```
class HeapPQueue : public Heap {  
private:  
    priority_queue<int> q;  
public:  
    void insert(int val) {  
        q.push(val);  
    }  
    int remove() {  
        if (q.empty()) return -1; //não pode remover de heap vazio  
        int ret = q.top();  
        q.pop();  
        return ret;  
    }  
};
```



Estratégia 4

priority_queue da STL

~0.005s

Comparativo

Heap como vetor não ordenado	0.23 s
Heap ordenando o vetor com Sort	27 s
Heap movendo na inserção para manter ordenado	0.19 s
STL Priority Queue	0.005 s

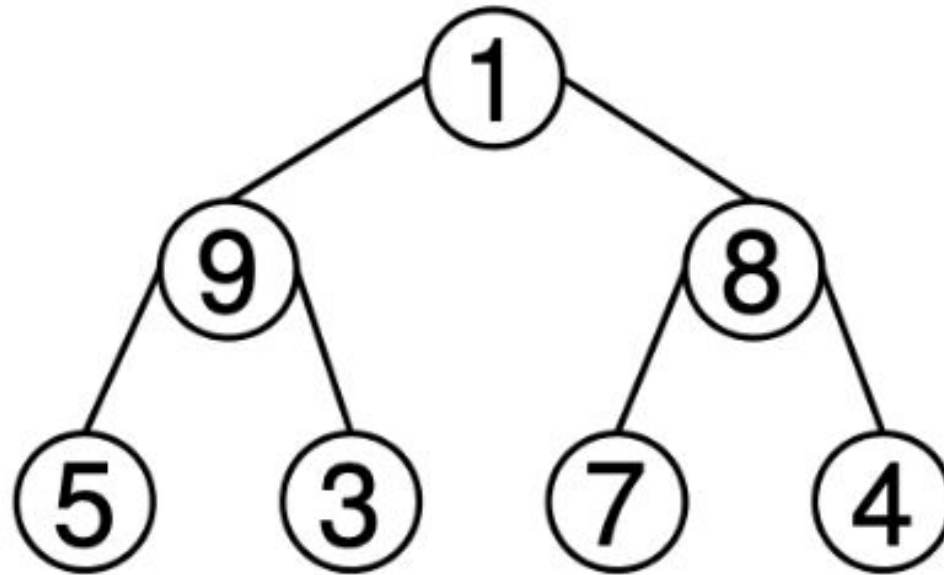
Como o HEAP funciona?

Por que é
mais
rápido?

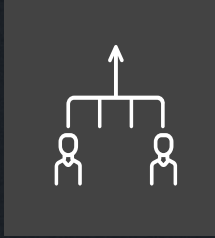


Vetores como Árvores Binárias Completas

1	9	8	5	3	7	4
---	---	---	---	---	---	---

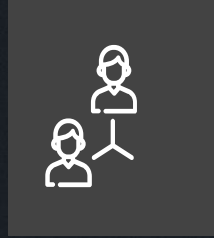


Primitivas para o Vetor como Árvore



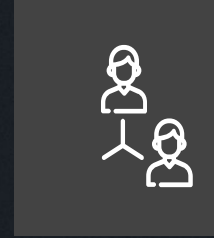
PARENT(idx)

`return (idx-1)/2;`



LEFT(idx)

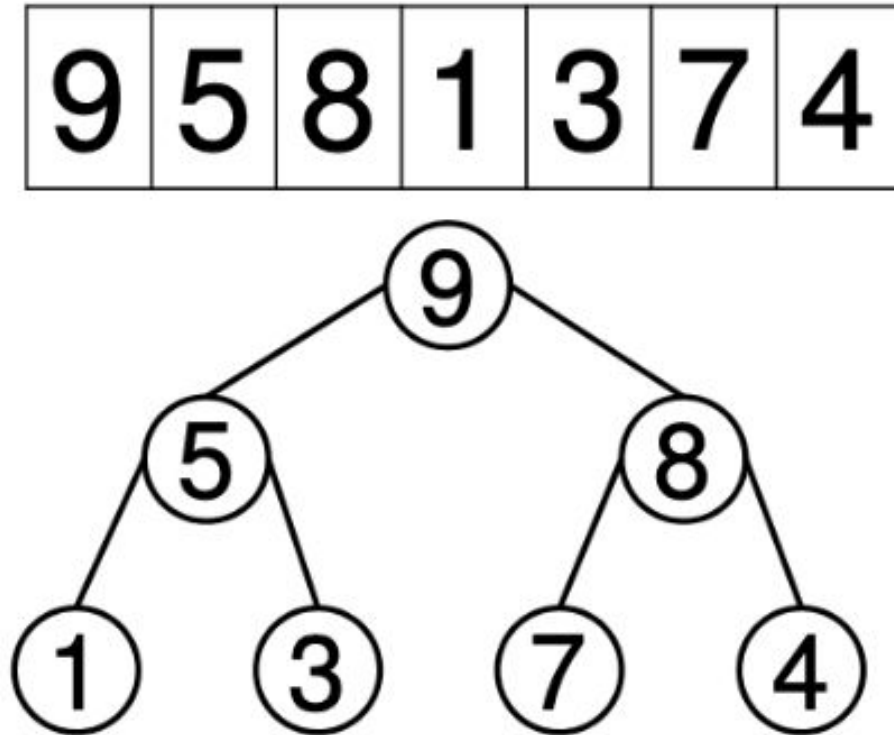
`return 2*idx+1;`



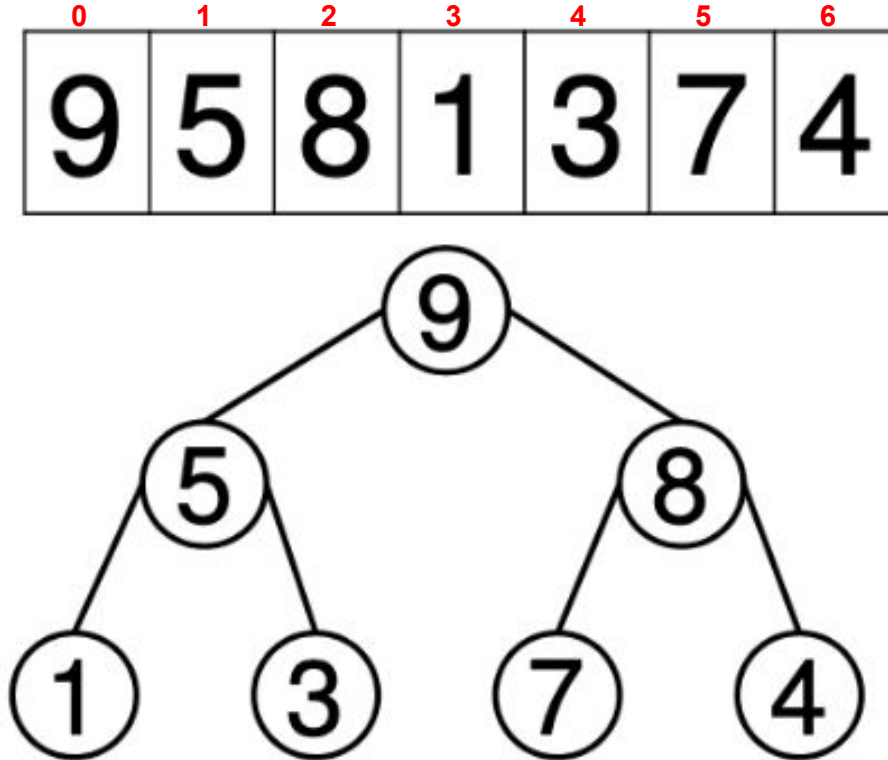
RIGHT(idx)

`return 2*idx+2;`

Todo nó pai é maior ou igual o nó filho



Acessando nós como árvore



PARENT(idx) return $(idx-1)/2$:

$$\text{Parent}(1) = (1-1)/2 = 0$$

$$\text{Parent}(2) = (2-1)/2 = 0$$

LEFT(idx) return $2*idx+1$:

$$\text{Left}(1) = (2*1)+1 = 3$$

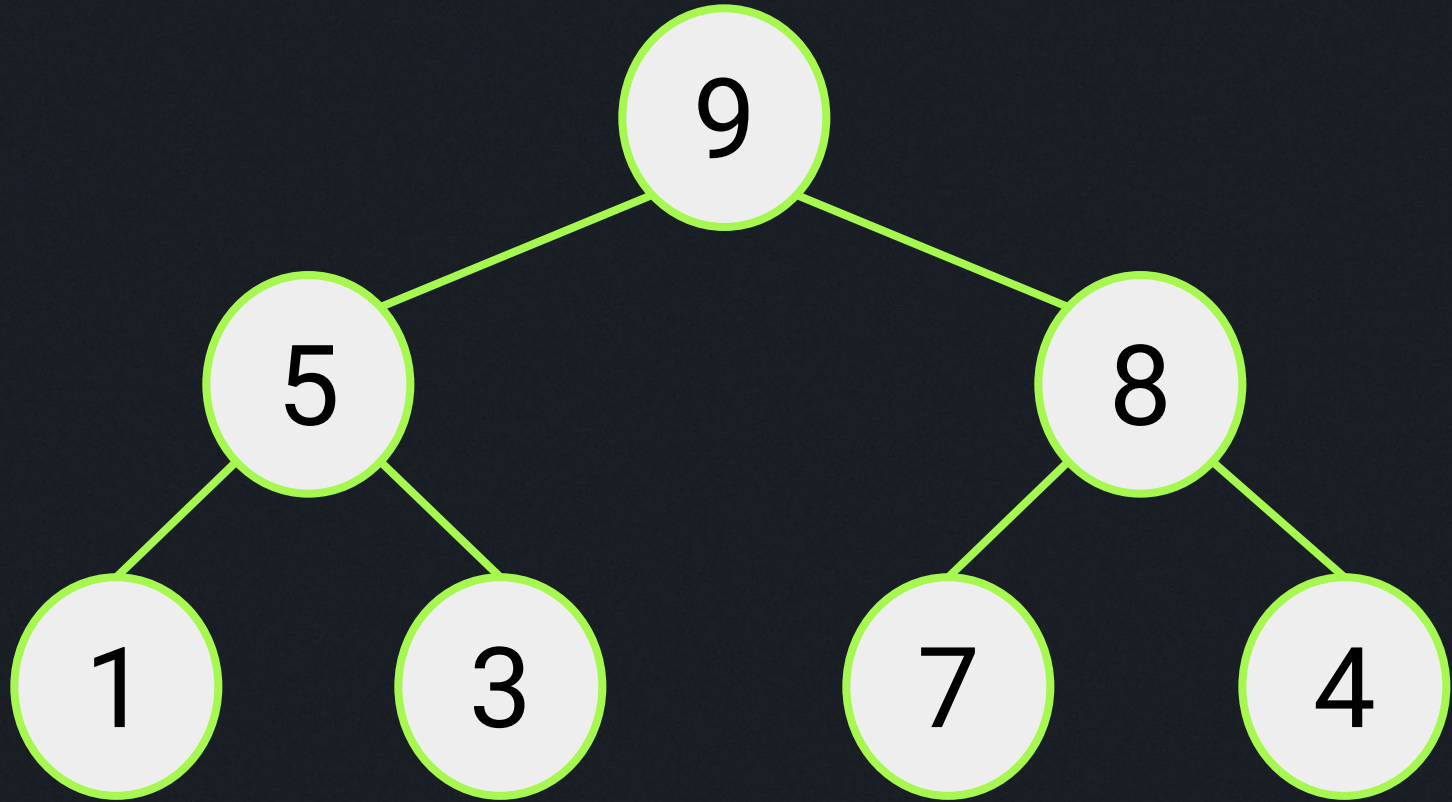
$$\text{Left}(2) = (2*2)+1 = 5$$

RIGHT(idx) return $2*idx+2$:

