Algoritmos e Estruturas de Dados II

HeapSort

Prof. Tiago Eugenio de Melo
tmelo@uea.edu.br

www.tiagodemelo.info

Observações

- As palavras com a fonte `Courier` indicam as palavras-reservadas da linguagem de programação.

Referências

- **Algorithms in a Nutshell**. George T. Heineman, Gary Pollice, Stanley Selkow. O’Reilly Media, 2009.
- **Projetos de Algoritmos – com implementações em Pascal e C**. Nivio Ziviani. 2ª edição. Thomson, 2005.

HeapSort

Fila de Prioridades

Fila de Prioridades

Fila de Prioridades

- É uma fila onde cada elemento possui uma prioridade.

Fila de Prioridades

- É uma fila onde cada elemento possui uma prioridade.
- Essa prioridade determina a posição de um elemento na fila.

Fila de Prioridades

- É uma fila onde cada elemento possui uma prioridade.
- Essa prioridade determina a posição de um elemento na fila.
- Numa fila comum, o elemento do início sempre é removido (FIFO – *First In, First Out*).

Fila de Prioridades

- É uma fila onde cada elemento possui uma prioridade.
- Essa prioridade determina a posição de um elemento na fila.
- Numa fila comum, o elemento do início sempre é removido (FIFO – *First In, First Out*).
- Na fila com prioridades, o elemento removido é determinado pela prioridade.

Fila de Prioridades

- É uma fila onde cada elemento possui uma prioridade.
- Essa prioridade determina a posição de um elemento na fila.
- Numa fila comum, o elemento do início sempre é removido (FIFO – *First In, First Out*).
- Na fila com prioridades, o elemento removido é determinado pela prioridade.
- A fila de prioridades possui o critério de ordenação de acordo com a prioridade.

Fila de Prioridades

Fila de Prioridades

- Aplicações:

Fila de Prioridades

- Aplicações:
  - Fila de pacientes que aguardam por algum órgão.

Fila de Prioridades

- Aplicações:
  - Fila de pacientes que aguardam por algum órgão.
  - Caminhos mínimos (algoritmo de Dijkstra).

Fila de Prioridades

- Aplicações:
  - Fila de pacientes que aguardam por algum órgão.
  - Caminhos mínimos (algoritmo de Dijkstra).
  - Escalonamento de processos (SO).

Fila de Prioridades

Fila de Prioridades

- Existem vários tipos de implementação:

Fila de Prioridades

- Existem vários tipos de implementação:
  - Lista encadeada.

Fila de Prioridades

- Existem vários tipos de implementação:
  - Lista encadeada.
  - Heap.

Fila de Prioridades

- Existem vários tipos de implementação:
  - Lista encadeada.
  - Heap.
  - Lista desordenada.

Fila de Prioridades

- Existem vários tipos de implementação:
  - Lista encadeada.
  - Heap.
  - Lista desordenada.
  - Lista ordenada.

Fila de Prioridades

- Existem vários tipos de implementação:
  - Lista encadeada.
  - Heap.
  - Lista desordenada.
  - Lista ordenada.
- A escolha do tipo de implementação depende da aplicação.

Fila de Prioridades

Fila de Prioridades

- Representação inicial

Fila de Prioridades

- Representação inicial
  - Lista linear ordenada.

Fila de Prioridades

- Representação inicial
  - Lista linear ordenada.
    - Construir: $O(n \log n)$ ou $O(n^2)$

Fila de Prioridades

- Representação inicial
  - Lista linear ordenada.
    - Construir: O(n log n) ou O(n^2)
    - Inserir: O(n)

Fila de Prioridades

- Representação inicial
  - Lista linear ordenada.
    - Construir: O(n log n) ou O(n^2)
    - Inserir: O(n)
    - Retirar: O(1)

Fila de Prioridades

- Representação inicial
  - Lista linear ordenada.
    - Construir: O(n log n) ou O(n^2)
    - Inserir: O(n)
    - Retirar: O(1)
  - Lista linear não-ordenada

Fila de Prioridades

- Representação inicial
  - Lista linear ordenada.
    - Construir: O(n log n) ou O(n^2)
    - Inserir: O(n)
    - Retirar: O(1)
  - Lista linear não-ordenada
    - Construir: O(n)

Fila de Prioridades

- Representação inicial
  - Lista linear ordenada.
    - Construir: O(n log n) ou O(n^2)
    - Inserir: O(n)
    - Retirar: O(1)
  - Lista linear não-ordenada
    - Construir: O(n)
    - Inserir: O(1)

Fila de Prioridades

- Representação inicial
  - Lista linear ordenada.
    - Construir: O(n log n) ou O(n^2)
    - Inserir: O(n)
    - Retirar: O(1)
  - Lista linear não-ordenada
    - Construir: O(n)
    - Inserir: O(1)
    - Retirar: O(n)

Fila de Prioridades

- Representação inicial
  - Lista linear ordenada.
    - Construir: O(n log n) ou O(n^2)
    - Inserir: O(n)
    - Retirar: O(1)
  - Lista linear não-ordenada
    - Construir: O(n)
    - Inserir: O(1)
    - Retirar: O(n)
  - Outra alternativa: *heap*

Heap

HeapSort

HeapSort

- Algoritmo criado em 1964 por John Williams.

