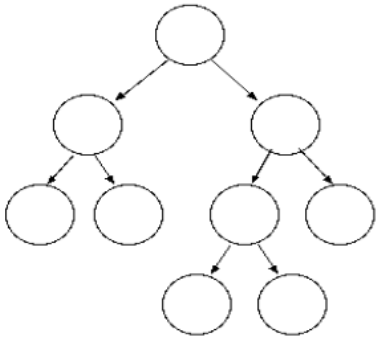
Estrutura de Dados Avançada

Daniel de Sousa Moraes
danielmoraes14@gmail.com

Heap Sort

- A ordenação se baseia na estrutura de dados Heap
  - Heap é um vetor que pode ser visto como uma árvore binária completa.
    - Uma árvore é dita completa se todo nível i, com exceção do último, tem o número máximo de elementos.

  - Cada nó da árvore equivale a um valor no vetor



Árvore binária completa

Primitivas da Heap

- lenght(V)
  - Número de elementos no vetor V

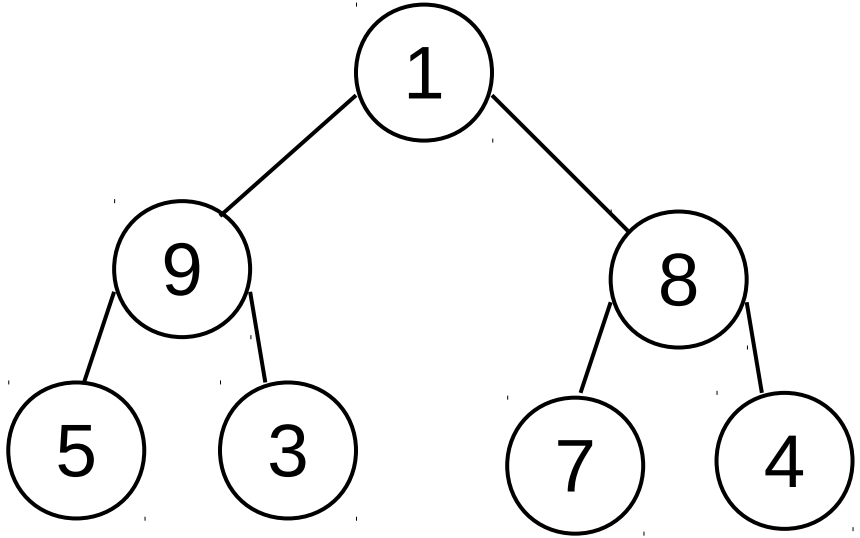
- heap-size(V)
  - Número de elementos da *heap* armazenados no vetor V

Vetor Heap – primitivas de árvore

- parent(i)
  - Return i/2
- left(i)
  - Return 2i
- right(i)
  - Return 2i+1

Vetor Heap – primitivas de árvore

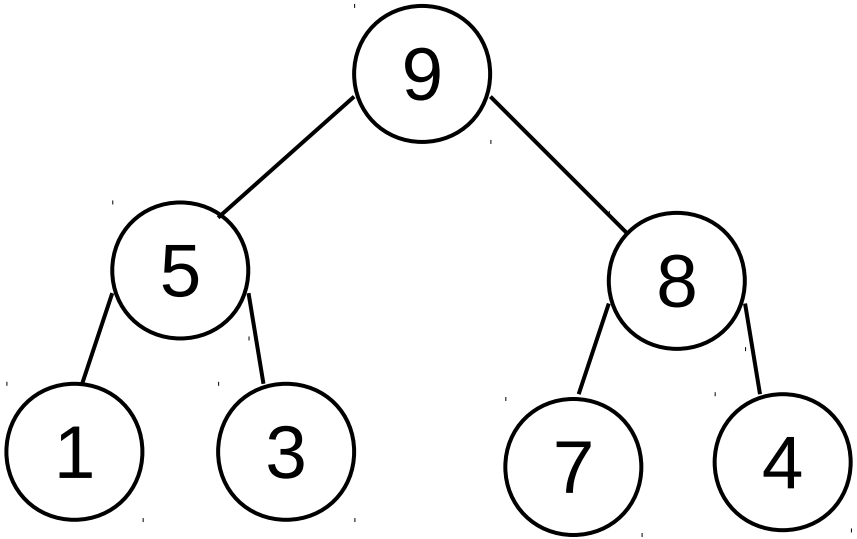
1	9	8	5	3	7	4



Propriedade Heap

- Para cada nó i que não seja a raiz, então:
  - V[parent(i)] >= V[i]

