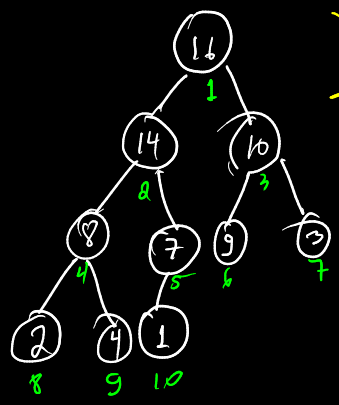
HeapSort

V = [16 14 10 8 7 9 3 2 4 1] p/ um nó i qualquer

(1 2 3 4 5 6 7 8 9 10)



Def Um vetor V é' Heap Máximo se $\forall i (i \neq 1 \rightarrow A[Pai(i)] \geq A[i])$.

$Esq(i) = 2i$

$Dir(i) = 2i + 1$

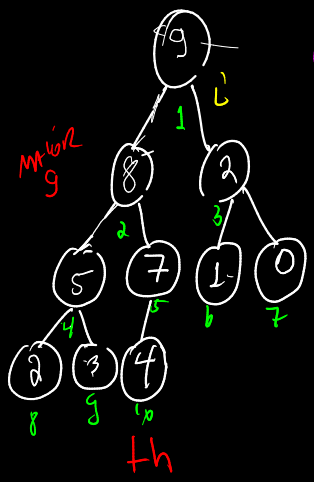
$Pai(i) = \lfloor \frac{i}{2} \rfloor$

"ARRUMA HEAP"

```
MAX-HEAPIFY(A, i, th)
1. l = ESQUERDO(i)
2. r = DIREITO(i)
3. IF l <= th AND A[l] > A[i]
4.     largest = l
5. ELSE largest = i
6. IF r <= th AND A[r] > A[largest]
7.     largest = r
8. IF largest <> i
9.     trocar A[i] e A[largest]
10.    MAX-HEAPIFY(A, largest, th)
```

$\theta(1)$

qual a posição do maior elemento entre A[i], a[l] a[r] (esq(i), DIR(i))



$T(n) \leq T(\frac{2n}{3}) + \theta(1)$

pelo M.M,

$a = 1$

$b = 3/2$

$f(n) = 1$

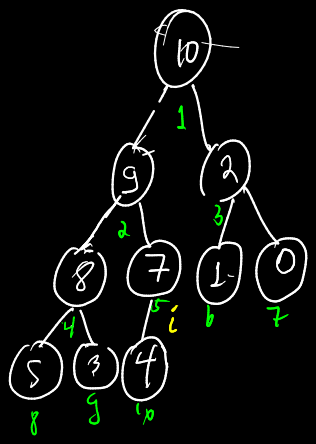
$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

$1 = \theta(1)$

pelo M.M., $T(n) = \theta(n^{\log_b a} \cdot \lg(n))$

$= \theta(\lg(n))$.

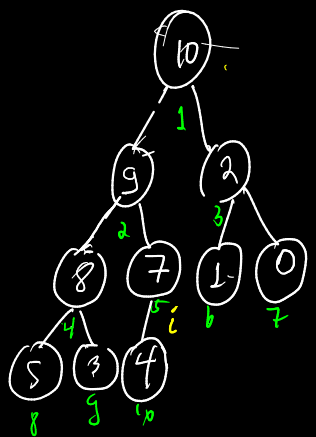
Análise Uma heap pode ser analisada como uma árvore binária quase completa. Portanto, ela tem altura $\lg(n)$. No pior caso, haverá uma recursão p/ cada nível da árvore, cada uma com custo $\theta(1)$. Como há $\lg(n)$ níveis na árvore, o custo do algoritmo é' $\lg(n) \cdot \theta(1) = \theta(\lg(n))$.




```
BUILD-MAX-HEAP(A, n)                i--
1. FOR i = CHAO(n/2) DOWNTO 1 DO
2.     MAX-HEAPIFY(A, i, n)
```

Invariante "No início de cada iteração do laço for todos os nós nas posições $i+1, i+2, \dots, n$ são raízes de heaps-máximos"

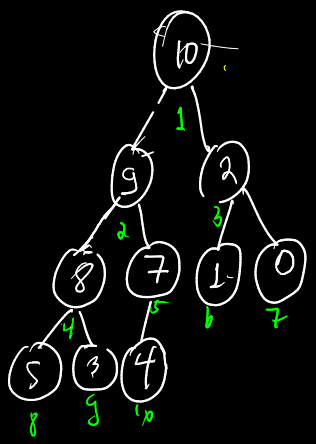
Inicialização Antes da primeira iteração, $i = \lfloor n/2 \rfloor$. Sabemos que o nó na posição $i = \lfloor n/2 \rfloor$ é o último nó que tem filhos. Logo, os nós nas posições $i+1, i+2, \dots, n$ são folhas. Como nós folhas são trivialmente raízes de heaps máximos, podemos concluir que os nós nas posições $i+1, i+2, \dots, n$ são raízes de heaps máximos. Assim, a invariante é verdadeira antes da primeira iteração.




```
BUILD-MAX-HEAP(A, n)                i--
1. FOR i = CHAO(n/2) DOWNTO 1 DO
2.     MAX-HEAPIFY(A, i, n)
```

Invariante "No início de cada iteração do laço for todos os nós nas posições $i+1, i+2, \dots, n$ são raízes de heaps-máximos"

Manutenção Pela invariante de laço os nós nas posições $i+1, i+2, i+3, \dots, n$ são raízes de heaps máximos. Como $\text{ESQ}(i) \geq i+1$ e $\text{DIR}(i) \geq i+1$, então sabemos que as subárvores enraizadas em $\text{ESQ}(i)$ e $\text{DIR}(i)$ são raízes de heaps máximos. Isto satisfaz o pré-requisito p/ invocar max-heapify, tornando i raiz de heap máximo. Portanto, após max-heapify, os elementos das posições $i, i+1, i+2, \dots, n$ são raízes de heaps máximos. Ao decrementar i no for, a invariante é restabelecida p/ a próxima iteração.




```
BUILD-MAX-HEAP(A, n)                    i--
1. FOR i = CHAO(n/2) DOWNTO 1 DO
2.     MAX-HEAPIFY(A, i, n)
```

INVARIANTE "No início de cada iteração do LAÇO FOR todos os nós nas posições $i+1, i+2, \ldots, n$ são raízes de heaps-máximos"

Término No término, $i = 0$. Pela invariante todos os nós nas posições $i+1, i+2, i+3, \ldots n = 1, 2, 3, \ldots, n$ são raízes de heaps máximos. Estes são todos os nós do vetor. Por definição, um vetor é heap máximo se todo nó é raiz de heap máximo. Isto é o que acabamos de mostrar e que é verdade ao final da execução de build-max-heap. Portanto, o algoritmo torna um vetor A qualquer em um um heap máximo. Isto indica que o algoritmo está correto!