Você sabia que se você remover suas veias e esticá-las ao redor do mundo, você morre?

Heapsort

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Heaps

Em uma max-heap, onde está o maior elemento?

Heaps

Em uma max-heap, onde está o maior elemento?

Na primeira posição

A heap é ordenada?

Heaps

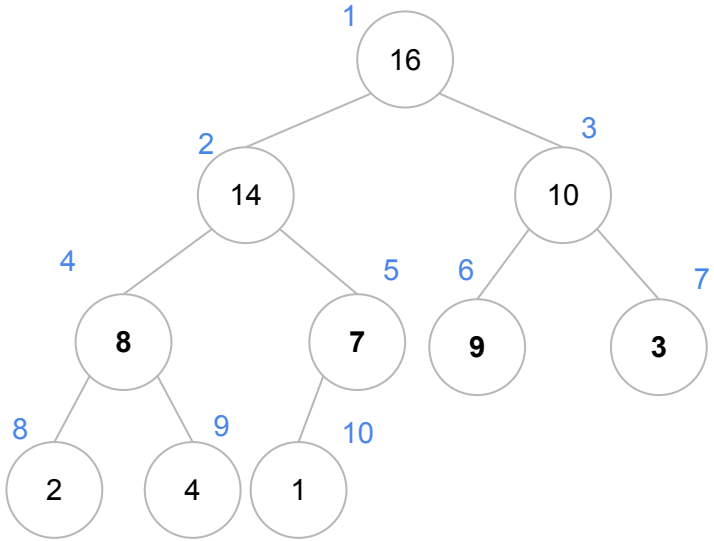
Em uma max-heap, onde está o maior elemento?

Na primeira posição

A heap é ordenada?

Não, elementos em um mesmo nível não possuem ordem

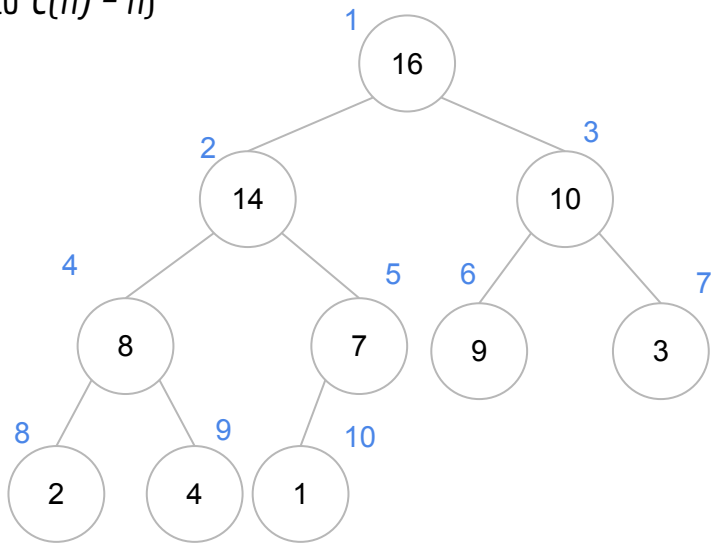
Só garantimos que o pai é maior ou igual ao seus filhos



1	2	3	4	5	6	7	8	9	10
16	14	10	**8**	**7**	**9**	**3**	2	4	1

Heapsort - Ideia

O heapsort primeiro constrói uma heap a partir do vetor (com um custo $C(n) = n$)

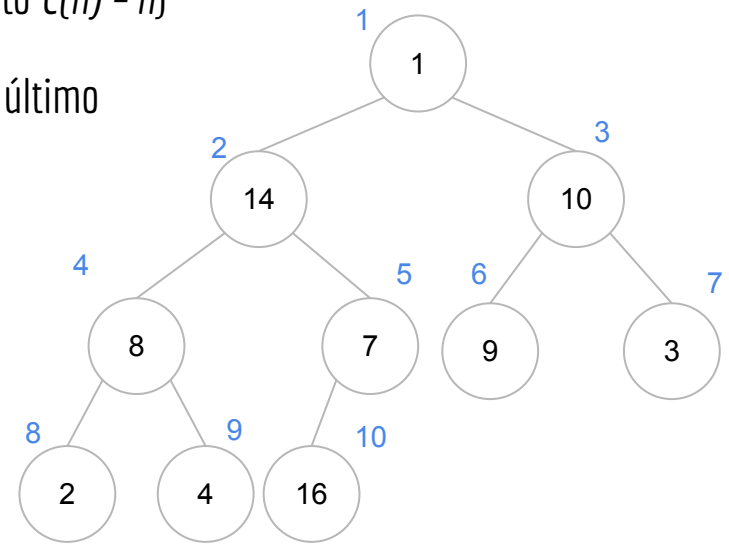


1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Heapsort - Ideia

O heapsort primeiro constrói uma heap a partir do vetor (com um custo $C(n) = n$)

O maior elemento está em h[1], então esse elemento é trocado com o último



1	2	3	4	5	6	7	8	9	10
1	14	10	8	7	9	3	2	4	16

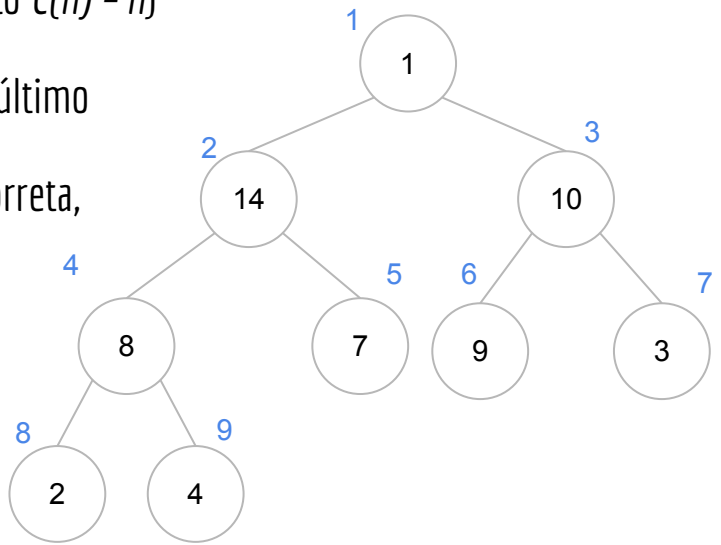
Heapsort - Ideia

O heapsort primeiro constrói uma heap a partir do vetor (com um custo $C(n) = n$)

O maior elemento está em h[1], então esse elemento é trocado com o último

Agora desconsideramos o último elemento, que está na sua posição correta, e assumimos que a heap é h[1..n-1]

O que fazer?