- Exemplo:



Primitiva Heapify

- Função para manter a propriedade da heap

```
function heapify(V, i)
    l = left(i), r = right(i)
    if l<=heap-size(V) and V[l]>V[i] then largest = l
    else largest = i end
    if r<=heap-size(V) and V[r]>V[largest] then
       largest=r end
    if largest != i then
       V[i], V[largest] = V[largest], V[i]
       heapify(V, largest)
    end
end
```

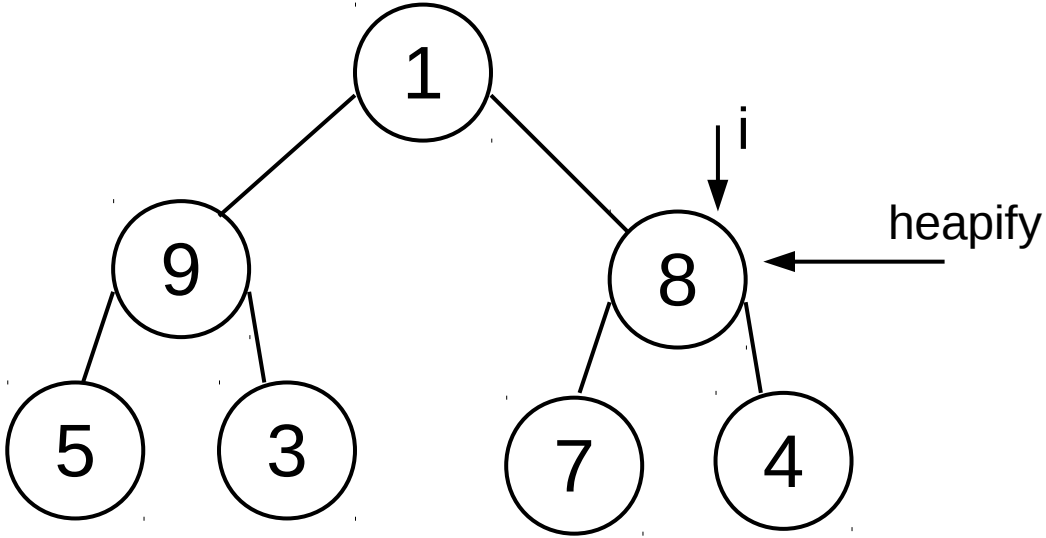
Construindo uma Heap

- Primitiva que constrói uma heap a partir de um vetor de N elementos.

```
function build_heap(V)
   heap-size(V) = length(V)
   for (i=(lenght(V)/2)+1; i>0; i--)
      heapify(V,i)
   end
end
```