HeapSort

- Algoritmo criado em 1964 por John Williams.
- A intuição do *heap* é que este entende o vetor (lista) como uma **árvore binária**.

HeapSort

- Algoritmo criado em 1964 por John Williams.
- A intuição do *heap* é que este entende o vetor (lista) como uma **árvore binária**.
- Há dois tipos de *heap*:

HeapSort

- Algoritmo criado em 1964 por John Williams.
- A intuição do *heap* é que este entende o vetor (lista) como uma **<u>árvore binária</u>**.
- Há dois tipos de *heap*:
  - Max-heap

HeapSort

- Algoritmo criado em 1964 por John Williams.
- A intuição do *heap* é que este entende o vetor (lista) como uma **árvore binária**.
- Há dois tipos de *heap*:
  - Max-heap
  - Min-heap

HeapSort

- Algoritmo criado em 1964 por John Williams.
- A intuição do *heap* é que este entende o vetor (lista) como uma **árvore binária**.
- Há dois tipos de *heap*:
  - Max-heap
  - Min-heap
- Um *heap* é um vetor (lista) em que o valor de todo pai é maior ou igual ao valor de cada um de seus dois filhos.

HeapSort

- Exemplo:

50	40	30	20	25	10	15
1	2	3	4	5	6	7

HeapSort

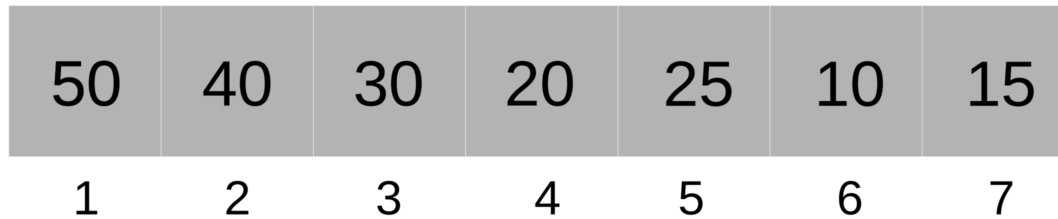
- Exemplo:

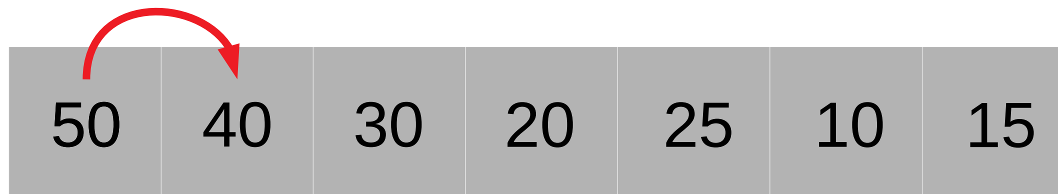
50	40	30	20	25	10	15
1	2	3	4	5	6	7

50	40	30	20	25	10	15

HeapSort

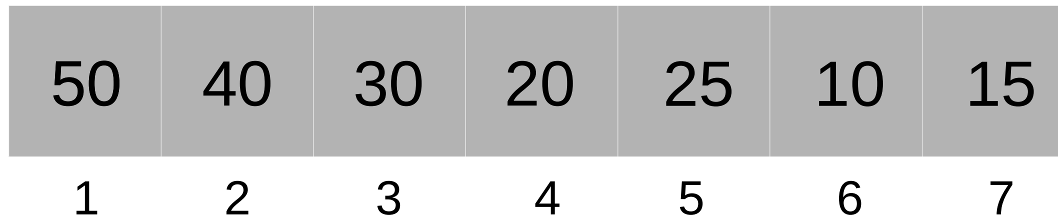
- Exemplo:

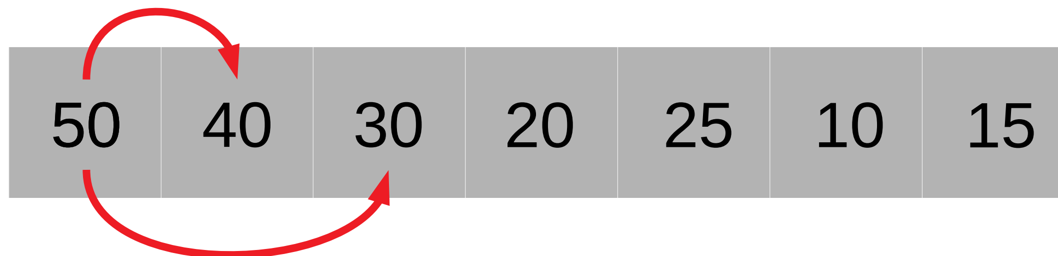




HeapSort

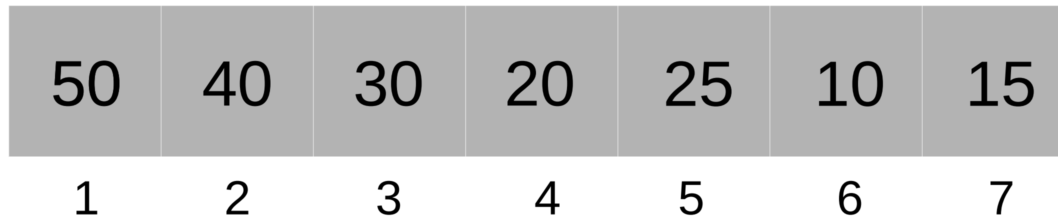
- Exemplo:

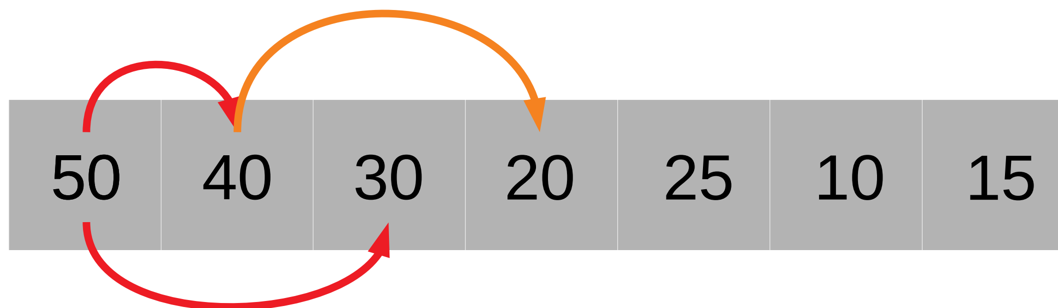




HeapSort

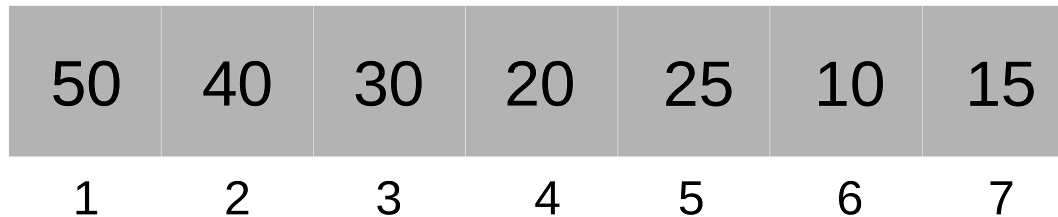
- Exemplo:

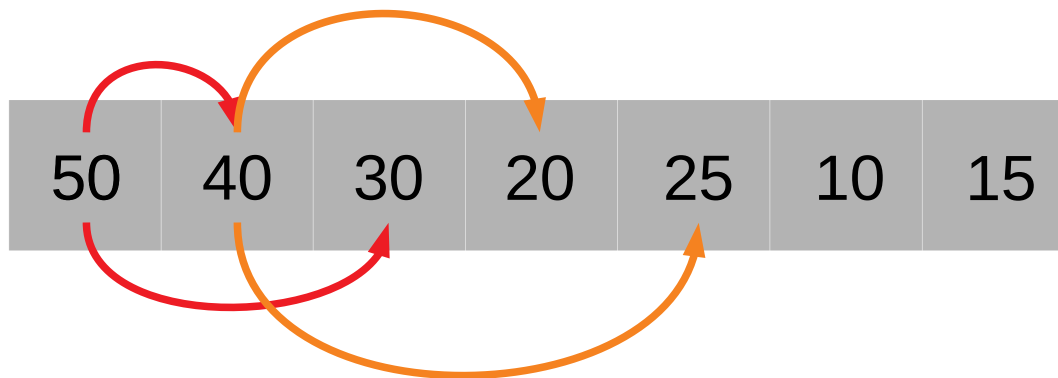




HeapSort

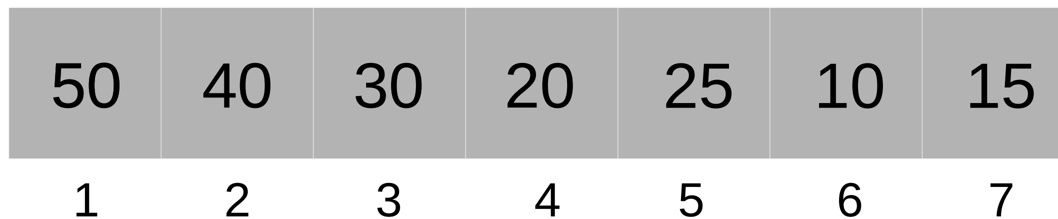
- Exemplo:

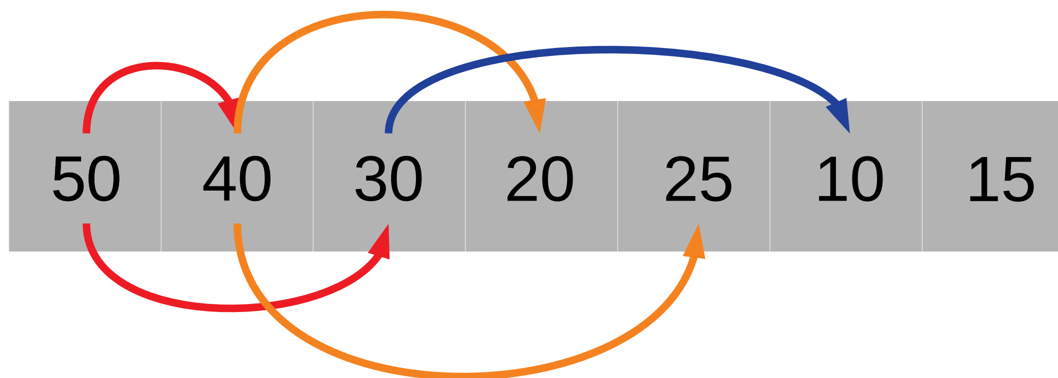




HeapSort

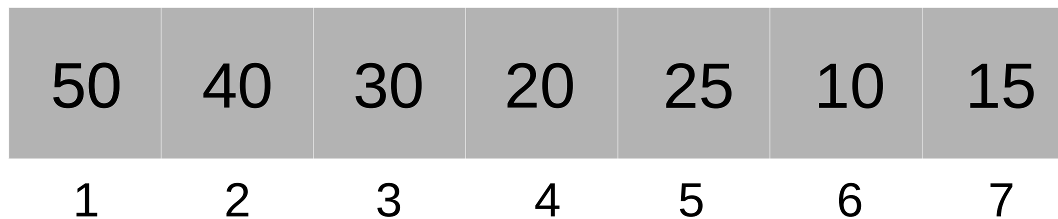
- Exemplo:

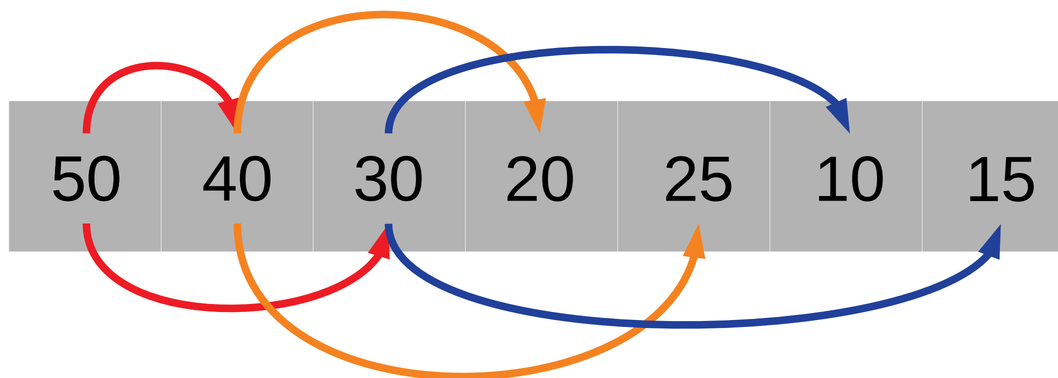




HeapSort

- Exemplo:



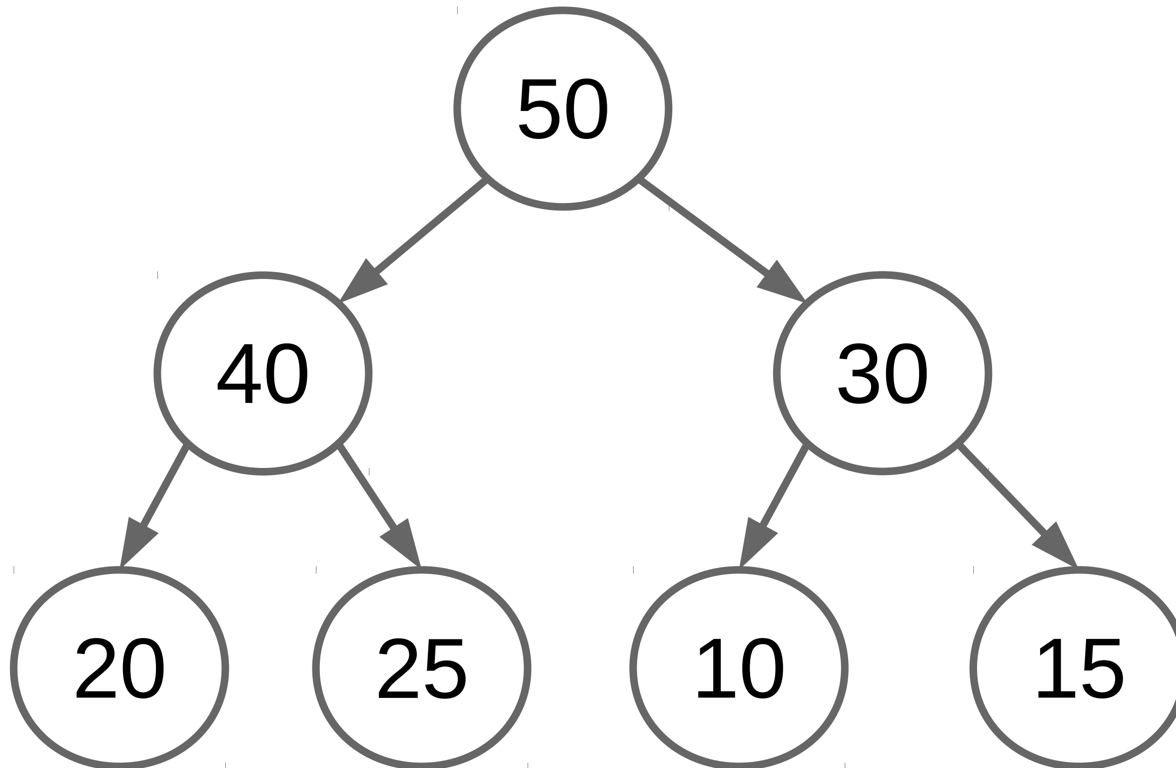


HeapSort

- Essa ordenação pode ser visualizada como uma árvore binária completa:

HeapSort

- Essa ordenação pode ser visualizada como uma árvore binária completa:



HeapSort

HeapSort

- Definição de *heap*:

HeapSort

- Definição de *heap*:
  - Sequência de elementos com chaves

HeapSort

- Definição de *heap*:
  - Sequência de elementos com chaves c[1], c[2], …, c[n]

HeapSort

- Definição de *heap*:
  - Sequência de elementos com chaves

    c[1], c[2], …, c[n]

    tal que

HeapSort

- Definição de *heap*:
  - Sequência de elementos com chaves

    c[1], c[2], …, c[n]

    tal que

    c[i] >= c[2.i],

HeapSort

- Definição de *heap*:
  - Sequência de elementos com chaves

    c[1], c[2], …, c[n]

    tal que

    c[i] >= c[2.i],

    c[i] >= c[2.i + 1],

HeapSort

- Definição de *heap*:
  - Sequência de elementos com chaves

    c[1], c[2], …, c[n]

    tal que

    c[i] >= c[2.i],

    c[i] >= c[2.i + 1],

    para todo i = 1, 2, …, n/2.

HeapSort

HeapSort

- Portanto, um *heap* é uma árvore binária completa na qual cada nó satisfaz a condição do *heap* apresentada na sua definição.

HeapSort

- Portanto, um *heap* é uma árvore binária completa na qual cada nó satisfaz a condição do *heap* apresentada na sua definição.
- Os elementos no *heap* não estão perfeitamente ordenados.

HeapSort

- Portanto, um *heap* é uma árvore binária completa na qual cada nó satisfaz a condição do *heap* apresentada na sua definição.
- Os elementos no *heap* não estão perfeitamente ordenados.
- Sabe-se apenas que o maior elemento está no nó raiz e que, para cada nó, todos os seus descendentes não são maiores que o elemento raiz.

HeapSort

HeapSort

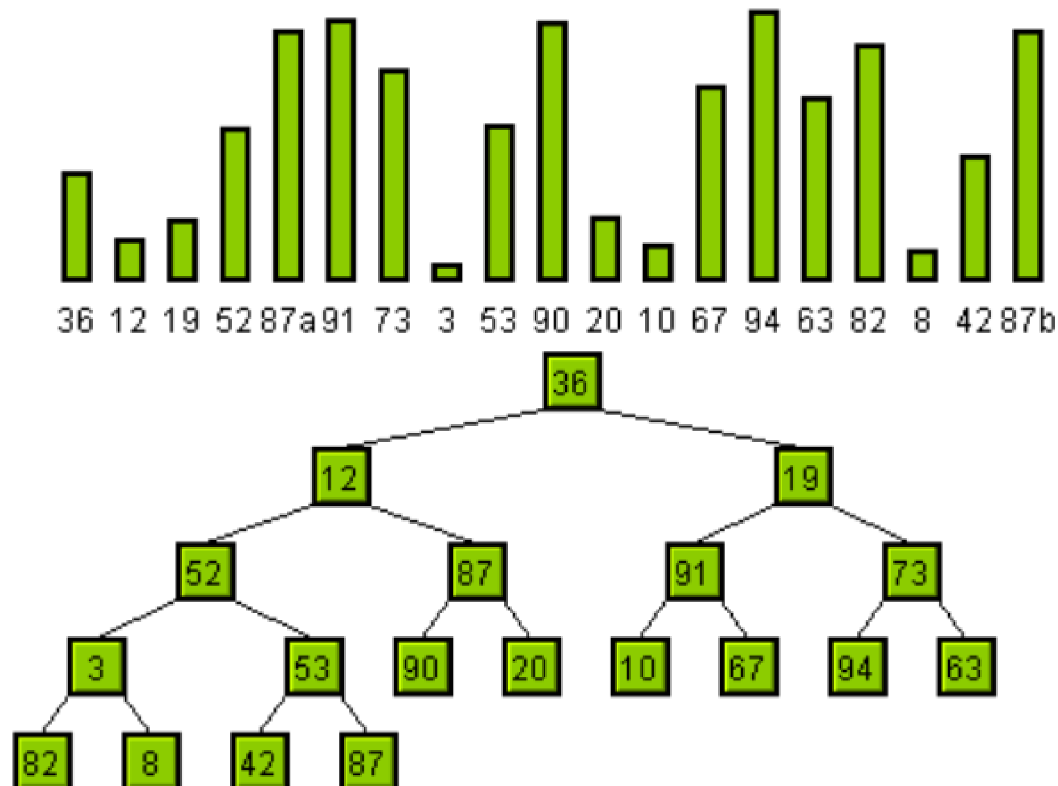
- Tenta-se evitar a utilização real de uma árvore.

