Algoritmos e Estrutura de Dados II
(ESTCMP011)
2do PERIODO 2018

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Algoritmos de ordenação:
Heap_sort.

Prof. Luis Cuevas Rodríguez, PhD
E-mail: lcuevasrodriguez@gmail.com / lrodriguez@uea.edu.br
Celular: 9298154648

EST
Escola Superior de Tecnologia da UEA

NUCOMP
NÚCLEO DE COMPUTAÇÃO

Conteúdo

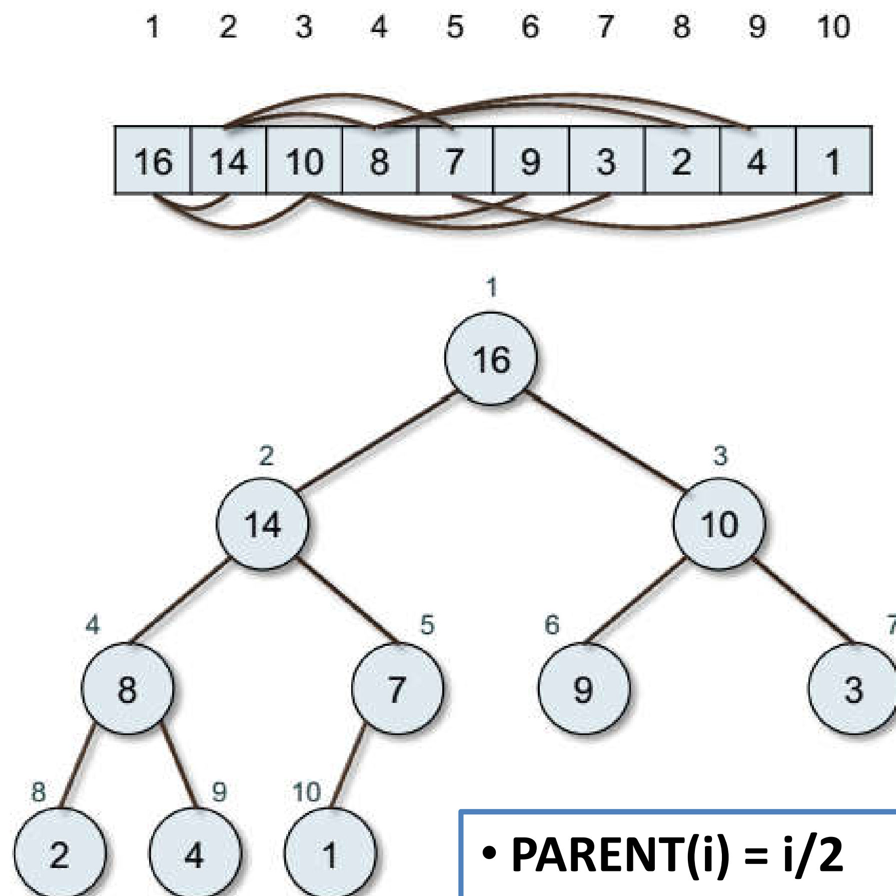
- Especificado o algoritmo de ordenação por:
  - Heapsort
- Analisar seu tempo de execução.

Heap_sort

- Combina os merge_sort e insertion_sort
  - ordena localmente
- Usa uma estrutura de dados tipo ***heap***.

Estrutura Heap

- Heap : Árvore binária
  - Cada nó da árvore corresponde a um elemento do arranjo .
  - Árvore completamente preenchida
- Atributos
  - Comprimento (A) → numero de elementos no arranjo
  - Tamanho_do_heap(A)→ numero de elementos no heap