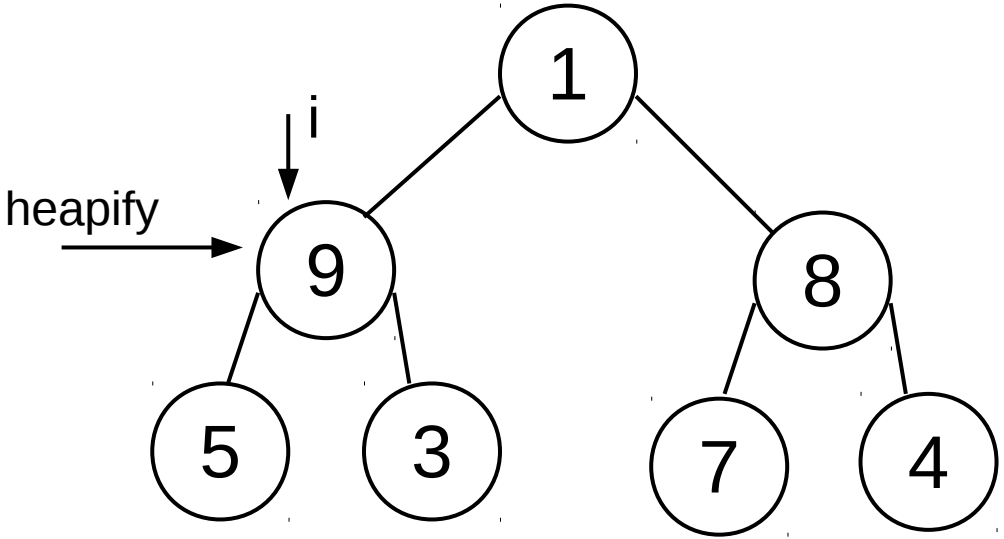
Construindo uma Heap

1	9	8	5	3	7	4



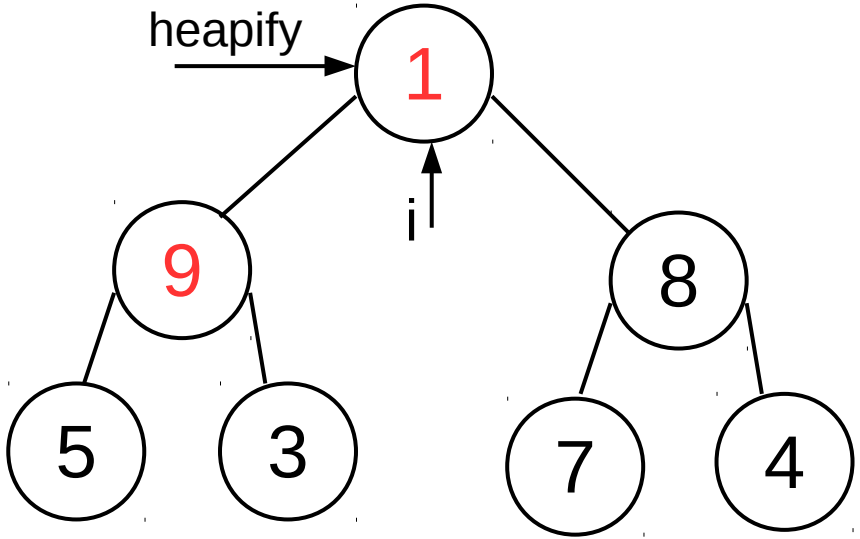
Construindo uma Heap

1	9	8	5	3	7	4



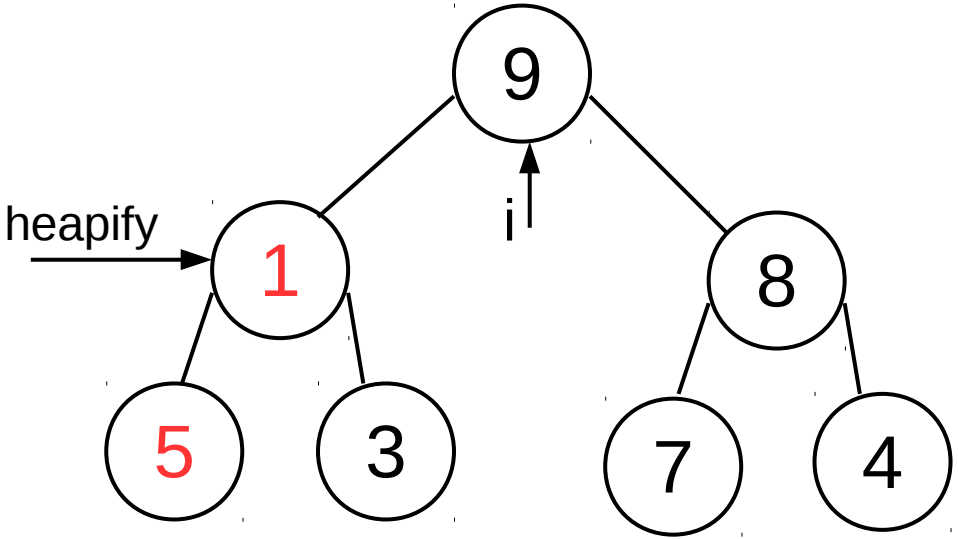
Construindo uma Heap

1	9	8	5	3	7	4



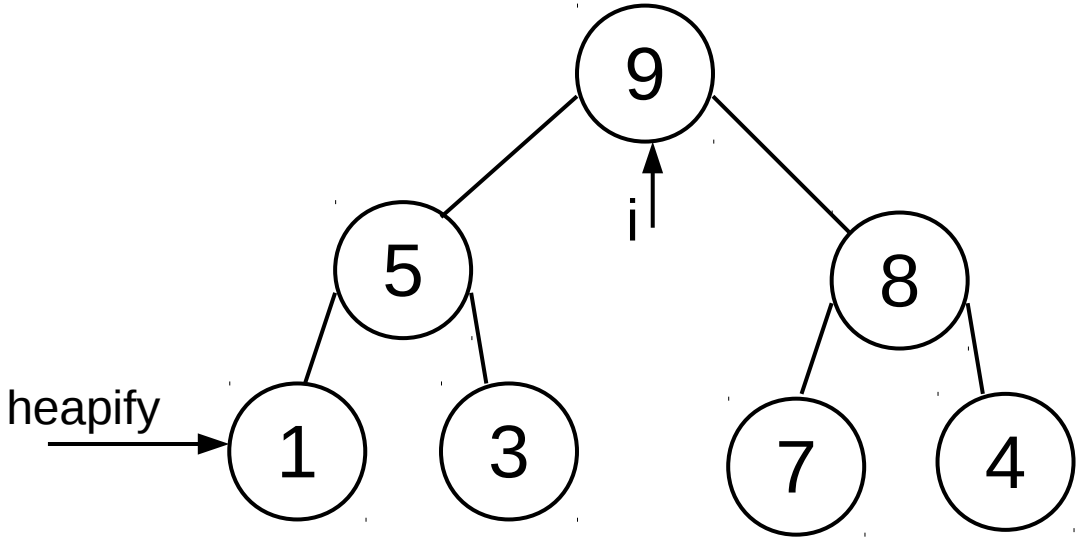
Construindo uma Heap

9	1	8	5	3	7	4



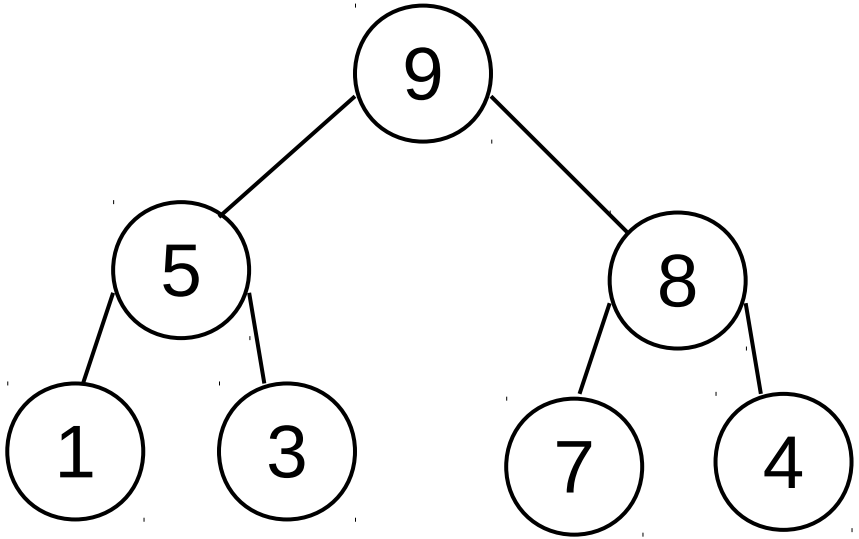