$$\text{Right}(1) = (2*1)+2 = 4$$

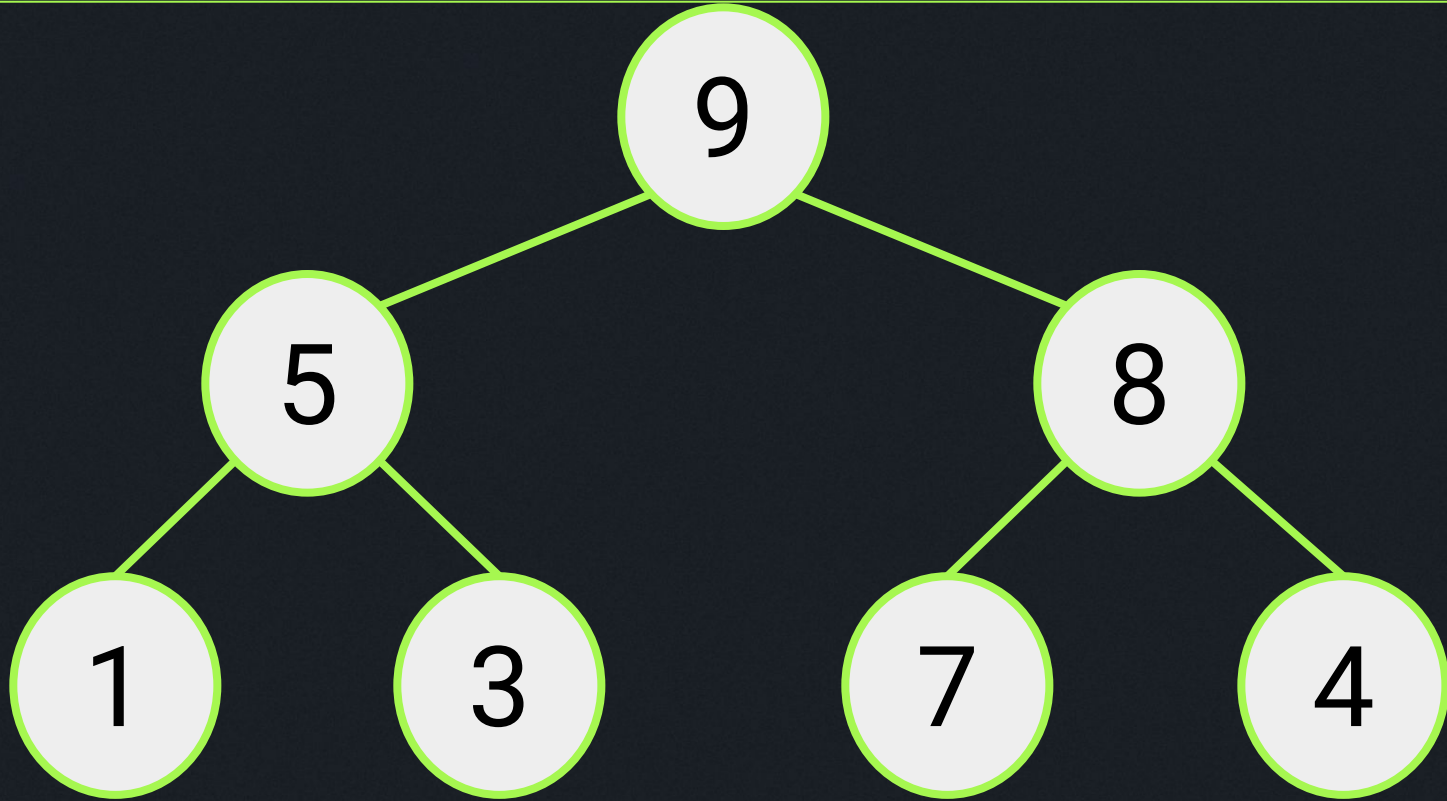
$$\text{Right}(2) = (2*2)+2 = 6$$

Como remover?