1	2	3	4	5	6	7	8	9	10
1	**14**	**10**	**8**	**7**	**9**	**3**	**2**	**4**	16

Heapsort - Ideia

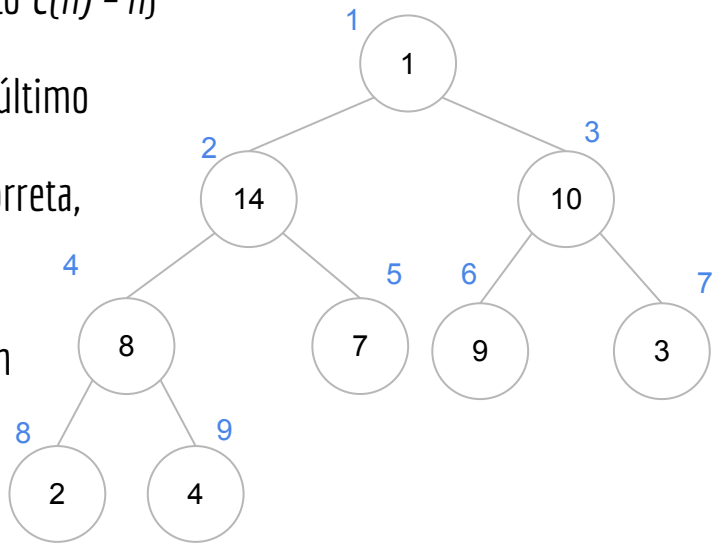
O heapsort primeiro constrói uma heap a partir do vetor (com um custo $C(n) = n$)

O maior elemento está em h[1], então esse elemento é trocado com o último

Agora desconsideramos o último elemento, que está na sua posição correta, e assumimos que a heap é h[1..n-1]

Note que as subárvores esquerda e direita a partir de h[1] permanecem max-heaps, mas a árvore que inicia em h[i] pode violar a propriedade da max-heap.

Basta chamar max-heapify



1	2	3	4	5	6	7	8	9	10
1	**14**	**10**	**8**	**7**	**9**	**3**	**2**	**4**	16

Heapsort - Ideia

O heapsort primeiro constrói uma heap a partir do vetor (com um custo $C(n) = n$)

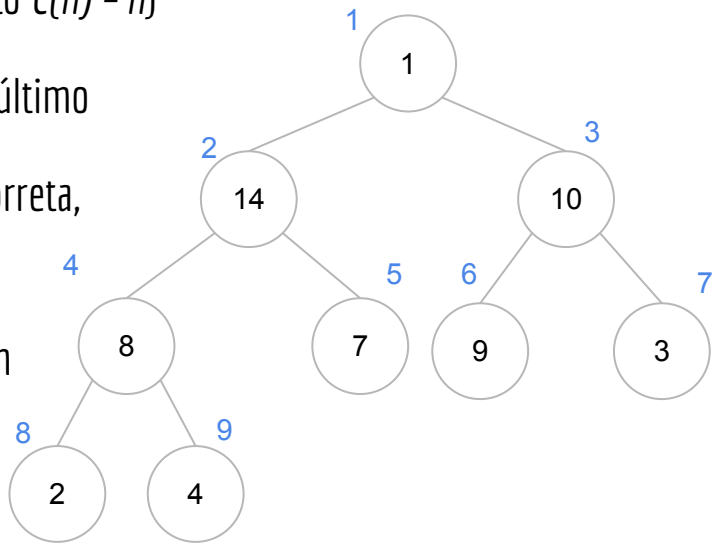
O maior elemento está em h[1], então esse elemento é trocado com o último

Agora desconsideramos o último elemento, que está na sua posição correta, e assumimos que a heap é h[1..n-1]

Note que as subárvores esquerda e direita a partir de h[1] permanecem max-heaps, mas a árvore que inicia em h[i] pode violar a propriedade da max-heap.

Basta chamar max-heapify

Agora repita o processo...



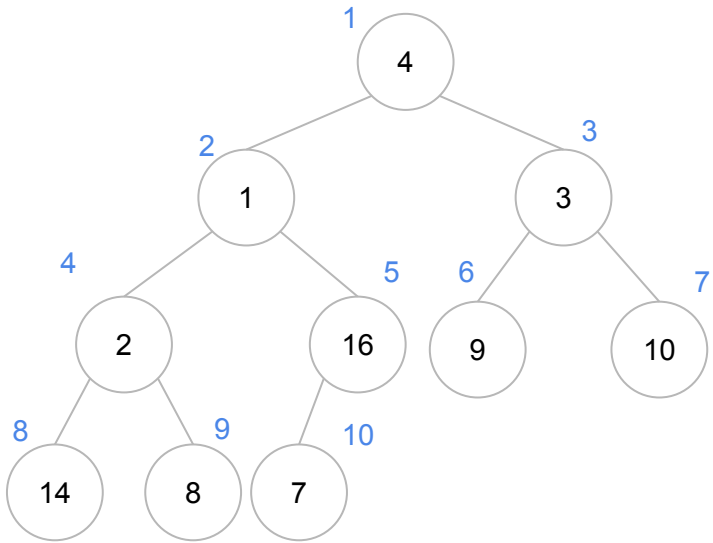
1	2	3	4	5	6	7	8	9	10
1	**14**	**10**	**8**	**7**	**9**	**3**	**2**	**4**	16

Heapsort

```
função heapsort (v,n)
entrada: vetor v, indexado por [1..n]
saída: o vetor v modificado de forma que v[1..n] é um vetor ordenado.

construir-max-heap(v,n)
para i ← n até 2 passo -1
     trocar(v, 1, i)
     n ← n - 1
     max-heapify(v,1,n)
```