Construindo uma Heap

9	5	8	1	3	7	4



Heap Sort

9	5	8	1	3	7	4



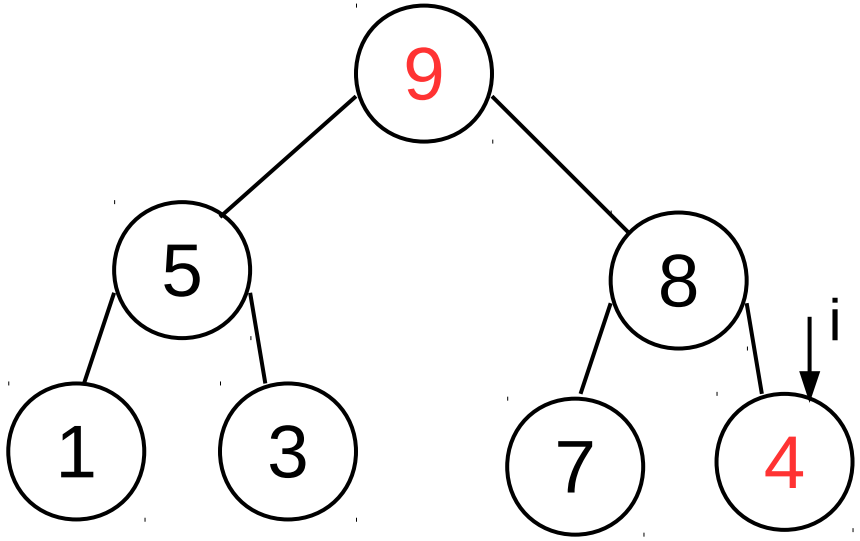
Heap Sort

- Primitiva que constrói uma heap a partir de um vetor de N elementos.

```
function heapsort(V)
     for (i=length(V)-1; i>=2,-1) do
        V[0], V[i] = V[i], V[0]
        heap-size(V)--
        heapify(V,0)
     end
end
```