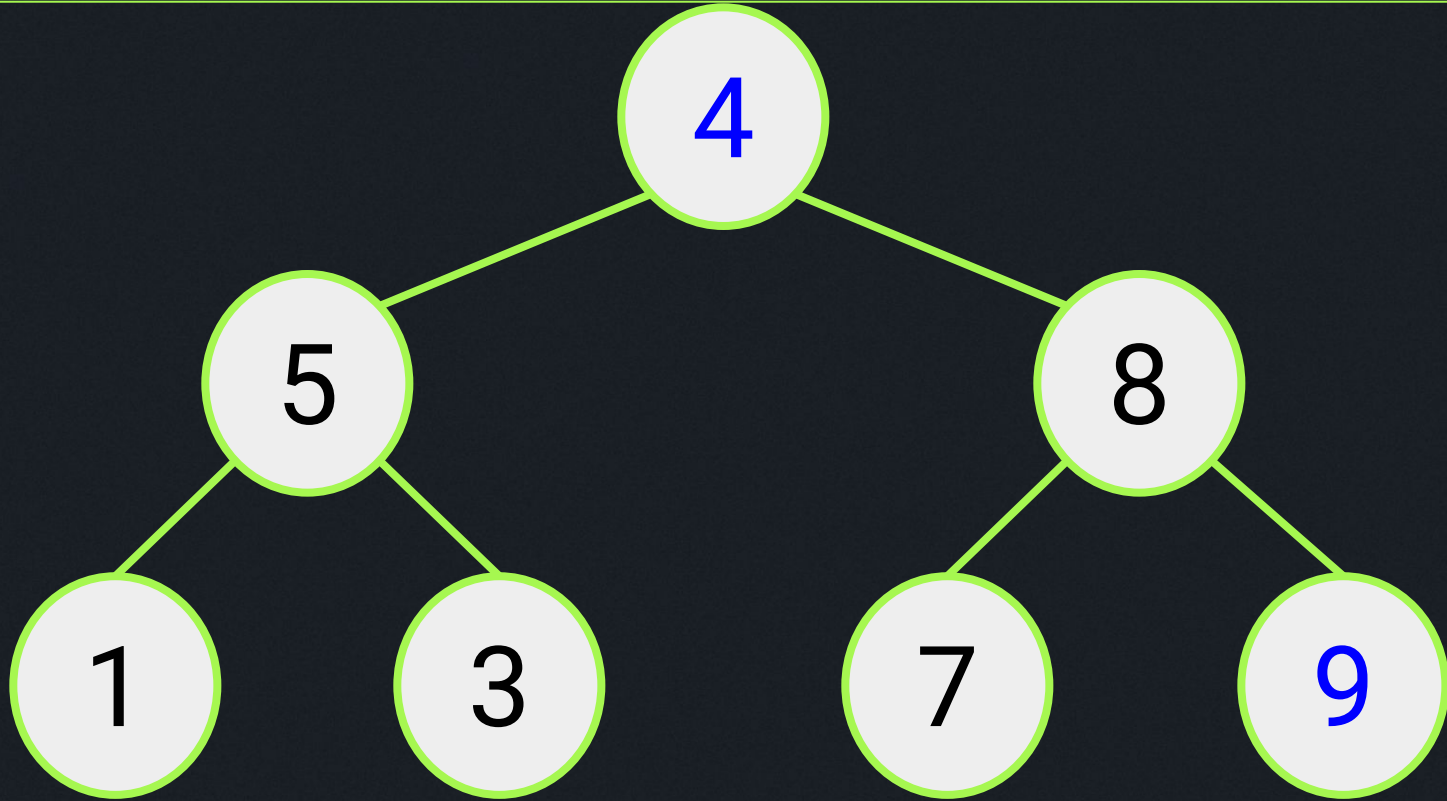
Removendo

Passo 1 - trocar raiz com último



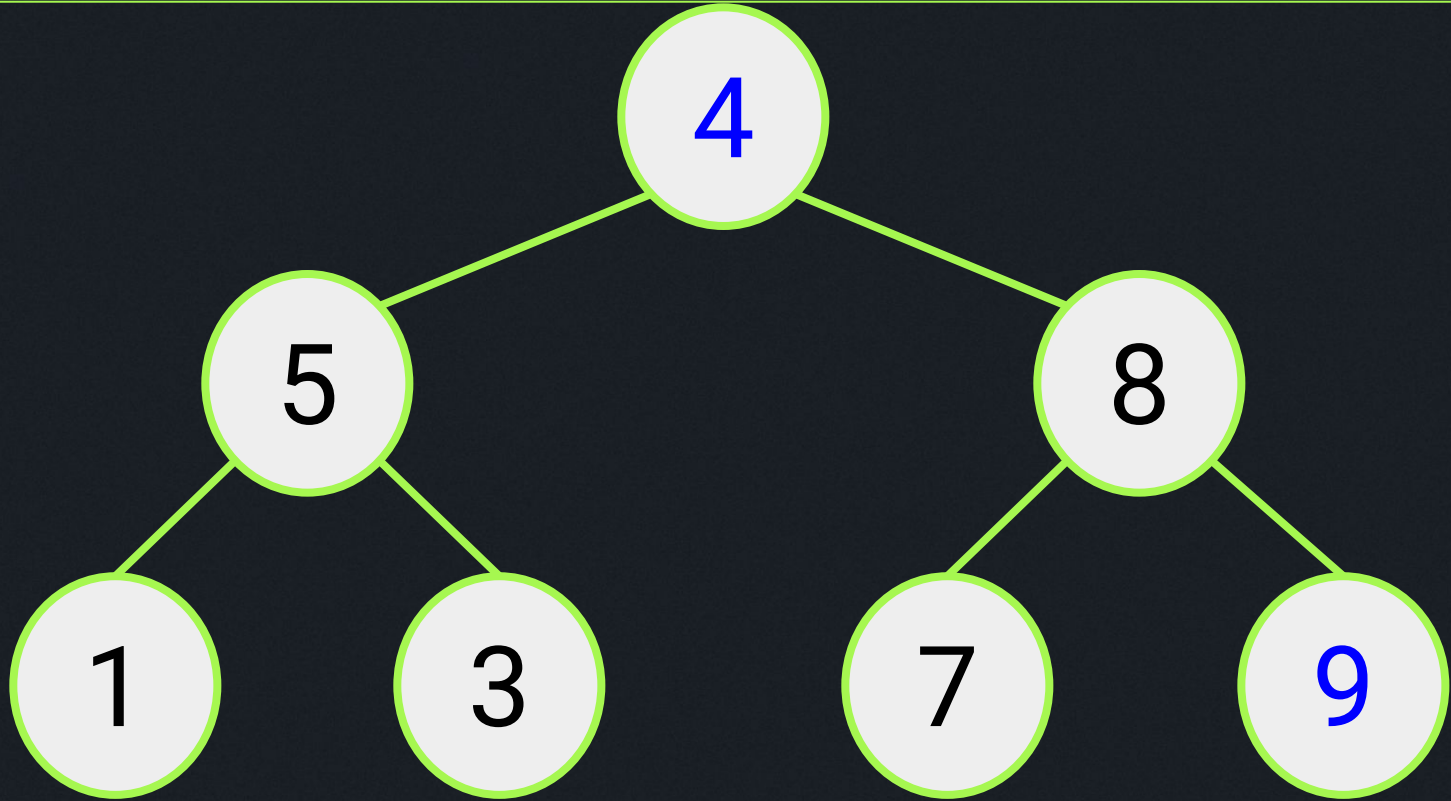
Removendo

Passo 1 - trocar raiz com último



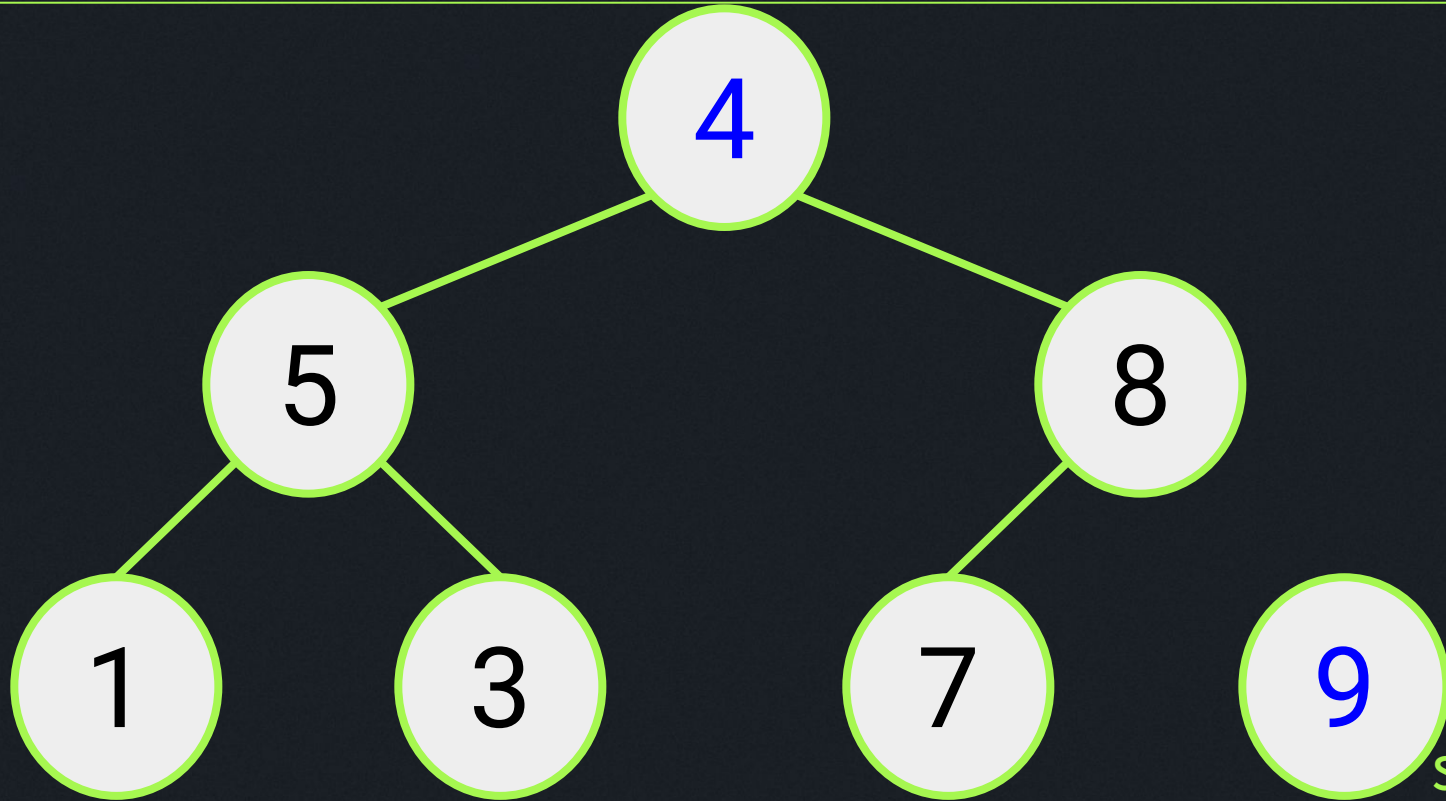
Removendo

Passo 2 - remover último elemento do vetor



Removendo

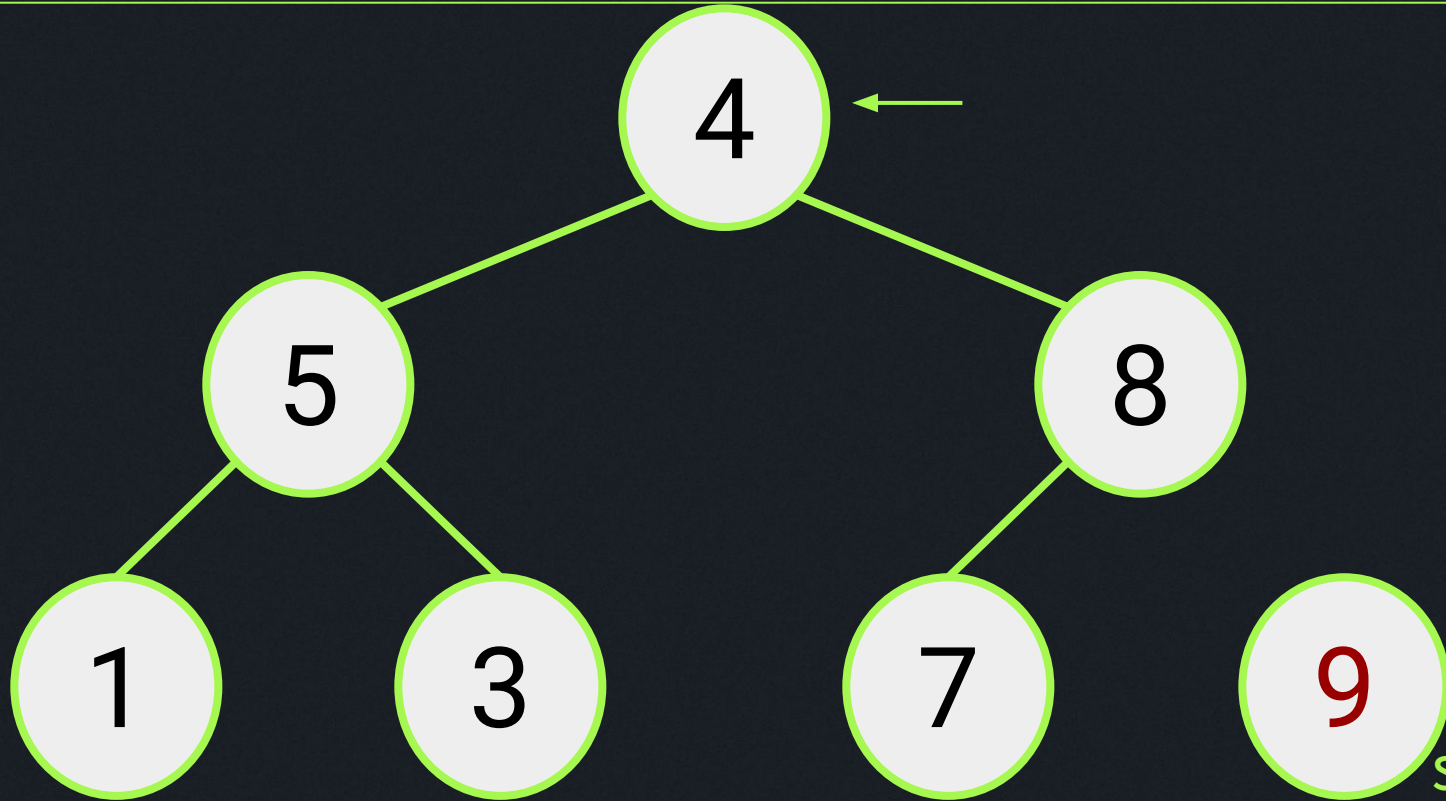
Passo 2 - remover último elemento do vetor