Teste de Mesa

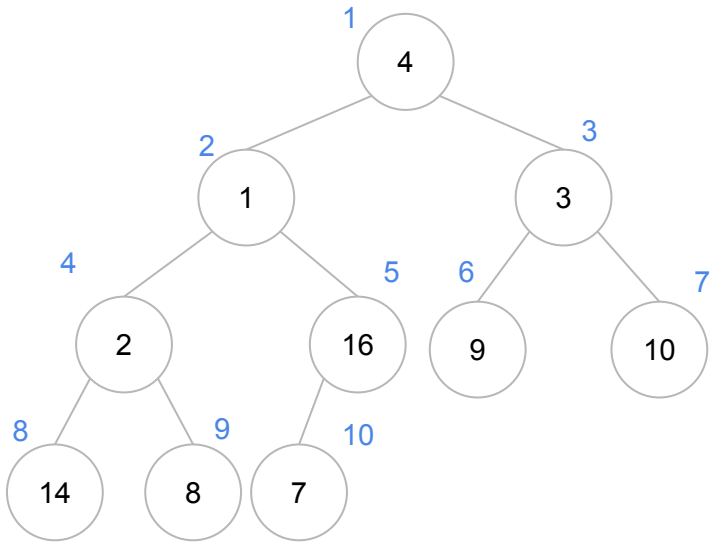



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7

Teste de Mesa

n	i
10	

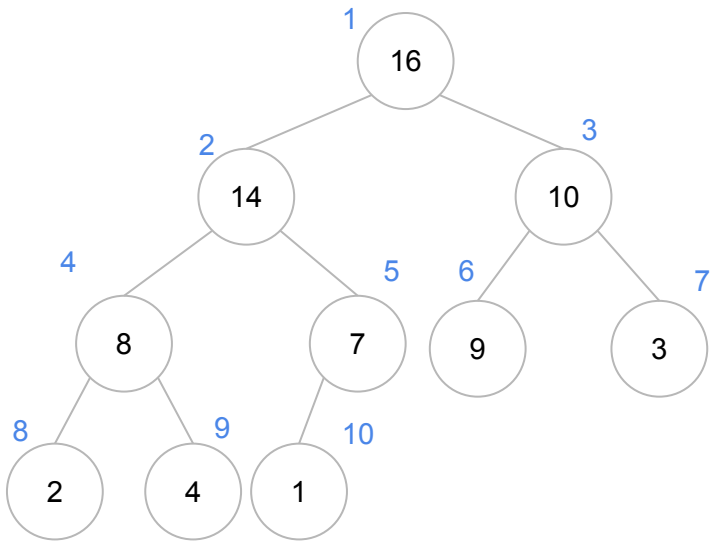



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7

Teste de Mesa

n	i
10	

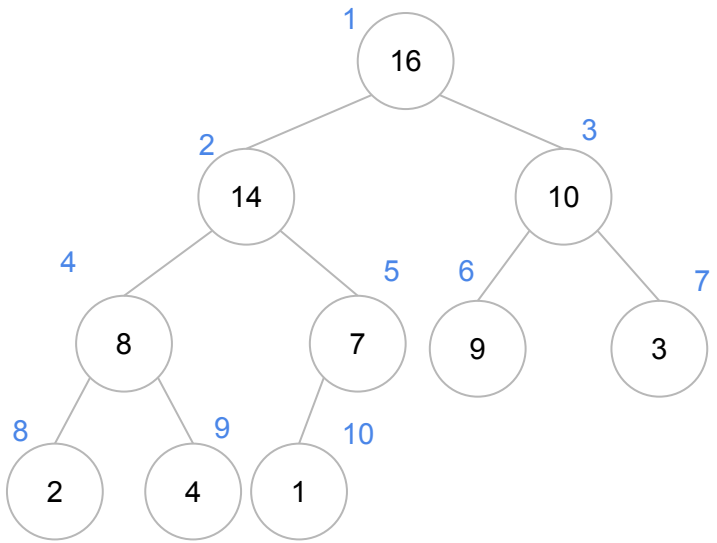



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Teste de Mesa

n	i
10	10

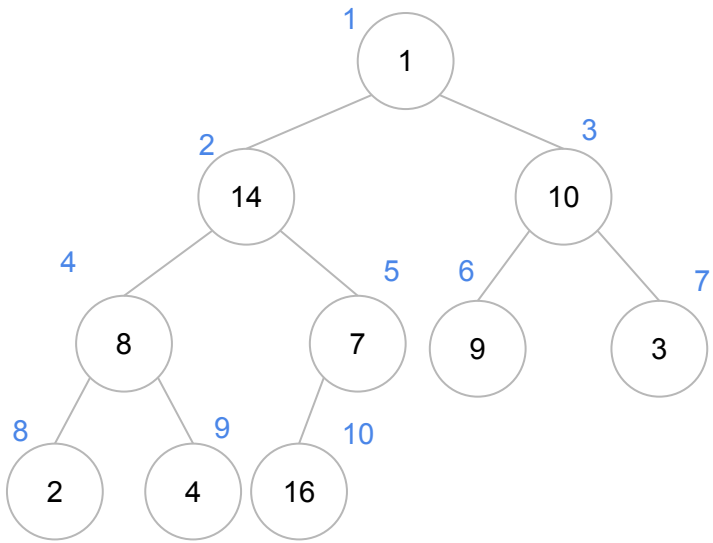



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Teste de Mesa

n	i
10	10

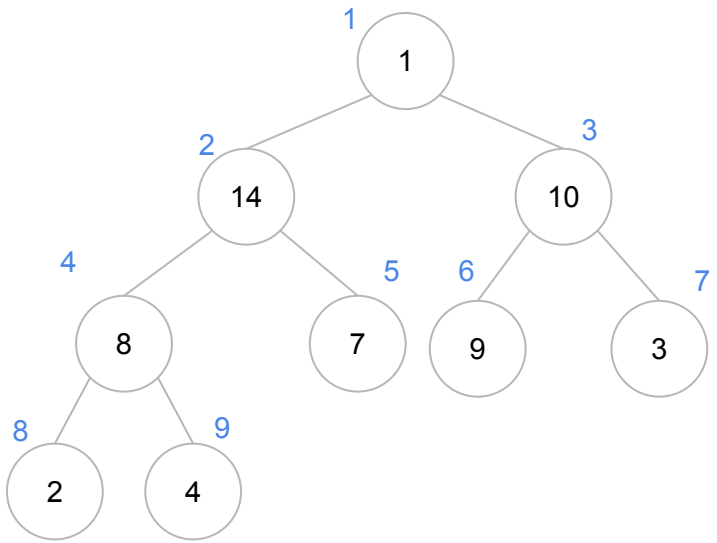



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	14	10	8	7	9	3	2	4	16

Teste de Mesa

n	i
10	10
9	

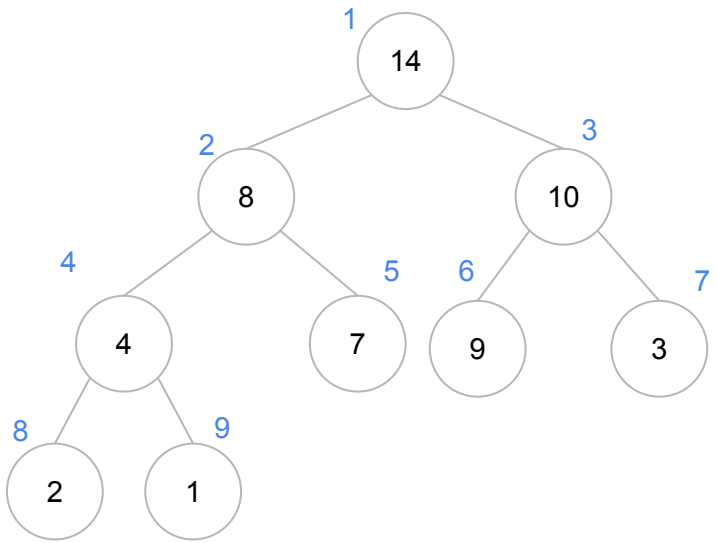