HeapSort

- Tenta-se evitar a utilização real de uma árvore.
- A ideia é utilizar a abordagem de *heap* representando uma árvore como uma lista.

HeapSort

- Tenta-se evitar a utilização real de uma árvore.
- A ideia é utilizar a abordagem de *heap* representando uma árvore como uma lista.



HeapSort

HeapSort

- Por que usar um *heap* é importante?

HeapSort

- Por que usar um *heap* é importante?
  - Descobrir quem é o maior elemento é $O(1)$.

HeapSort

- Por que usar um *heap* é importante?
  - Descobrir quem é o maior elemento é O(1).
  - Se o valor da raiz for alterado, o heap pode ser refeito rapidamente O(log n).

HeapSort

HeapSort

- Algoritmo

HeapSort

- Algoritmo
  - Transformar uma lista L em um heap H.

HeapSort

- Algoritmo
  - Transformar uma lista L em um heap H.
  - Pegar o maior elemento de H e trocar com a posição L[máx].

HeapSort

- Algoritmo
  - Transformar uma lista L em um heap H.
  - Pegar o maior elemento de H e trocar com a posição L[máx].
  - Repetir os passos anteriores com os demais elementos.

HeapSort

HeapSort

- Exemplo:

HeapSort

- Exemplo:
  - Considere o conjunto de valores abaixo:

HeapSort

- Exemplo:
  - Considere o conjunto de valores abaixo:

    82, 90, 10, 12, 15, 77, 55, 23

HeapSort

- Exemplo:
  - Considere o conjunto de valores abaixo:

    82, 90, 10, 12, 15, 77, 55, 23

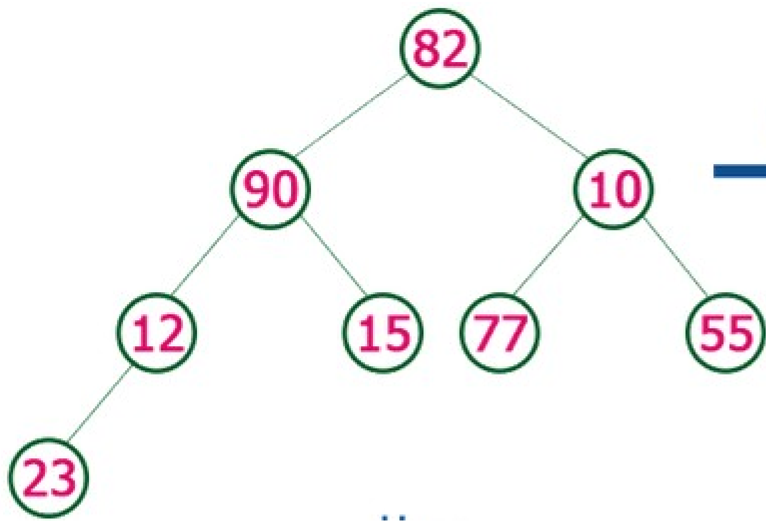
  - Passo 1 (Max-*heapfy*):

HeapSort

- Exemplo:
  - Considere o conjunto de valores abaixo:

    82, 90, 10, 12, 15, 77, 55, 23

  - Passo 1 (Max-*heapfy*):