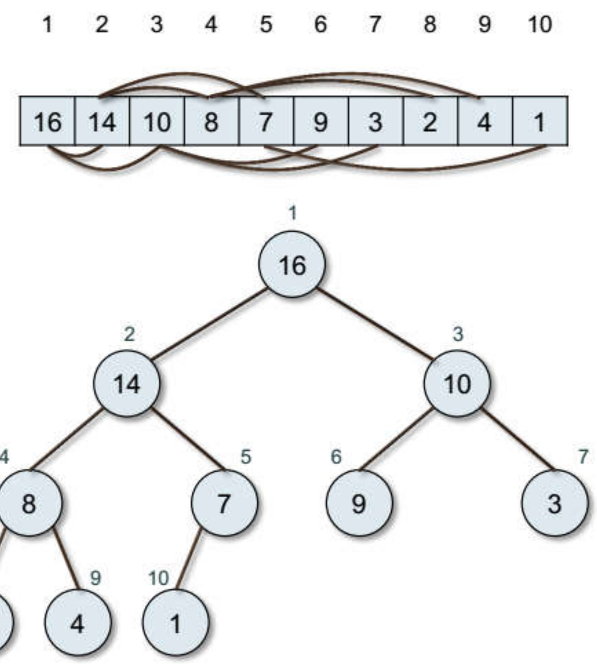


- **PARENT(i) = i/2**
- **LEFT(i) = 2i**
- **RIGHT(i) = 2i+1**

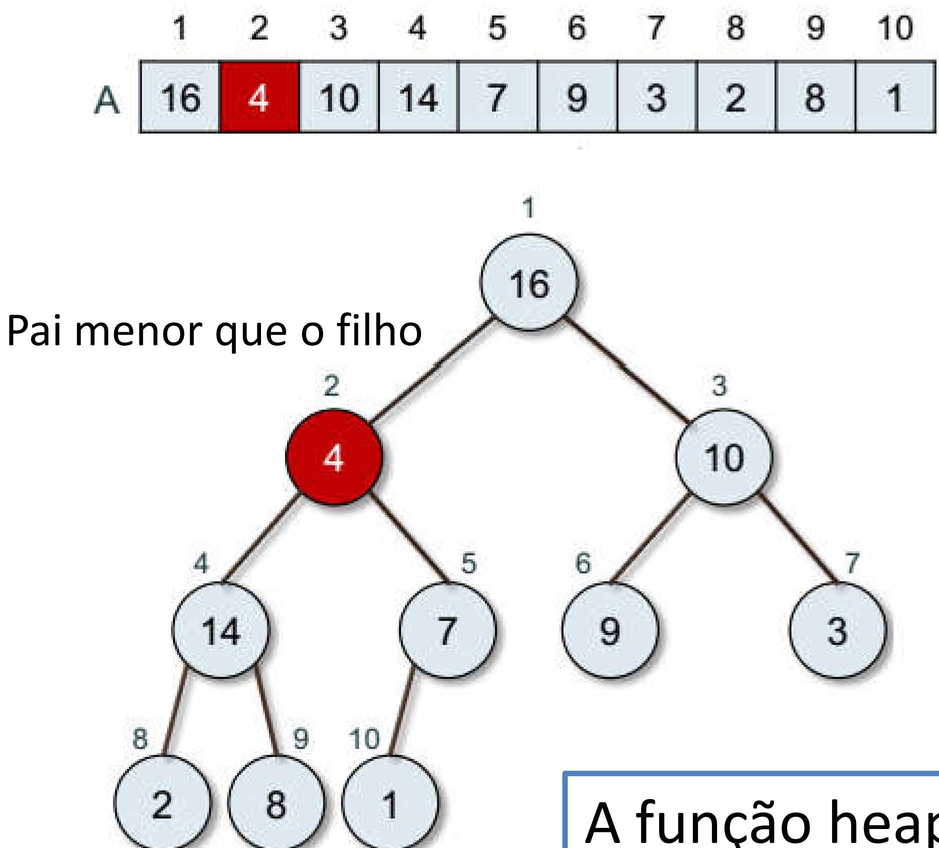
Heap_sort

- Propriedade de heap
  - heap máximo: **A[parent(i)] ≥ A[i]**
  - heap míximo: **A[parent(i)] ≤ A[i]**

Altura do nó: número de arestas no caminho descendente simples mais longo desde o nó ate uma folha
Altura do heap: como a altura de sua raiz = log n



Heap_sort



Funções para manter o heap

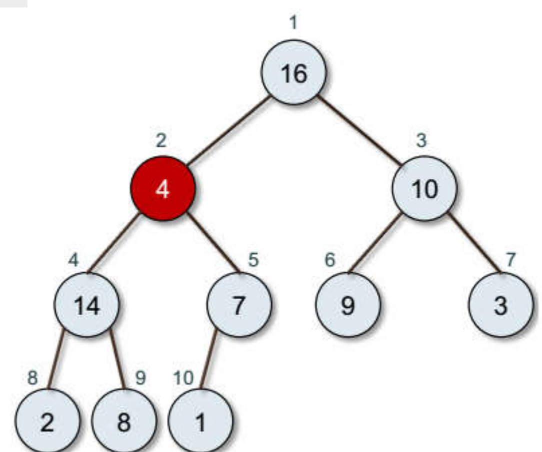
- **Heapify** recebe como parâmetro um array A e um índice i , e requer que:
  - As arvores binárias com raízes em left(i) e right(i) sejam heaps máximos

A função heapify deixa que o valor A[i] “flutue para baixo”, de maneira que a sub arvore com raiz no índice i se torne um heap

Heap_sort

```
HEAPFY(A,i)
   l ← LEFT(i);     // LEFT(i)  = 2i
   r ← RIGHT(i);    // RIGHT(i) = 2i + 1
   m ← i;
   if l ≤ tam-heap(A) & A[l]>A[i] then
     m ← l;
   end if
   if r ≤ tam-heap(A) & A[r]>A[m] then
     m ← r;
   end if
   if m ≠ i then
     trocar(A[i],A[m]);
     HEAPFY(A,m);
   end if
```

- **tam-heap()**: numero de elementos no heap
- HEAPFY é chamado sempre (começando da raiz até a folha mais distante)