Salvo para
retornar

Removendo

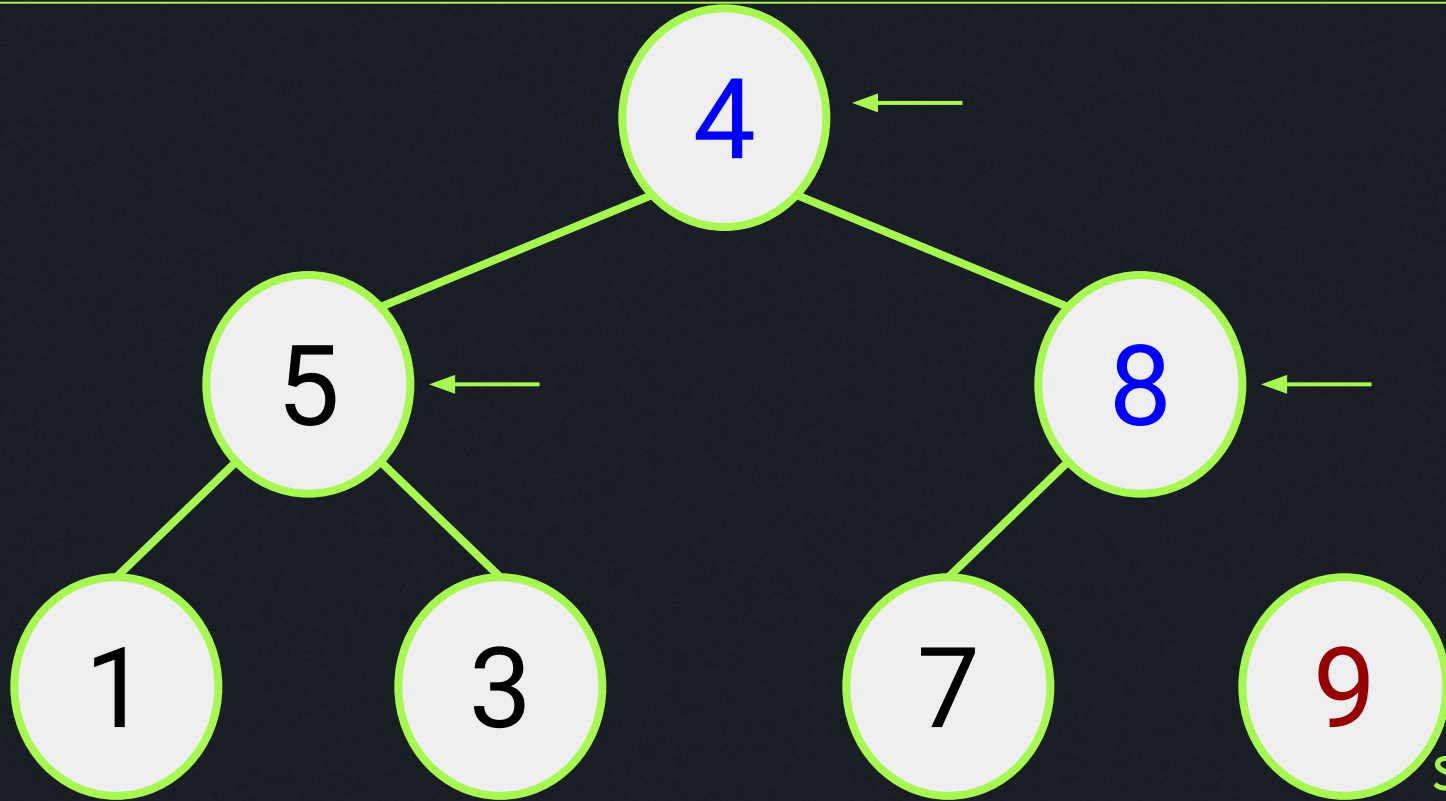
Passo 3 - heapify



Salvo para
retornar

Removendo

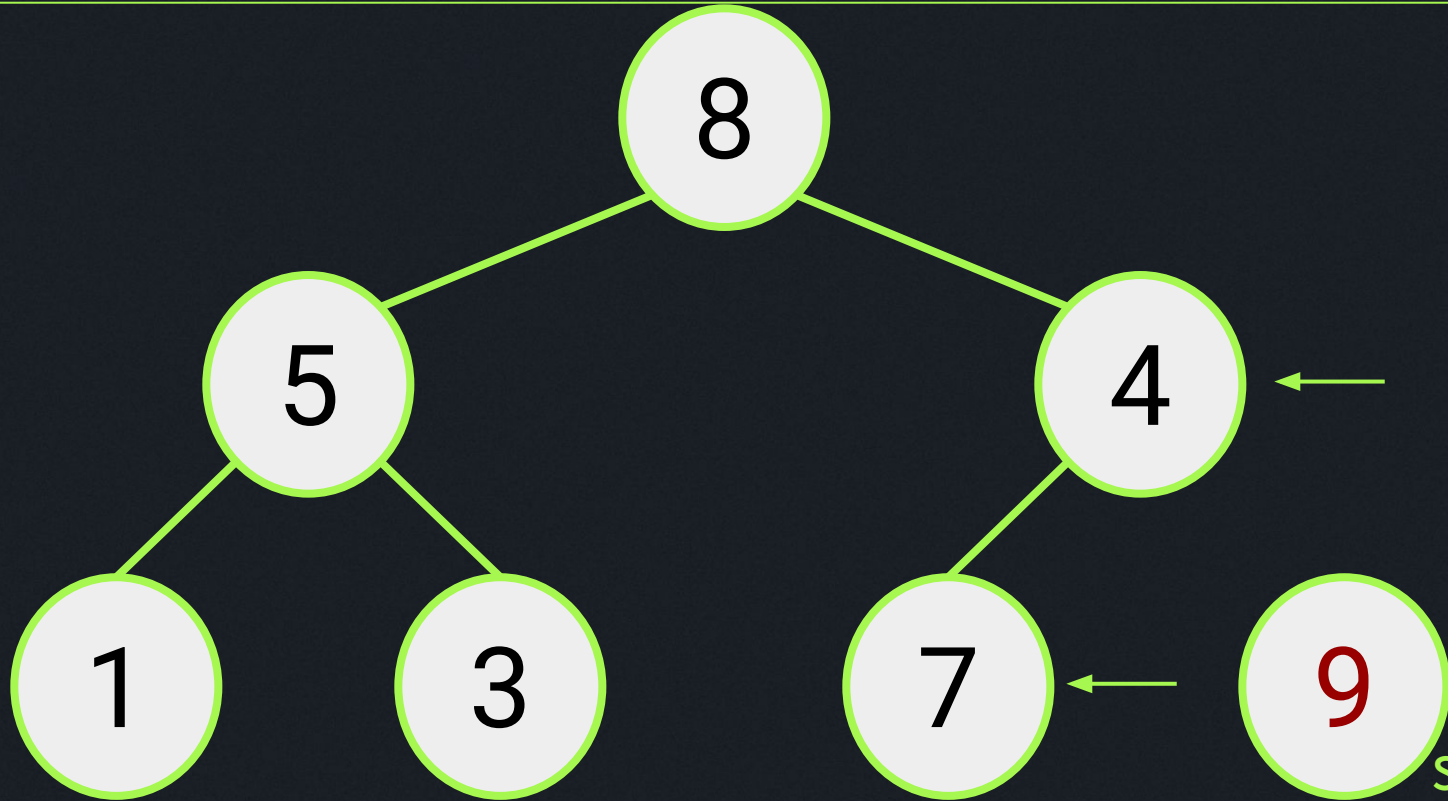
Passo 3 - heapify



Salvo para
retornar

Removendo

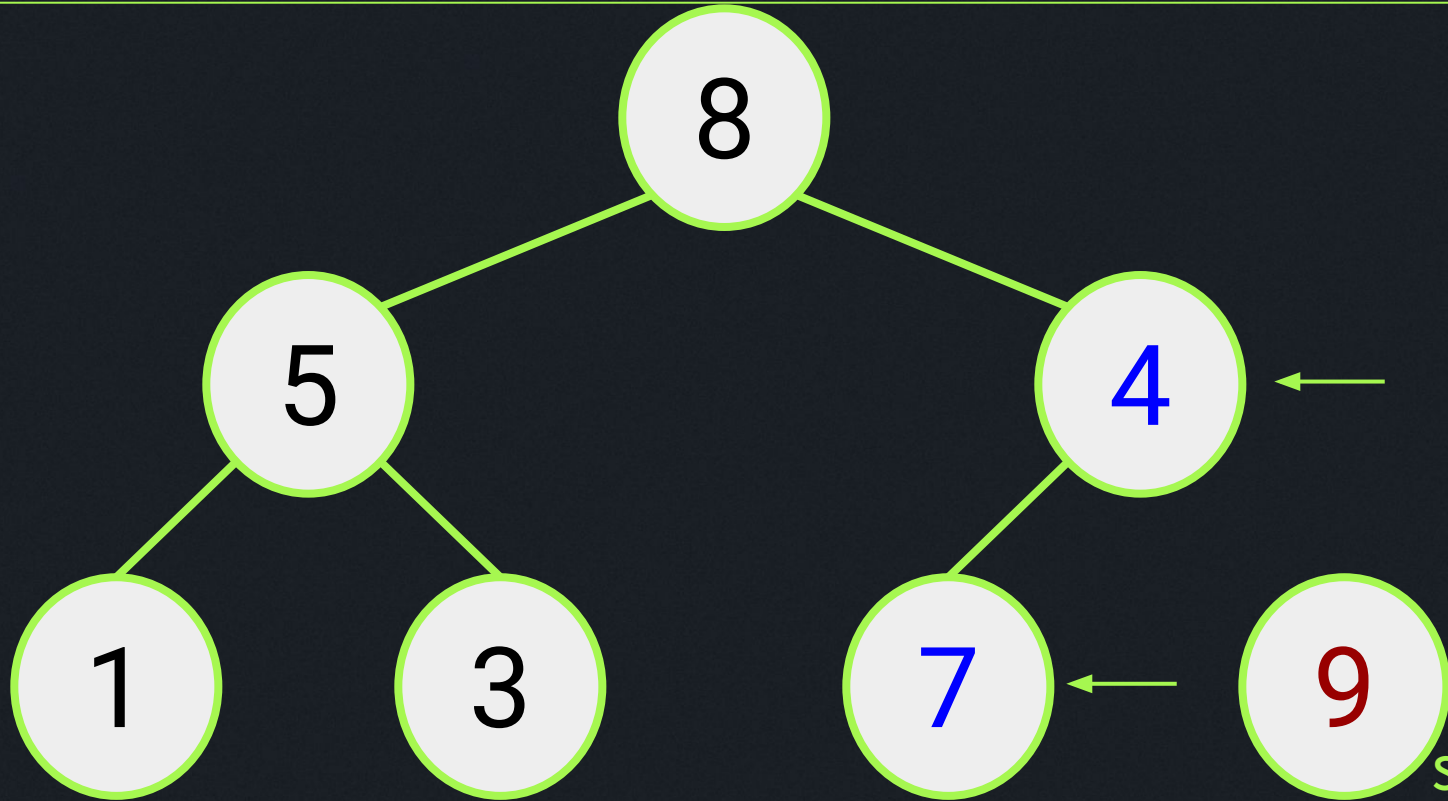
Passo 3 - heapify



Salvo para
retornar

Removendo