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	14	10	8	7	9	3	2	4	**16**

Teste de Mesa

n	i
10	10
9	

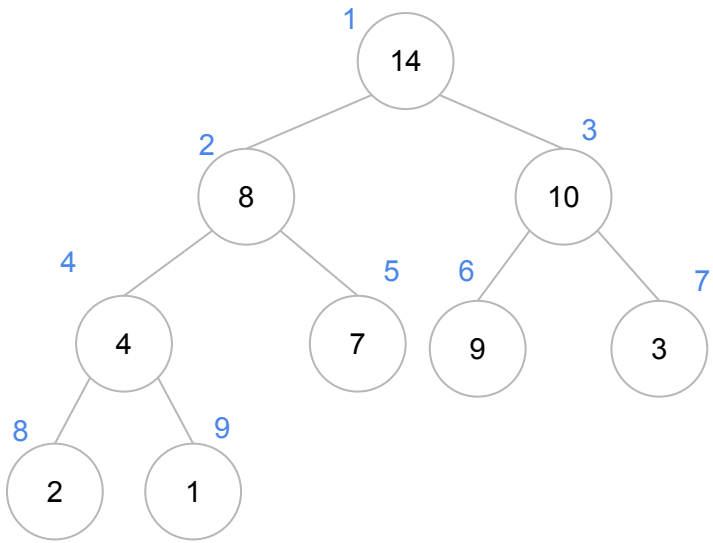



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
14	8	10	4	7	9	3	2	1	**16**

Teste de Mesa

n	i
10	10
9	9

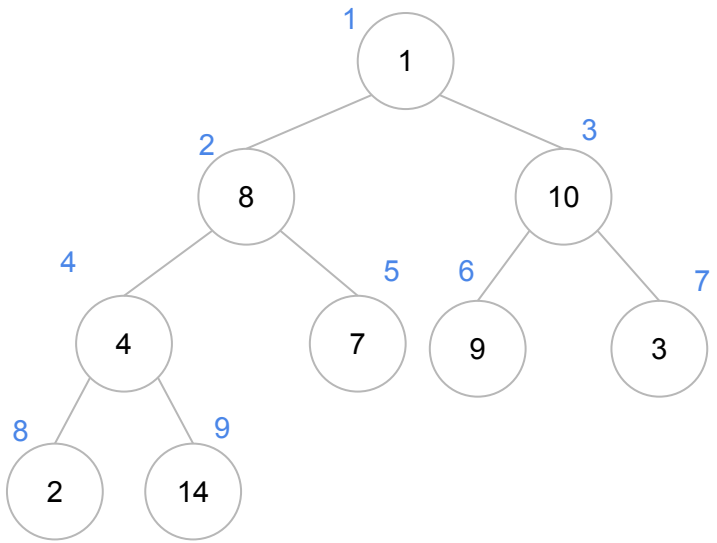



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
14	8	10	4	7	9	3	2	1	**16**

Teste de Mesa

n	i
10	10
9	9

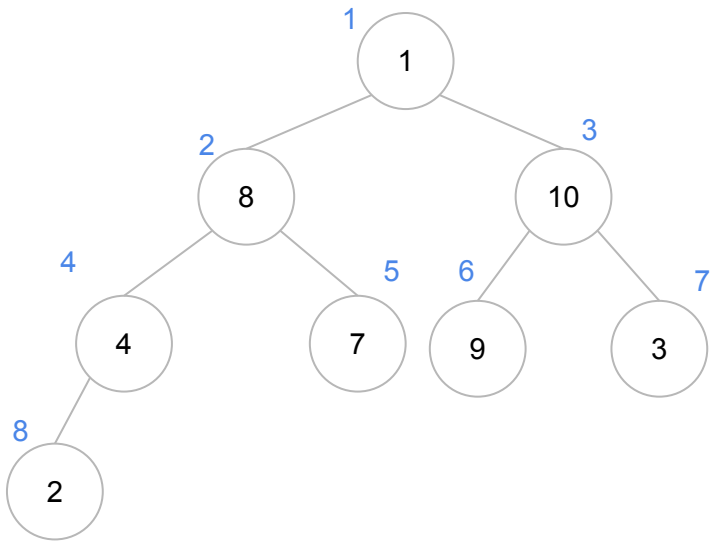



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	8	10	4	7	9	3	2	14	**16**

Teste de Mesa

n	i
10	10
9	9
8	

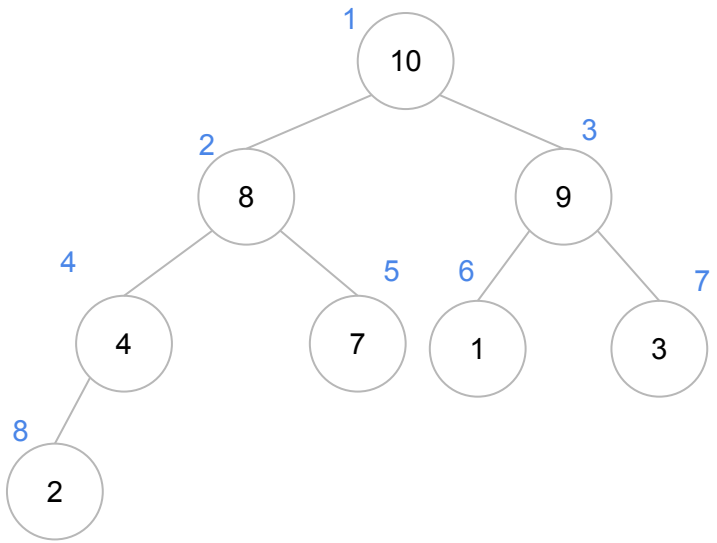



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	8	10	4	7	9	3	2	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	