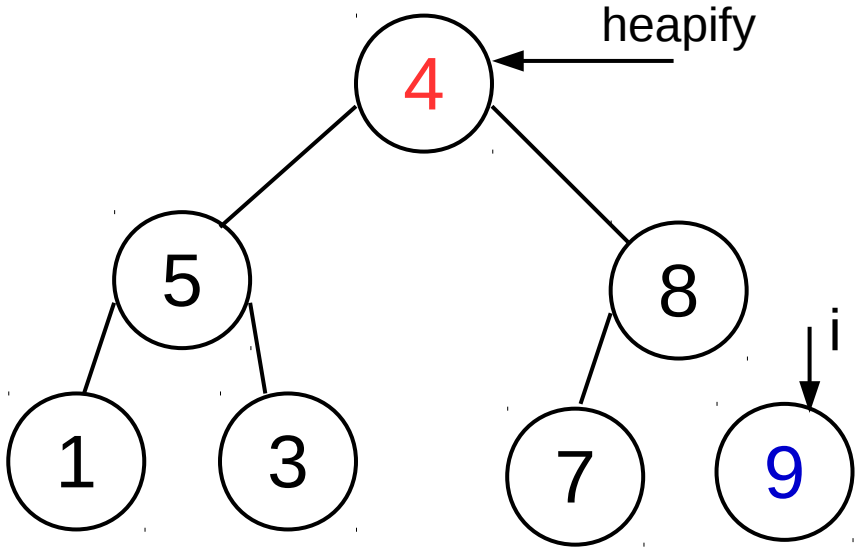
Heap Sort

9	5	8	1	3	7	4



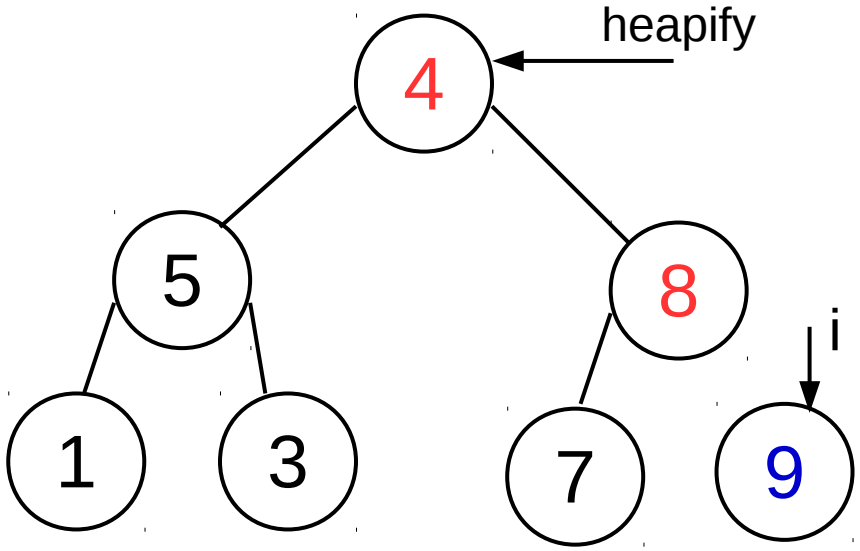
Heap Sort

4	5	8	1	3	7	9



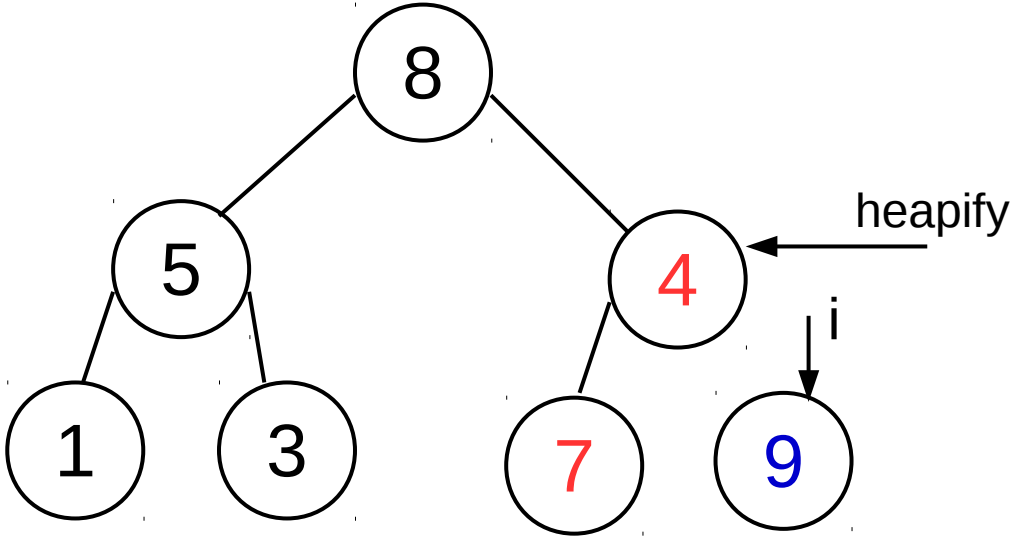