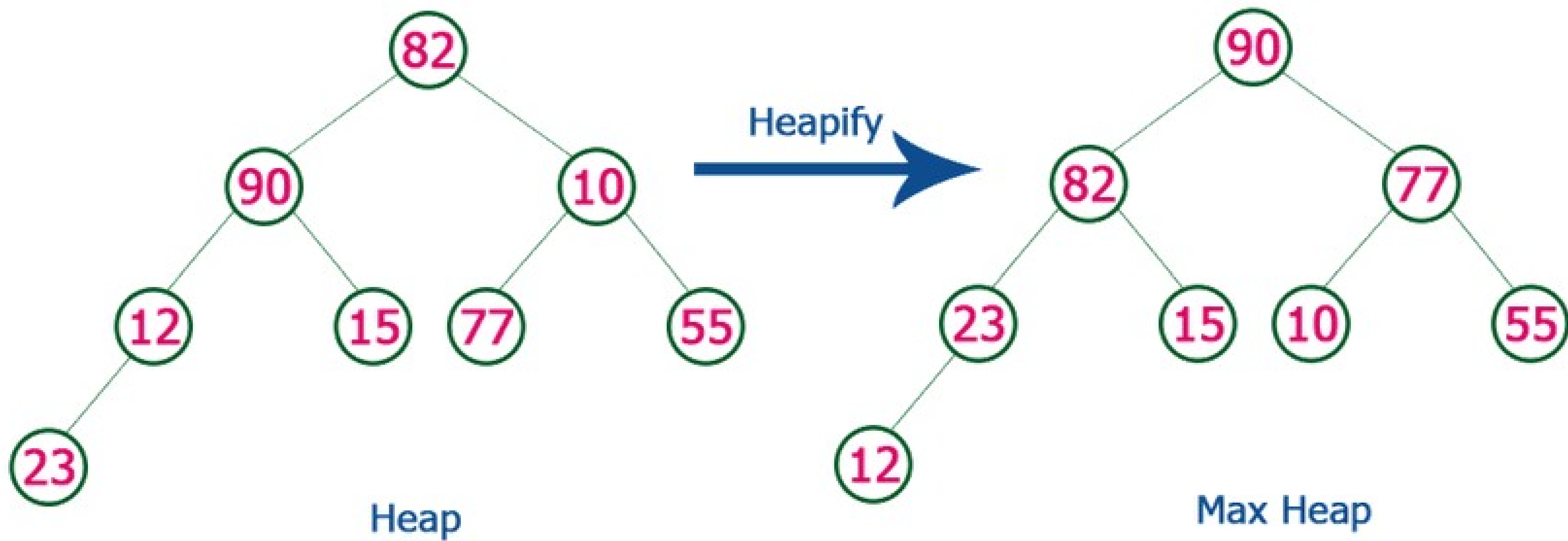
HeapSort

HeapSort

- Exemplo (cont.):

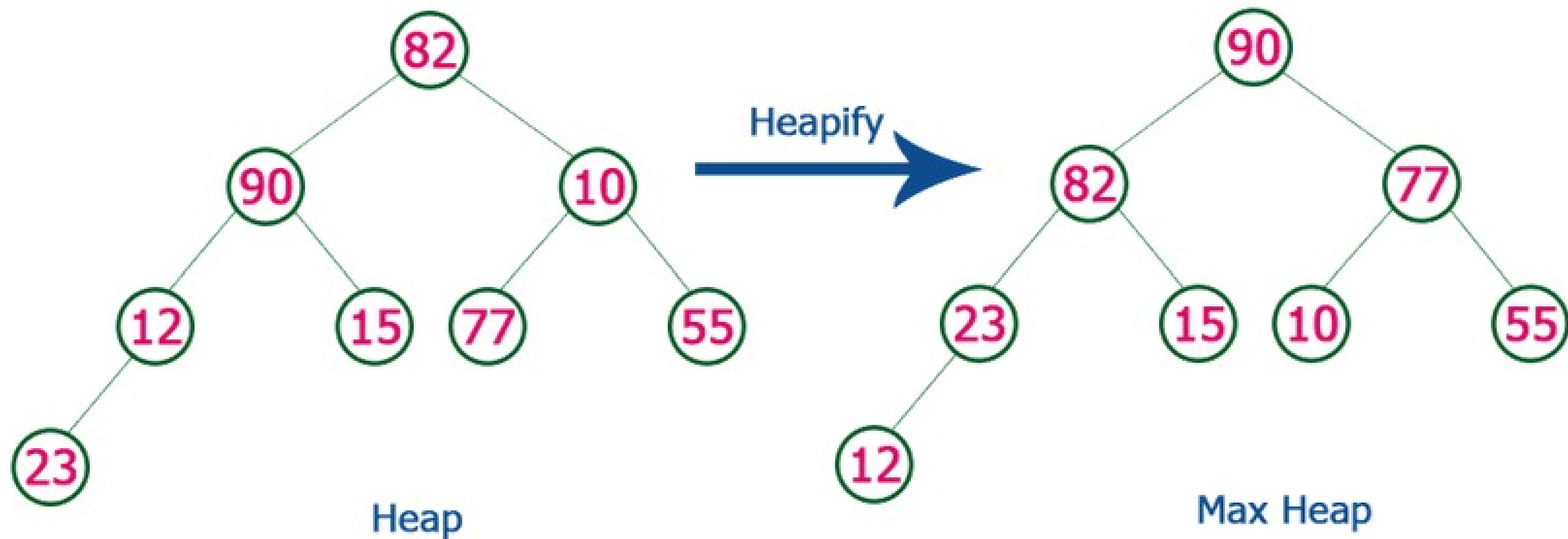
HeapSort

- Exemplo (cont.):