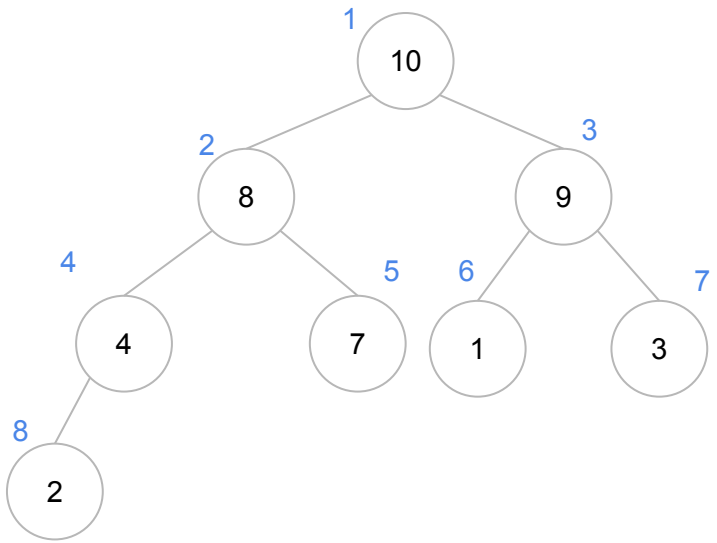



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
10	8	19	4	7	1	3	2	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8

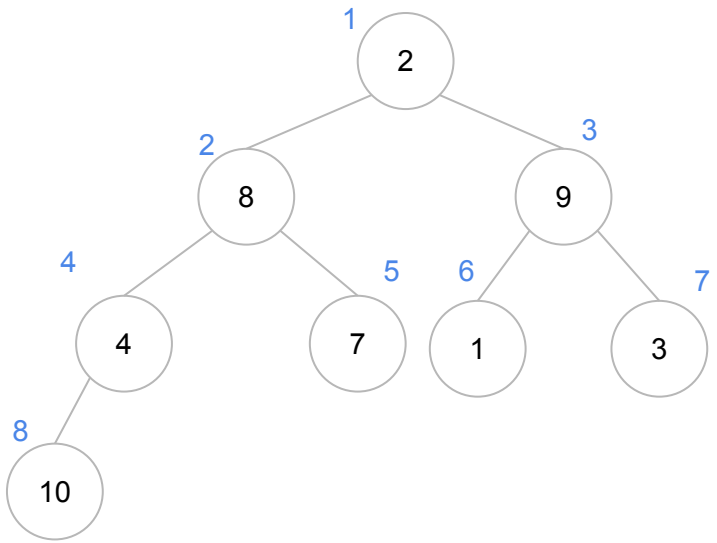



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
10	8	19	4	7	1	3	2	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8

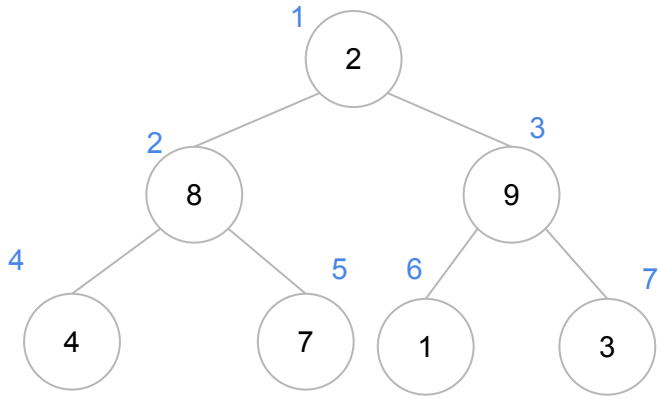



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
2	8	19	4	7	1	3	10	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7

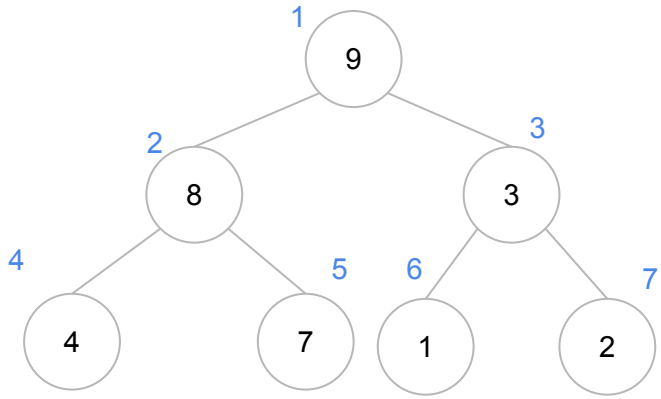



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
2	8	19	4	7	1	3	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7