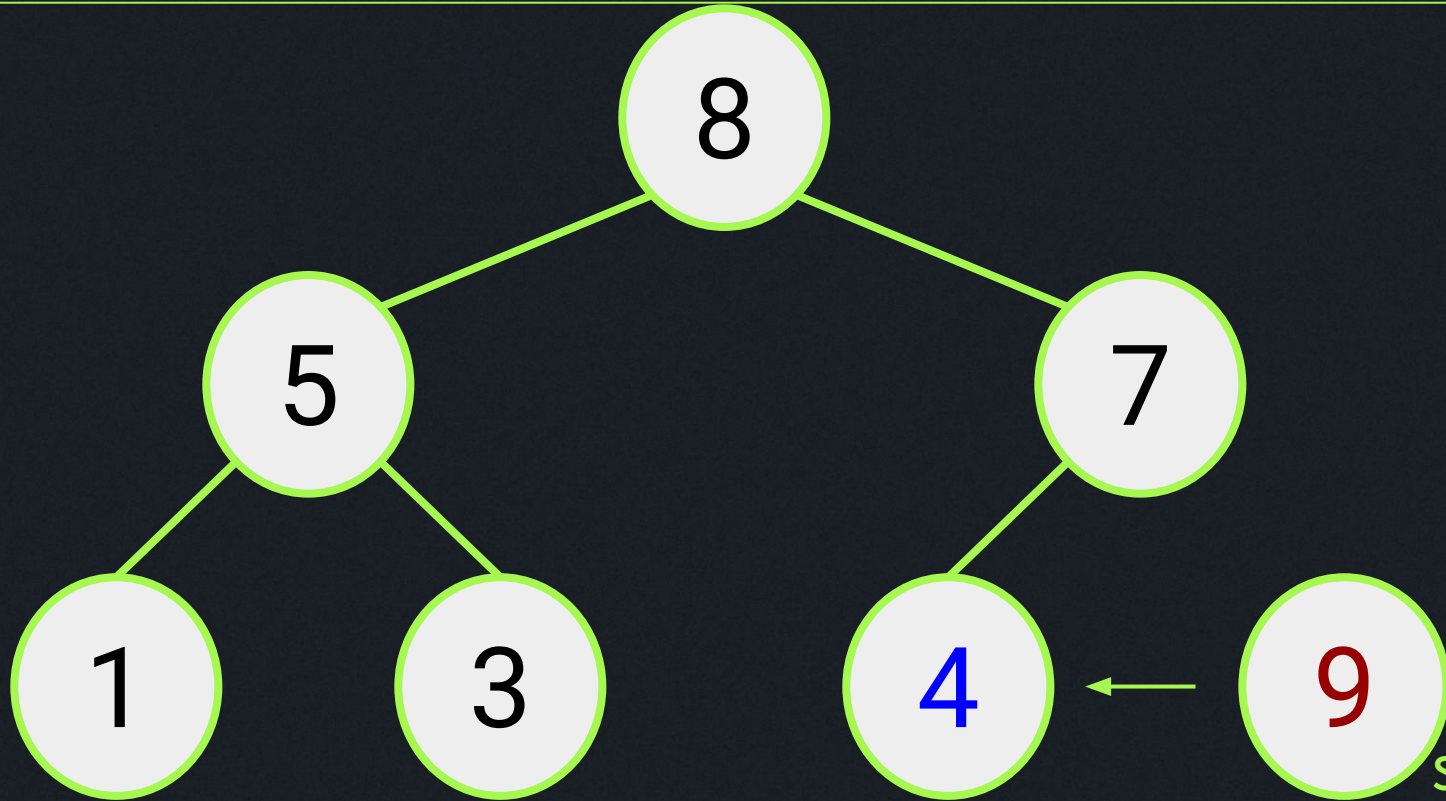
Passo 3 - heapify



Salvo para
retornar

Removendo

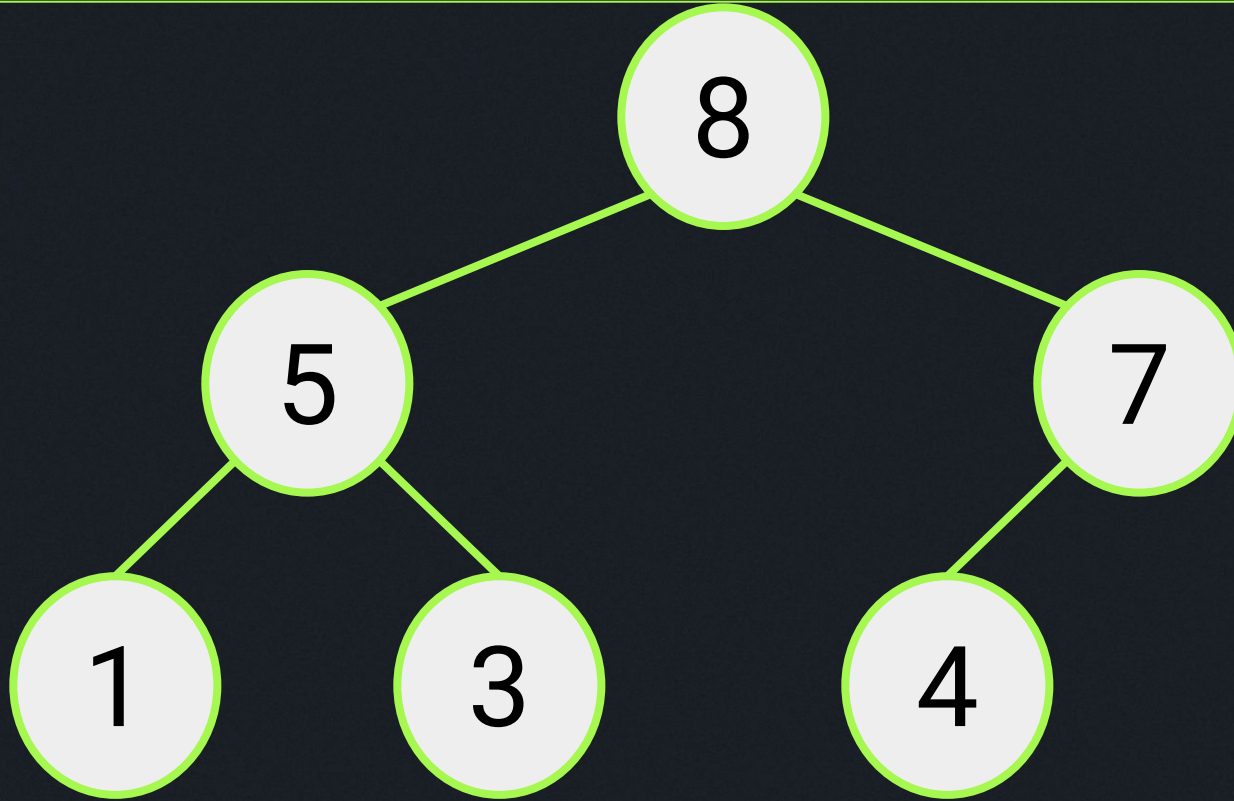
Passo 3 - heapify



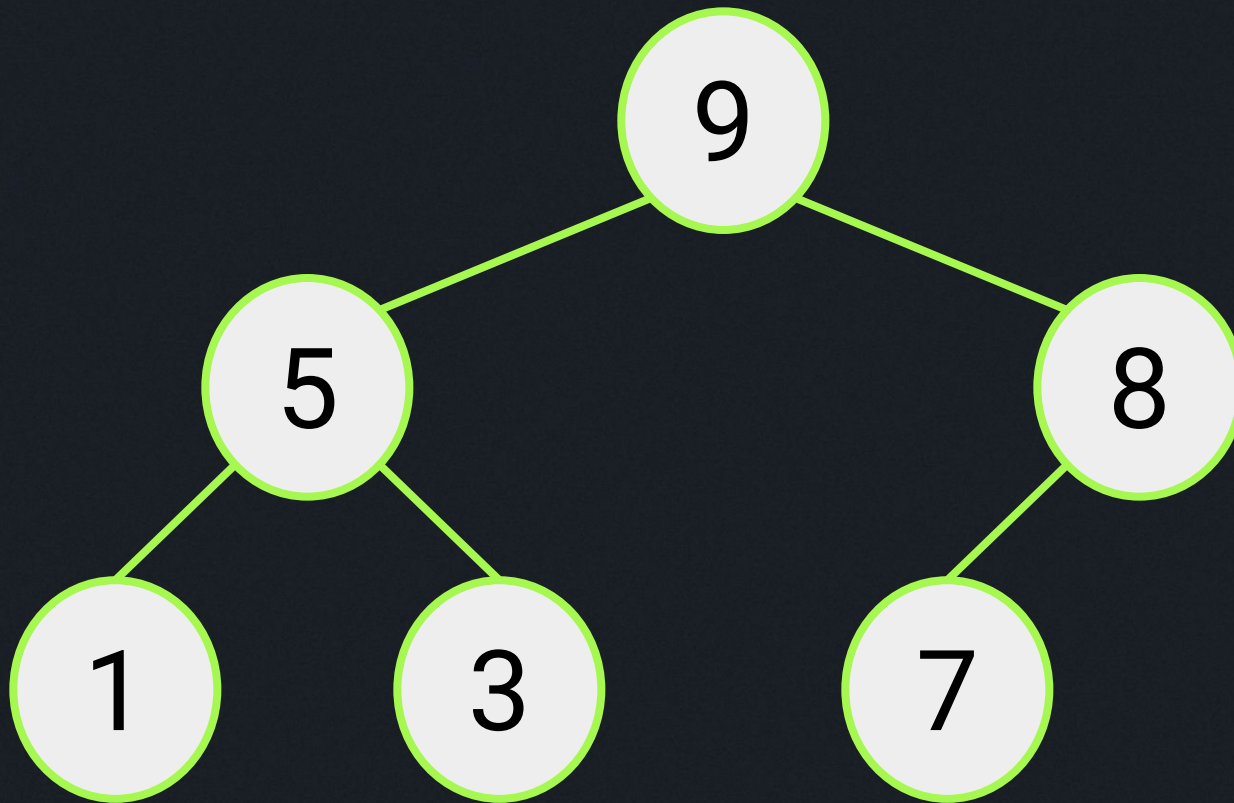
Salvo para
retornar

Removendo

Passo 3 - heapify

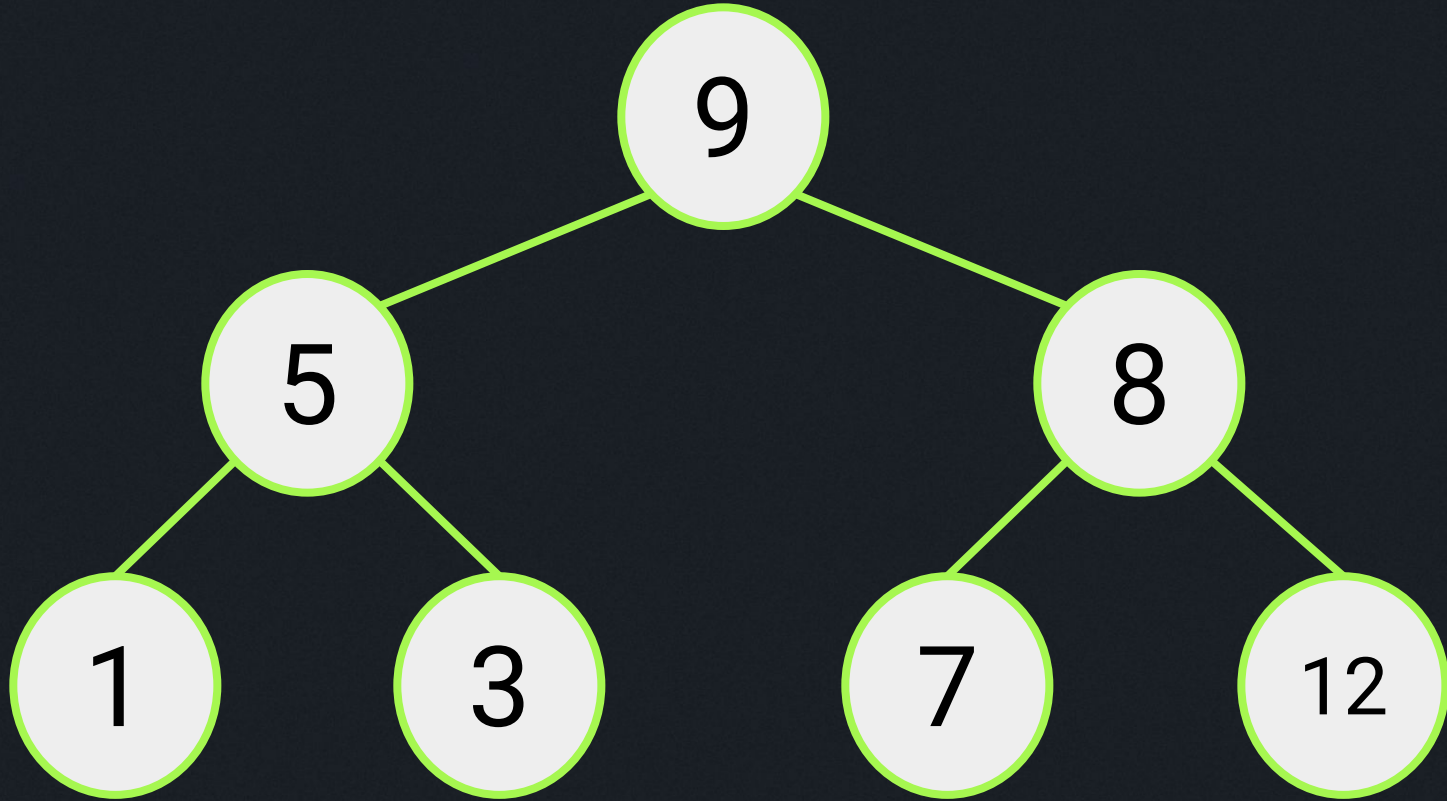


Como inserir?