Prof. Luis Cuevas Rodríguez, PhD

UNIVERSIDADE DO ESTADO DO AMAZONAS

Heap_sort

Função HEAPFY

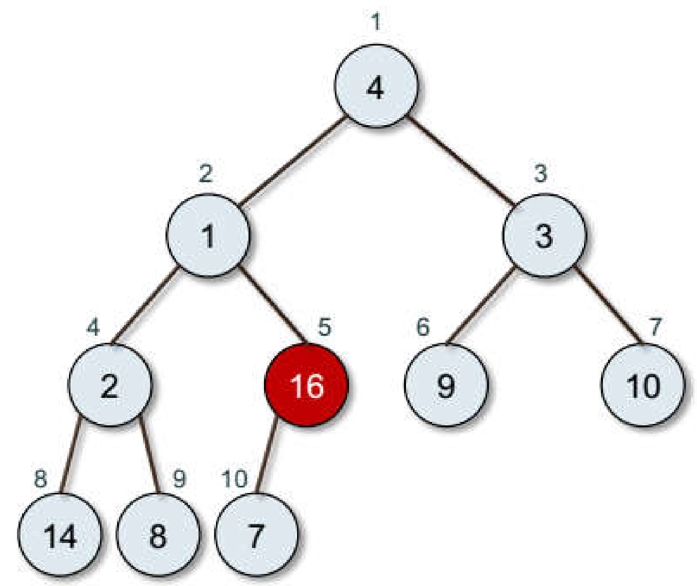
- **Melhor caso**, nossa árvore é um heap.
- No **pior caso** o número de trocas será a altura da árvore binária

Heap_sort

- Usar o procedimento HEAPFY para construir o heap de baixo para cima, para converter um arranjo qualquer A[1..*n*] em um heap

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

```
BUILD-HEAP(A,n)
  tam-heap(A) ← n;
  for i ← ⌊n/2⌋ downto 1 do
    HEAPFY(A,i);
  end for
```



Prof. Luis Cuevas Rodríguez, PhD

Heap_sort

```
BUILD-HEAP(A,n)
  tam-heap(A) ← n;
  for i ← ⌊n/2⌋ downto 1 do
     HEAPFY(A,i);
  end for
```

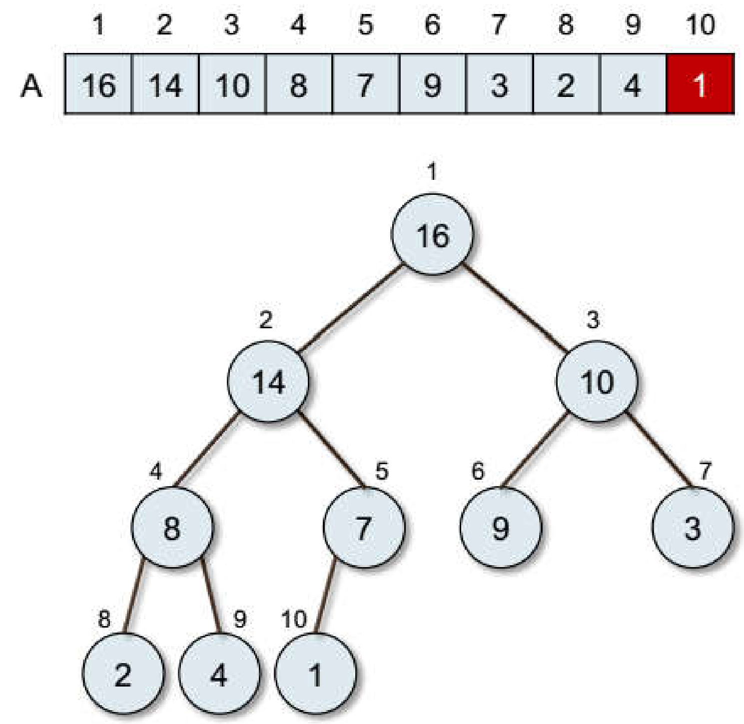
$\lfloor n/2 \rfloor$ vezes

$O(\log n)$

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

Heap_sort

```
HEAPSORT(A,n)
  BUILD-HEAP(A,n);
  for i ← n downto 2 do
     troca(A[1],A[i]);
     tam-heap(A)← tam-heap(A) - 1;
     HEAPFY(A,1);
  end for
```



1. Construir heap
2. Tocar ultimo elemento como o primeiro.
3. Manter heap.
4. Repetir 2

Heap_sort

`HEAPSORT(A,n)`	
`BUILD-HEAP(A,n);`	$O(n)$
`for i ← n downto 2 do`	n - 1 vezes
`troca(A[1],A[i]);`	$O(1)$
`tam-heap(A) ← tam-heap(A) - 1;`	
`HEAPFY(A,1);`	$O(\log n)$
`end for`	

$$T(n) = O(n) + (n-1)\big(O(1) + O(\log n)\big)$$

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

Exercícios

1. Um arranjo que está em seqüência ordenada é um heap mínimo?
2. A seqüência (23, 17, 14, 6, 13, 10, 1, 5, 7, 12) é um heap máximo?
3. Ilustre a operação HEAPIFY(A,3) sobre o arranjo A=(27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4,8,9,0)
4. Ilustre a operação BUILDHEAP sobre o arranjo A=(5, 3, 17, 10,84, 19,6,22,9)
5. Ilustre a operação HEAPSORT sobre o arranjo A=(5, 13, 2,25,7,17,20,8,4)