HeapSort

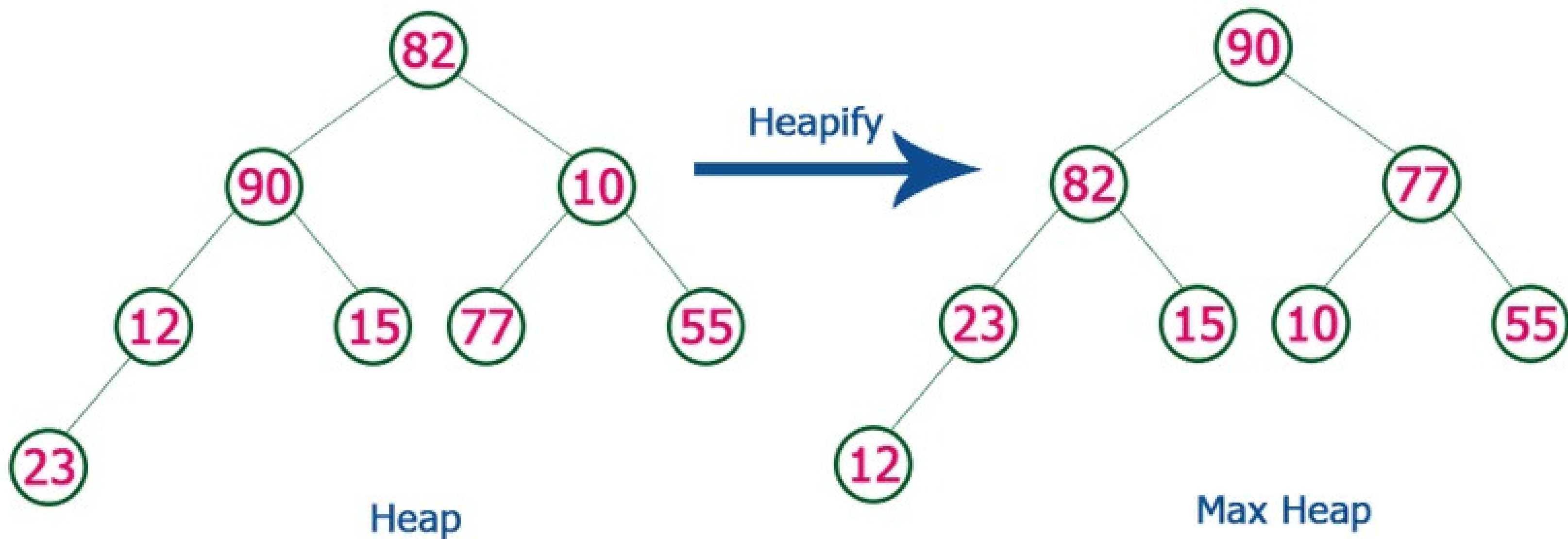
- Exemplo (cont.):



  - Resultado:

HeapSort

- Exemplo (cont.):



  - Resultado:

90, 82, 77, 23, 15, 10, 55, 12

HeapSort

HeapSort

- Análise:

HeapSort

- Análise:
  - O algoritmo aparenta não ser eficiente.

HeapSort

- Análise:
  - O algoritmo aparenta não ser eficiente.
  - Porém, refazer o *heap* gasta cerca de **`log n`** operações (pior caso).

HeapSort

- Análise:
  - O algoritmo aparenta não ser eficiente.
  - Porém, refazer o *heap* gasta cerca de **`log n`** operações (pior caso).
  - Portanto, *heapsort* é proporcional a **`n log n`** no pior caso.

HeapSort

- Análise:
  - O algoritmo aparenta não ser eficiente.
  - Porém, refazer o *heap* gasta cerca de **`log n`** operações (pior caso).
  - Portanto, *heapsort* é proporcional a **`n log n`** no pior caso.
  - O Quicksort é, em média, cerca de duas vezes mais rápido que o HeapSort.

HeapSort

- Análise:
  - O algoritmo aparenta não ser eficiente.
  - Porém, refazer o *heap* gasta cerca de **`log n`** operações (pior caso).
  - Portanto, *heapsort* é proporcional a **`n log n`** no pior caso.
  - O Quicksort é, em média, cerca de duas vezes mais rápido que o HeapSort.
  - HeapSort é melhor que o ShellSort para grandes arquivos.

HeapSort

HeapSort

- Pontos positivos

HeapSort

- Pontos positivos
  - O(**`n log n`**) qualquer que seja a entrada.

HeapSort

- Pontos positivos
  - O(**`n log n`**) qualquer que seja a entrada.
- Pontos negativos

HeapSort

- Pontos positivos
  - O(**`n log n`**) qualquer que seja a entrada.
- Pontos negativos
  - Método não estável.

Exercícios

- Os elementos em um heap não estão perfeitamente ordenados. Porém, é possível seguir os elementos de maneira sequencial ordenada. Como isso é possível?
- Qual é a diferença entre heap máximo e heap mínimo? Existe diferença de eficiência entre eles?
- Implemente em Python uma função para reconstruir o heap, de tal maneira que as suas propriedades sejam mantidas.

Exercícios

- O vetor 161 41 101 141 71 91 31 21 81 17 16 é um *heap*?
- Mostre um exemplo que o algoritmo heap sort não é estável.
- Dado o seguinte conjunto de elementos `X={8, 6, 7, 4, 5, 3, 2, 1}`, desenhe o heap (árvore).