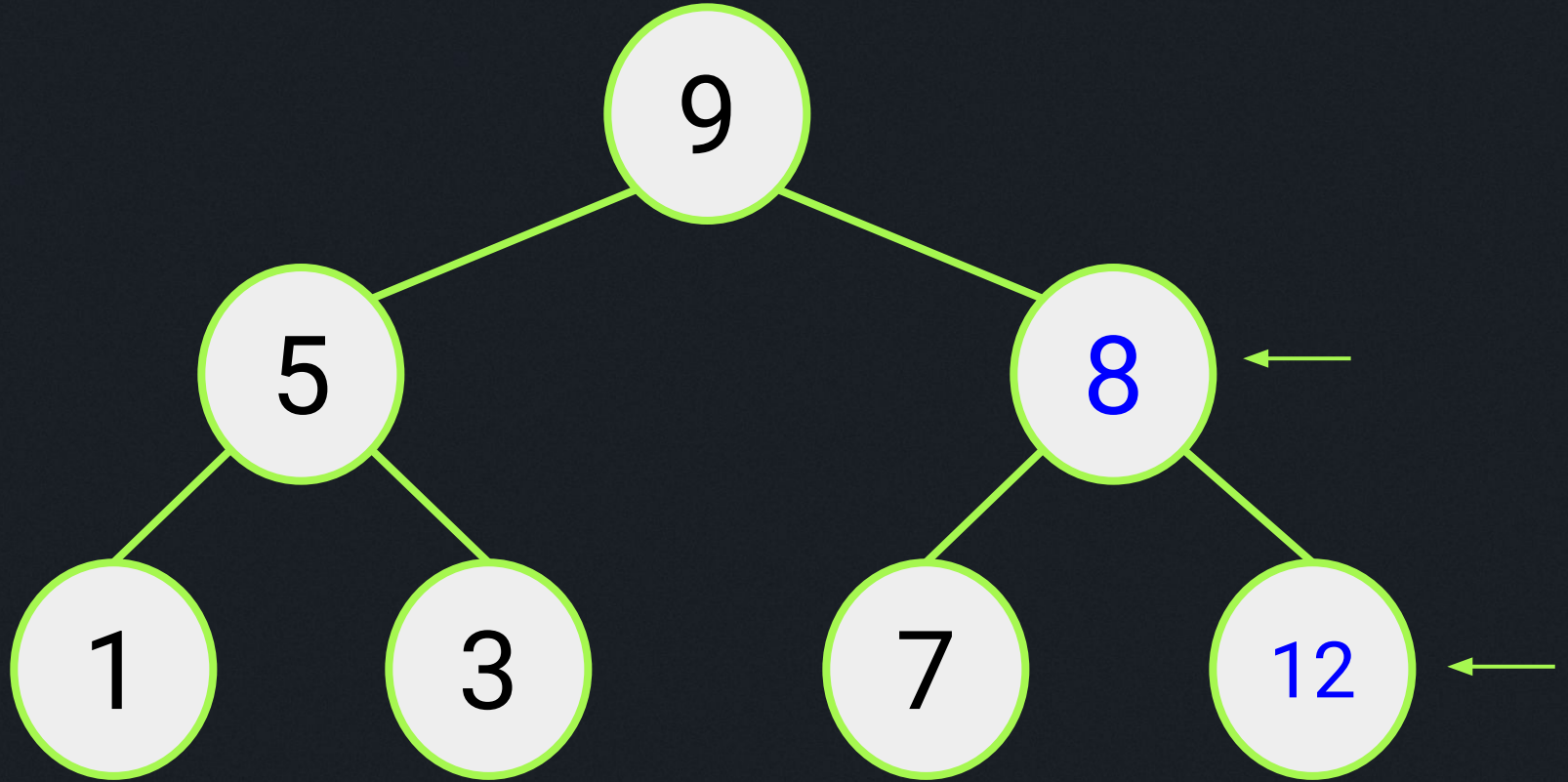
Inserindo

Passo 1 - adicionar elemento ao final do vetor



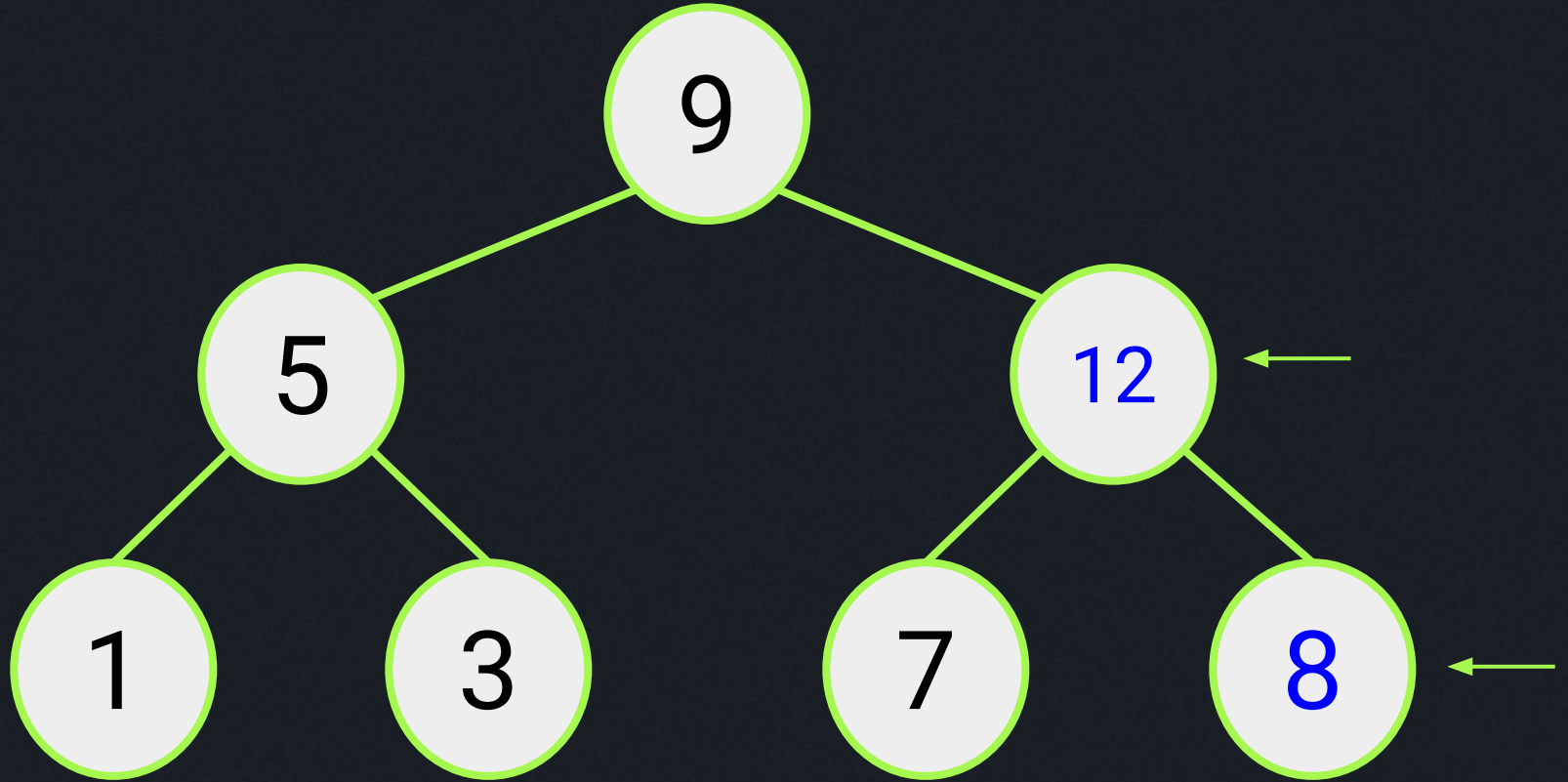
Inserindo

Passo 2 - ir trocando com o pai até lugar certo



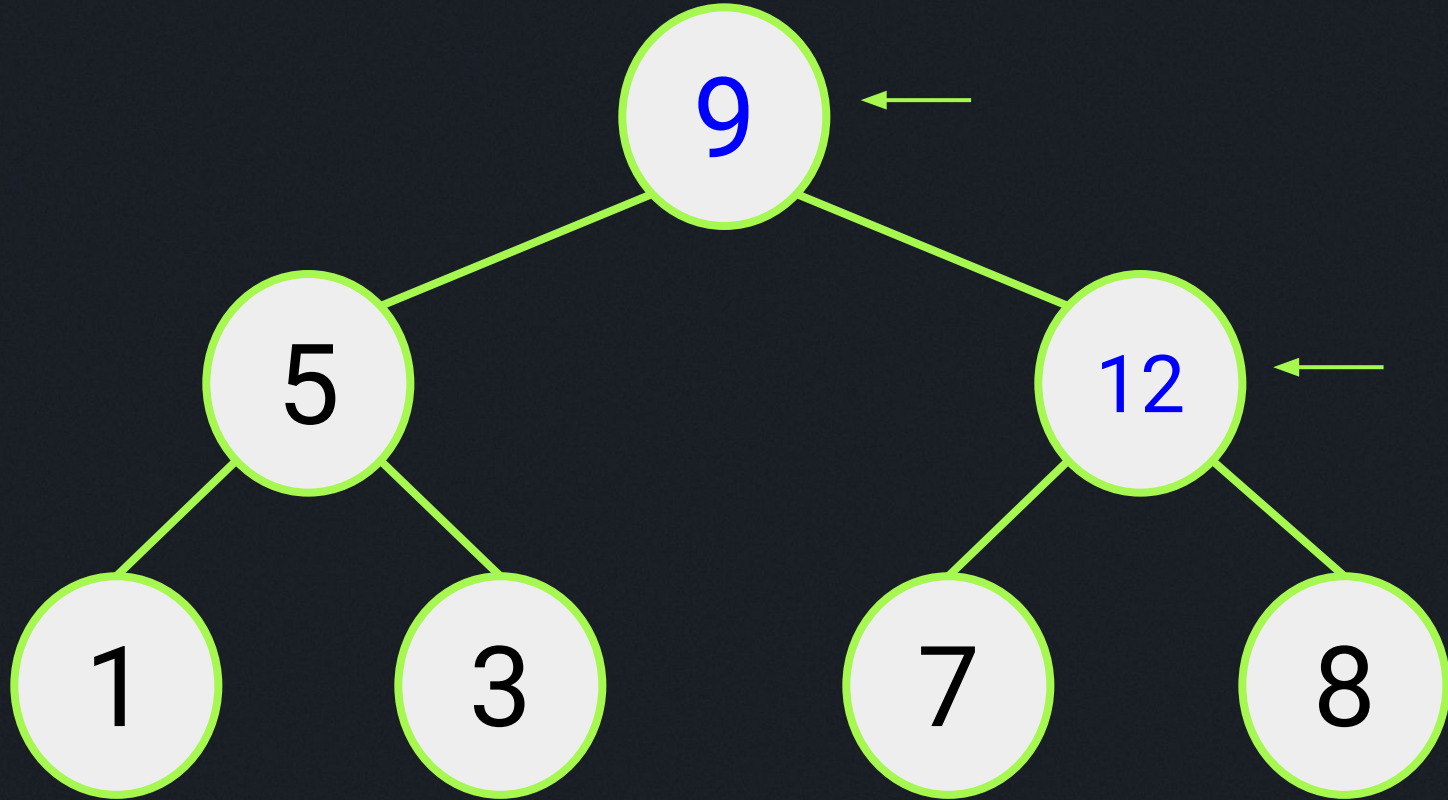
Inserindo

Passo 2 - ir trocando com o pai até lugar certo



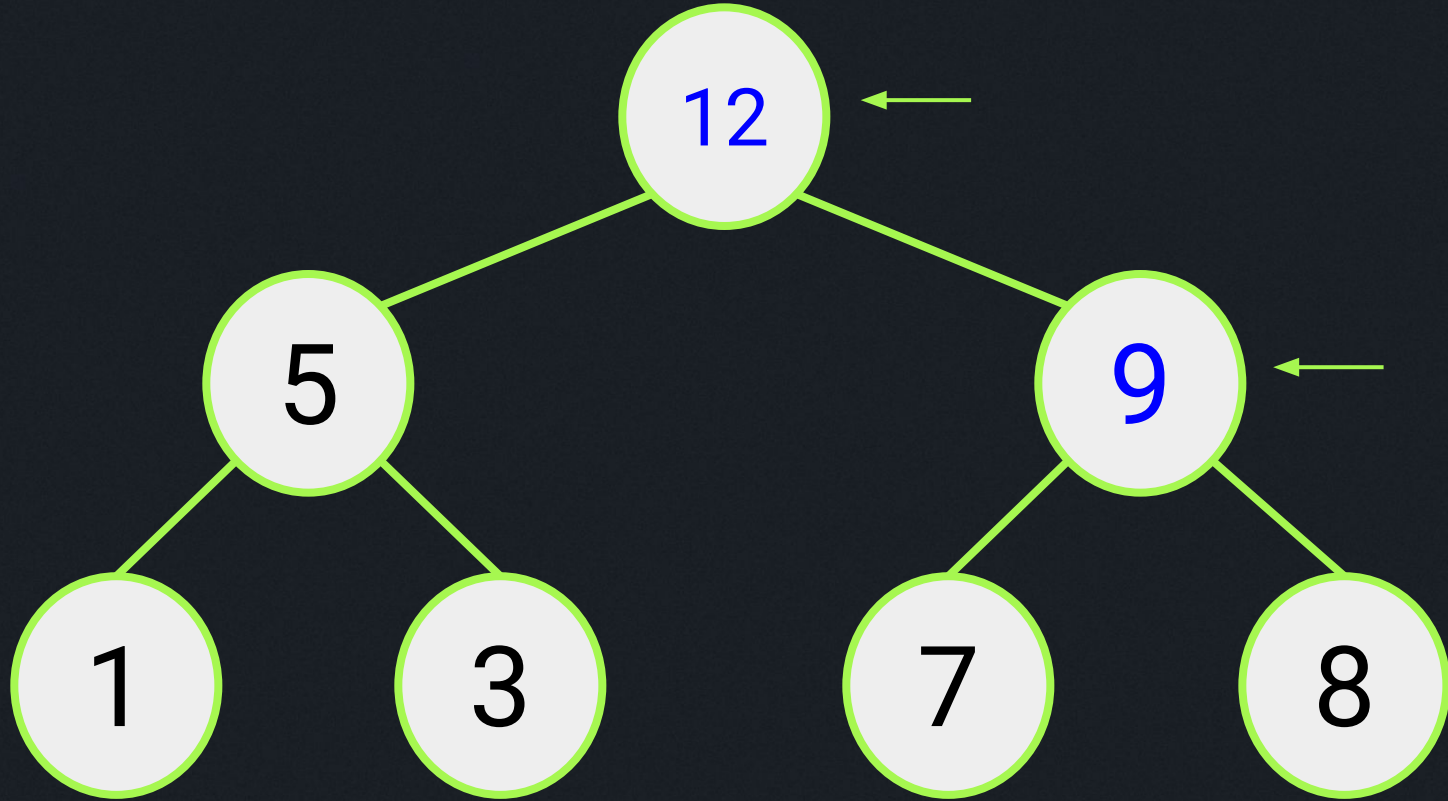
Inserindo

Passo 2 - ir trocando com o pai até lugar certo



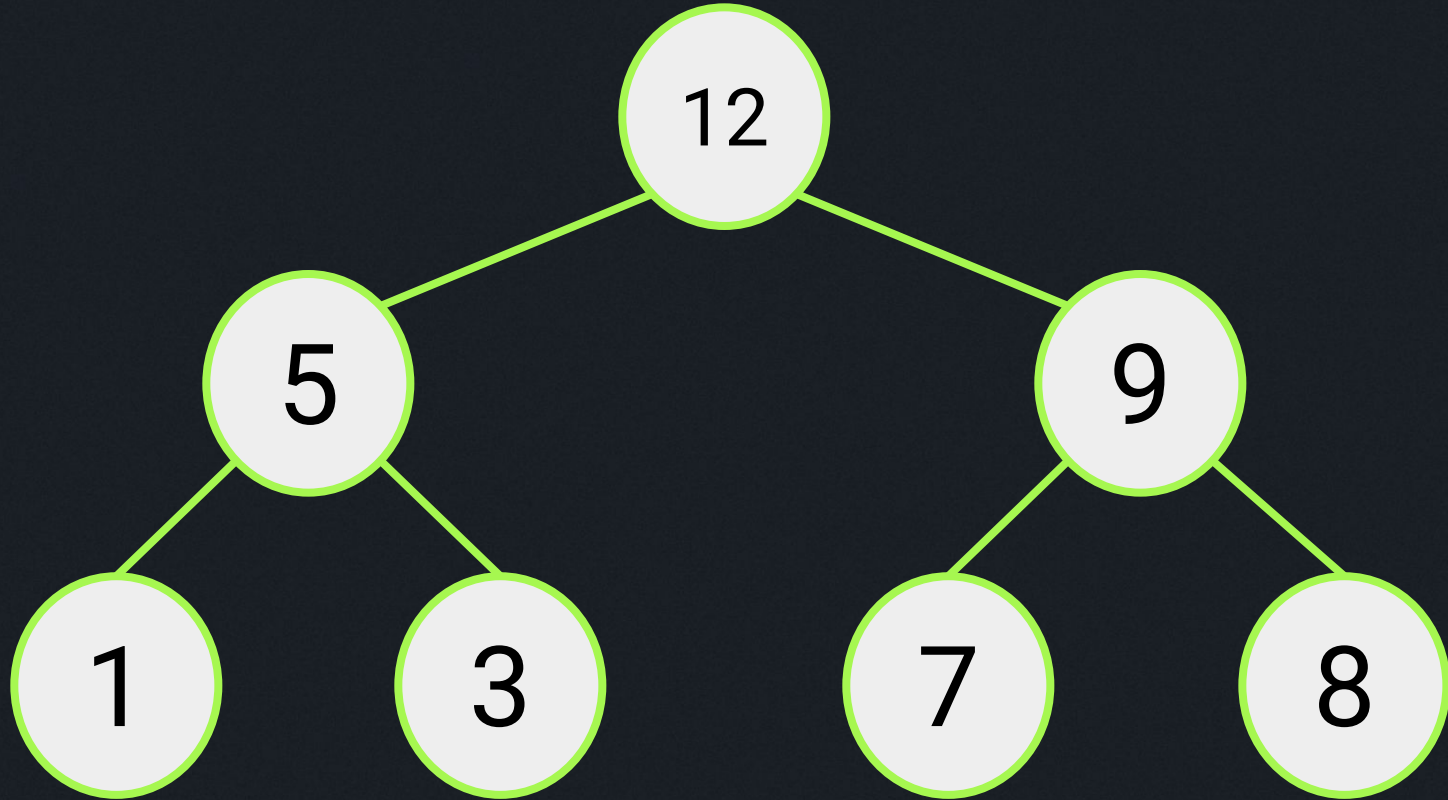
Inserindo

Passo 2 - ir trocando com o pai até lugar certo



Inserindo

Passo 2 - ir trocando com o pai até lugar certo



Complexidade

Inserir (Adiciona + Heapify/bubble up)	$O(\log n)$
Remover (Remove raíz + Heapify/bubble down)	$O(\log n)$
Topo (Acessar Raíz)	$O(1)$
Construir Heap (à partir vetor não ordenado)	$O(n)$
Heapify (restaurar propriedade de heap)	$O(\log n)$



Estratégia 5


heap implementado
à mão

~0.005s

Comparativo

Heap como vetor não ordenado	0.23 s
Heap ordenando o vetor com Sort	27 s
Heap movendo na inserção para manter ordenado	0.19 s
STL Priority Queue	0.005 s
Nossa Heap	0.005 s

4 - Exercícios no Leetcode



Praticando o aprendizado

2558. Take Gifts From the Richest Pile

<https://leetcode.com/problems/take-gifts-from-the-richest-pile>

2558. Take Gifts From the Richest Pile

Solved ✓

Easy

Topics

Companies

Hint

You are given an integer array `gifts` denoting the number of gifts in various piles. Every second, you do the following:

- Choose the pile with the maximum number of gifts.
- If there is more than one pile with the maximum number of gifts, choose any.
- Leave behind the floor of the square root of the number of gifts in the pile. Take the rest of the gifts.

Return the number of gifts remaining after `k` seconds.

Example 1:

Input: `gifts = [25,64,9,4,100]`, `k = 4`

Output: 29

Explanation:

The gifts are taken in the following way:

– In the first second, the last pile is chosen and 10 gifts

Um max heap poderia
representar os presentes?

2558. Take Gifts From the Richest Pile

<https://leetcode.com/problems/take-gifts-from-the-richest-pile>

```
import heapq

class Solution:
    def pickGifts(self, gifts: List[int], k: int) -> int:
        q = [ -gift for gift in gifts ] #maxheap with negatives
        heapq.heapify(q) #build a heap with gifts

        for i in range(k):
            higherGift = -heapq.heappop(q)
            sqrtGift = int(math.sqrt(higherGift))