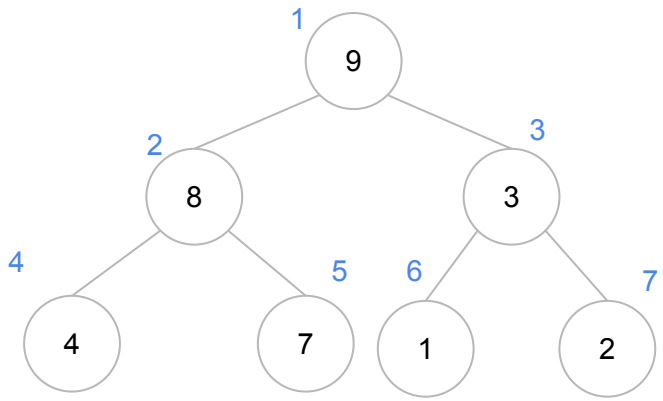



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
9	8	3	4	7	1	2	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
	6

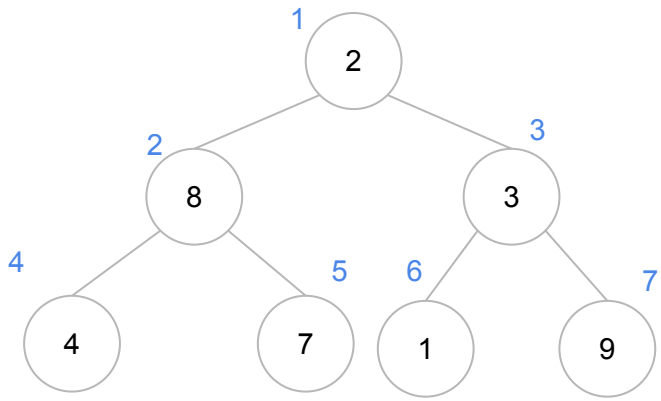



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
9	8	3	4	7	1	2	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
	6

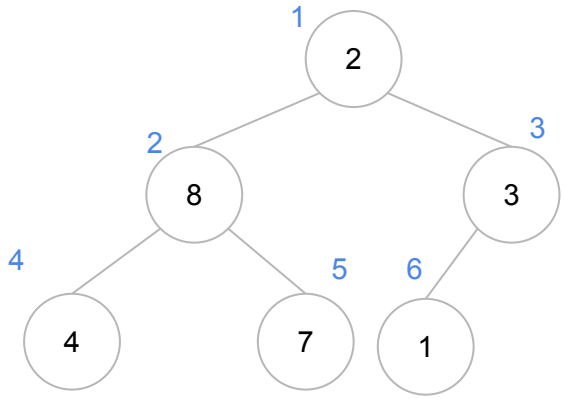



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
2	8	3	4	7	1	9	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6

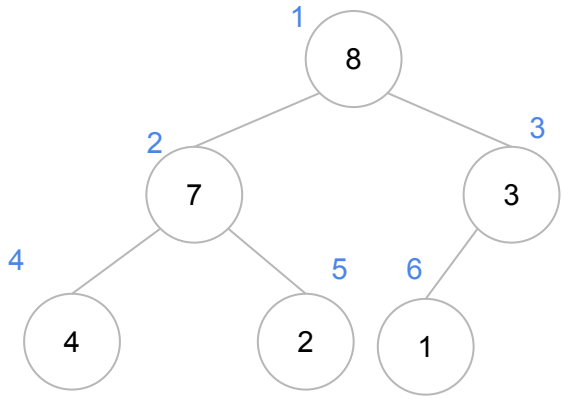



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
2	8	3	4	7	1	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6

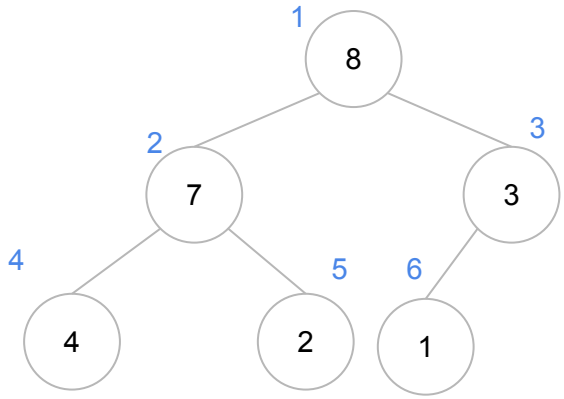



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
8	7	3	4	2	1	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
	5

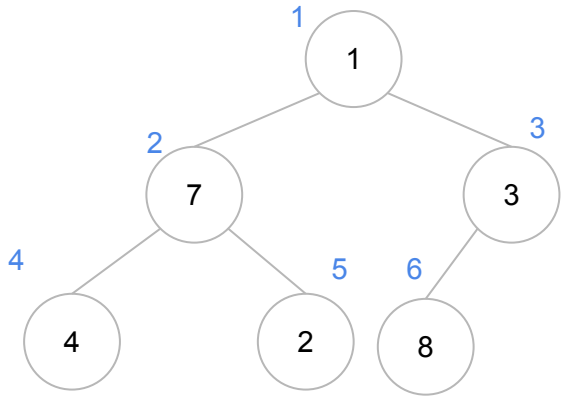



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
8	7	3	4	2	1	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
	5