            heapq.heappush(q, -sqrtGift)

        return -sum(q)
```


1046. Last Stone Weight

<https://leetcode.com/problems/last-stone-weight/>

1046. Last Stone Weight

Solved ✓

Easy

Topics

Companies

Hint

You are given an array of integers `stones` where `stones[i]` is the weight of the i^{th} stone.

We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights x and y with $x \leq y$. The result of this smash is:

- If $x == y$, both stones are destroyed, and
- If $x \neq y$, the stone of weight x is destroyed, and the stone of weight y has new weight $y - x$.

At the end of the game, there is **at most one** stone left.

Return *the weight of the last remaining stone*. If there are no stones left, return `0`.

Example 1:

Input: `stones = [2,7,4,1,8,1]`

Output: `1`

Que tal simular o jogo com um max heap?

Para o caso de teste `[4, 5, 4, 5]`, sua resposta é `0`?

1046. Last Stone Weight

<https://leetcode.com/problems/last-stone-weight/>

```
import heapq

class Solution:
    def lastStoneWeight(self, stones: List[int]) -> int:
        q = [-stone for stone in stones] #maxHeap with negative values
        heapq.heapify(q) #build the maxHeap

        while len(q)>1:
            stone1 = heapq.heappop(q)
            stone2 = heapq.heappop(q)
            if stone1!=stone2:
                newStone = - abs(stone1-stone2)
                heapq.heappush(q, newStone)

        return 0 if len(q)==0 else -heapq.heappop(q)
```

239. Sliding Window Maximum

<https://leetcode.com/problems/sliding-window-maximum>

239. Sliding Window Maximum

Solved 

Hard  Topics  Companies  Hint

You are given an array of integers `nums`, there is a sliding window of size `k` which is moving from the very left of the array to the very right. You can only see the `k` numbers in the window. Each time the sliding window moves right by one position.

Return the max sliding window.

Example 1:

Input: `nums = [1,3,-1,-3,5,3,6,7]`, `k = 3`

Output: `[3,3,5,5,6,7]`

Explanation:

Window position	Max
[1 3 -1] -3 5 3 6 7	3
1 [3 -1 -3] 5 3 6 7	3
1 3 [-1 -3 5] 3 6 7	5
1 3 -1 [-3 5 3] 6 7	5
1 3 -1 -3 [5 3 6] 7	6
1 3 -1 -3 5 [3 6 7]	7

Um max heap é a melhor forma de encontrar o maior valor entre os k primeiros.

O problema é quando a janela desliza. Adicionar um novo elemento no max heap é fácil.

Mas remover um elemento e manter a propriedade do heap não é!

Solução? Não remover!

Inserir no heap o valor e o índice no vetor.

Quando for olhar o maior, se o índice estiver fora da sliding window, você o remove. Assim vai considerar apenas valores dentro da sliding window.

239. Sliding Window Maximum

<https://leetcode.com/problems/sliding-window-maximum>

```
import heapq
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        maxHeap = [ (-nums[i], i) for i in range(k) ] # negative for maxHeap, indexes
        heapq.heapify(maxHeap)
        output = []
        startWindow, endWindow = 0, k-1
        while endWindow<len(nums):
            while True:
                maxVal, index = maxHeap[0] #peek maxHeap
                if index>=startWindow and index<=endWindow:
                    break #stops when peek maxHeap is in sliding window
                heapq.heappop(maxHeap) #remove elements out of sliding window
            output.append( -maxVal )
            startWindow += 1
            endWindow += 1
            if endWindow<len(nums):
                heapq.heappush(maxHeap, (-nums[endWindow], endWindow))
        return output
```


23. Merge k Sorted Lists

<https://leetcode.com/problems/merge-k-sorted-lists/>

23. Merge k Sorted Lists

Solved ✓

Hard

Topics

Companies

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: `lists = [[1,4,5],[1,3,4],[2,6]]`

Output: `[1,1,2,3,4,4,5,6]`

Explanation: The linked-lists are:

```
[
  1->4->5,
  1->3->4,
  2->6
]
```

Um min heap pode ajudar a
implementar a ideia do merge com
várias listas?

23. Merge k Sorted Lists

<https://leetcode.com/problems/merge-k-sorted-lists/>

```
import heapq

class Solution:
    def mergeKLists(self, lists:
List[Optional[ListNode]]) ->
Optional[ListNode]:
    minHeap = []
    for index, head in enumerate(lists):
        if head:
            minHeap.append( (head.val,
index) )
    heapq.heapify(minHeap)
```

```
    head = None
    ptr = None
    while minHeap:
        _, minIndex = heapq.heappop(minHeap)
        if head==None:
            head = lists[minIndex]
            ptr = head
        else:
            ptr.next = lists[minIndex]
            ptr = ptr.next

        lists[minIndex] = lists[minIndex].next
        if lists[minIndex]:
            heapq.heappush(minHeap,
(lists[minIndex].val, minIndex))
    return head
```



Todos os códigos

para facilitar o
estudo, todos os
códigos juntos

Obrigado