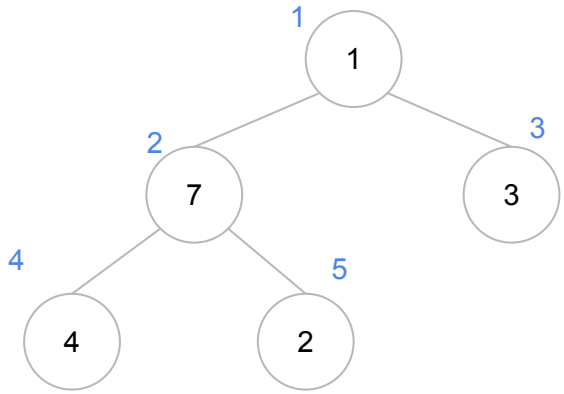



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	7	3	4	2	8	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5

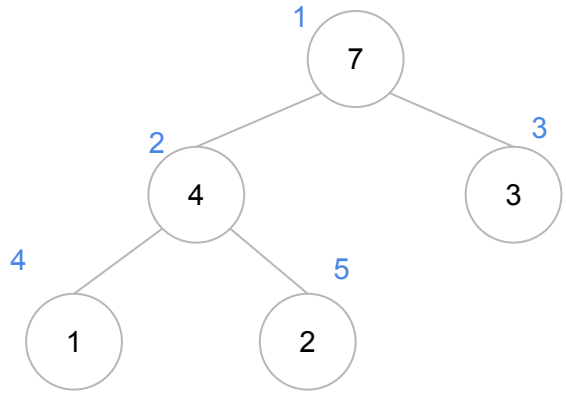



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	7	3	4	2	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5

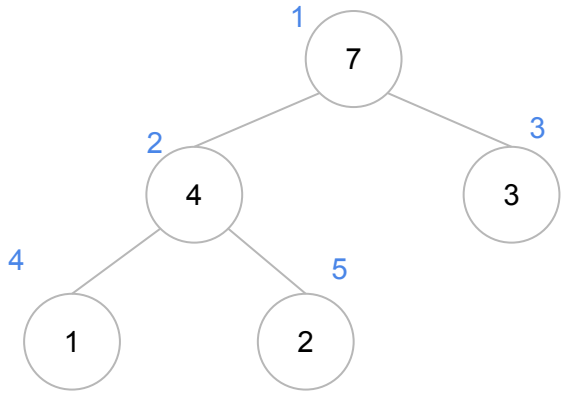



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
7	4	3	1	2	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
	4

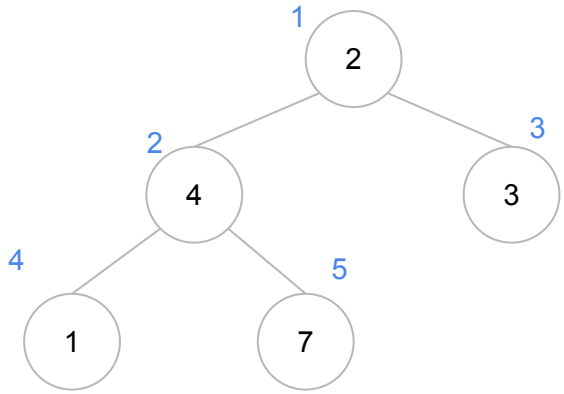



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
7	4	3	1	2	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
	4

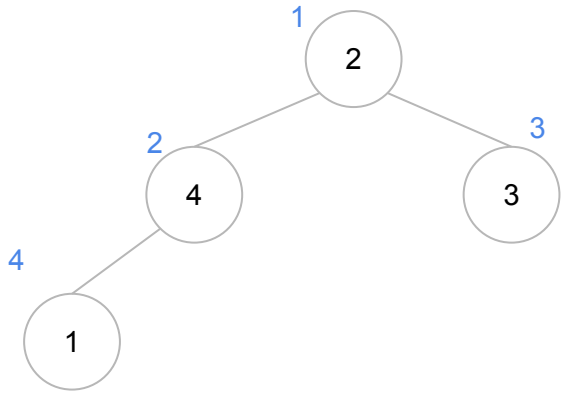



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
        trocar(v, 1, i)
        n ← n - 1
        max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
2	4	3	1	7	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4

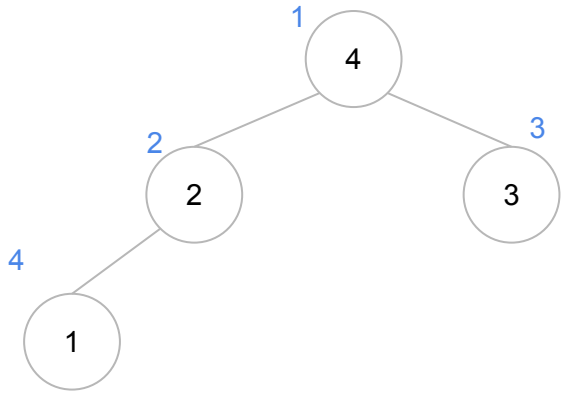



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
2	4	3	1	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4

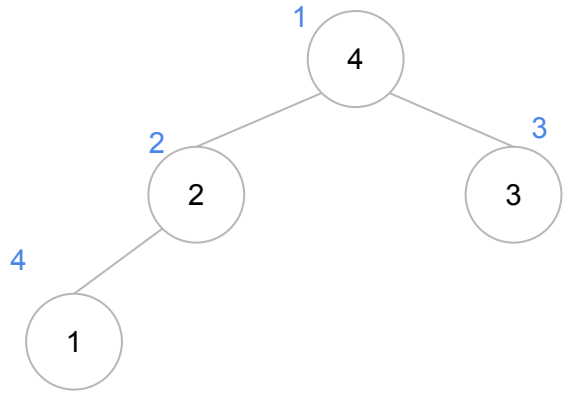



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
4	2	3	1	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
	3

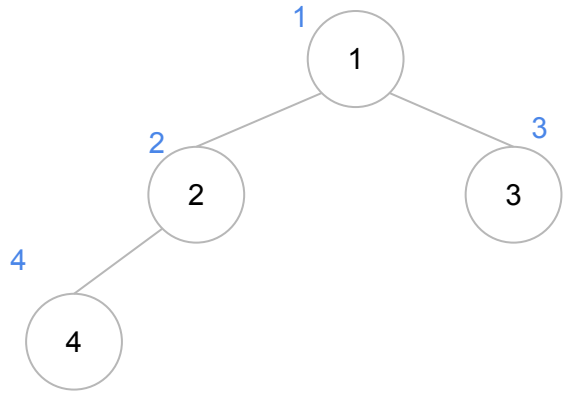



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
4	2	3	1	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
	3

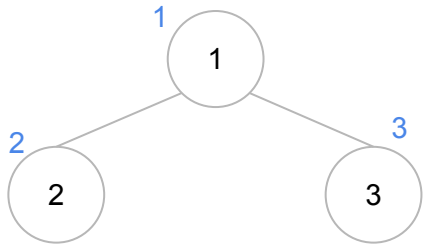



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3

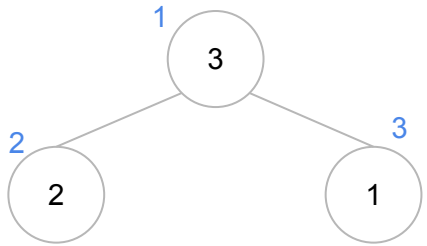



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3

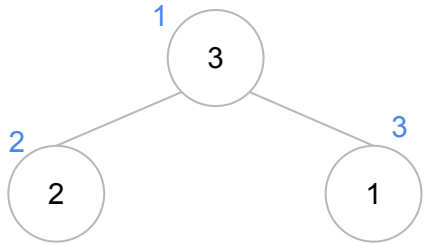



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
3	2	1	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3
	2

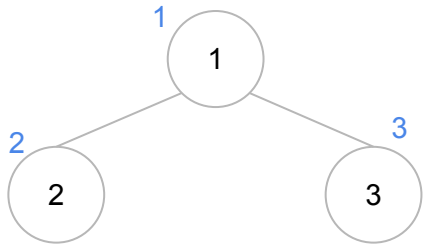



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
3	2	1	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3
	2

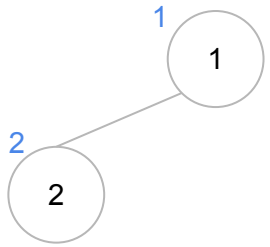



```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

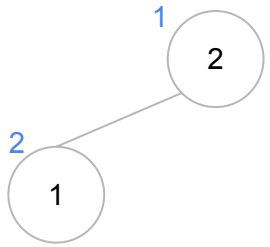
n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3
2	2




```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
1	2	**3**	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa



n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3
2	2

```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
2	1	**3**	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3
2	2

```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
2	**1**	**3**	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Teste de Mesa

n	i
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3
2	2

Note que da forma que foi construído, poderíamos eliminar *n ← n - 1* e chamar *max-heapify(v,1,i-1).*

```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

1	2	3	4	5	6	7	8	9	10
2	**1**	**3**	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Pergunta

Durante o teste de mesa, parece que o *max-heapify* sempre tinha algum trabalho a fazer (sempre precisava mudar algo de lugar)

Por que isso acontece?

```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
     trocar(v, 1, i)
     n ← n - 1
     max-heapify(v,1,n)
```

Análise

Considerando mais uma vez o número de comparações entre elementos do vetor

Da aula passada, o custo de *construir-max-heap* é *n* para o pior caso (e vamos considerar 0 para o melhor)

Então, no melhor caso:

$$C^{-}(n) = 0$$

```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
     trocar(v, 1, i)
     n ← n - 1
     max-heapify(v,1,n)
```

Análise

No pior caso, considerando o custo de max-heapify

$$C^{+}_{mh}(n) = 2\lfloor \log_2(n) \rfloor \approx 2\log_2(n)$$

Temos que

$$C^{+}(n) = n + 2\log_2(n-1) + 2\log_2(n-2) + 2\log_2(n-3) + \cdots + 2\log_2(2)$$

```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

Análise

Colocando em um somatório

$$C^{+}(n) = n + 2\sum_{i=2}^{n} \log_2 i$$

```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

Análise

Colocando em um somatório

$$C^+(n) = n + 2\sum_{i=2}^{n} \log_2 i$$

Lembrando-se que $log_b(xy) = log_b(x) + log_b(y)$

$$C^+(n) = n + 2\log_2((n-1)!)$$

```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

Análise

Pela aproximação de Stirling

$$C^+(n) = n + 2(n-1)\log_2(n-1) - 2(n-1)\log_2 e$$

Então

$$C^+(n) \approx 2n \log_2 n$$

Atenção: essa análise serve como uma base para o custo. Esse tipo de análise será detalhada na disciplina de Análise de Algoritmos e afins. Mas o custo realmente é **aproximadamente esse**.

```
função heapsort (v,n)
construir-max-heap(v,n)
para i ← n até 2 passo -1
      trocar(v, 1, i)
      n ← n - 1
      max-heapify(v,1,n)
```

https://en.wikipedia.org/wiki/Stirling%27s_approximation

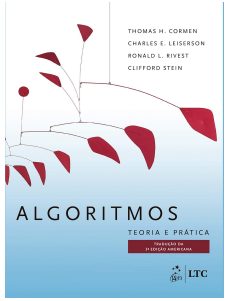
Exercícios

1. Implemente o algoritmo Heapsort em C.

Referências

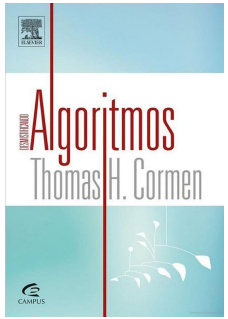
Renato Carmo. Algoritmos e Estruturas de Dados. www.inf.ufpr.br/renato

T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos: Teoria e Prática. 3a ed. 2012

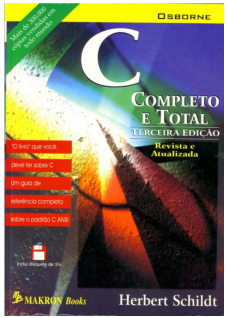


R. Sedgewick, K. Wayne. Algorithms Part I. 4a ed. 2014

T. Cormen. Desmistificando algoritmos. 2017.



H. Schildt. C completo e total. 1996



Licença