Heap Sort

4	5	8	1	3	7	9



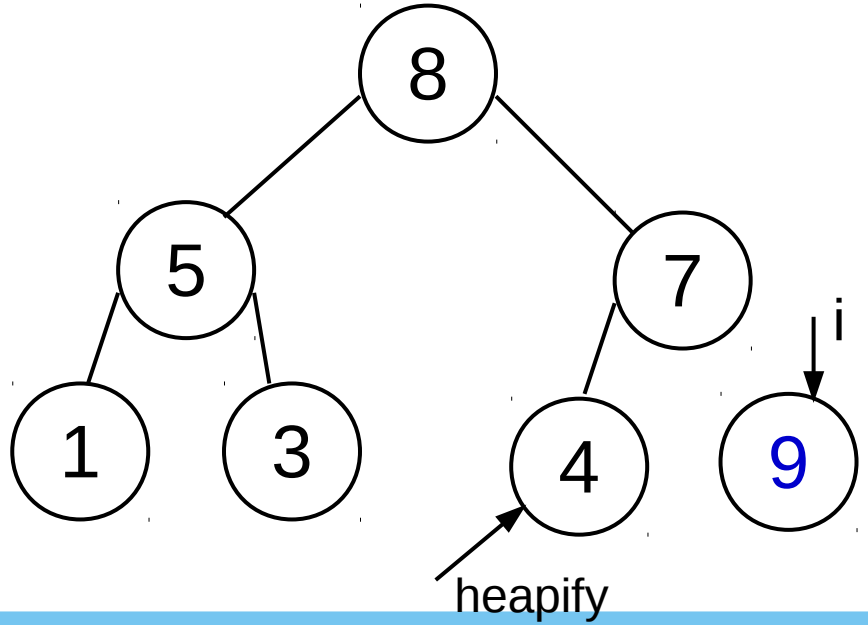
Heap Sort

8	5	4	1	3	7	9



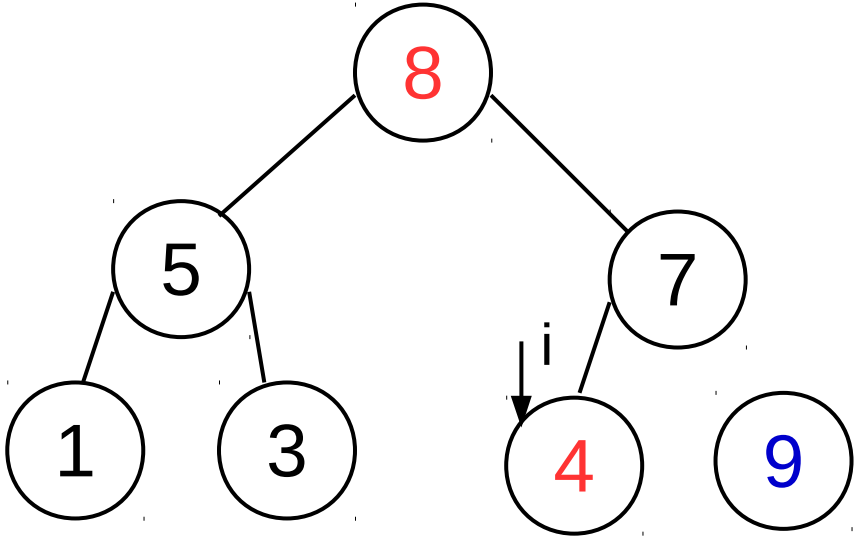
Heap Sort

8	5	7	1	3	4	9



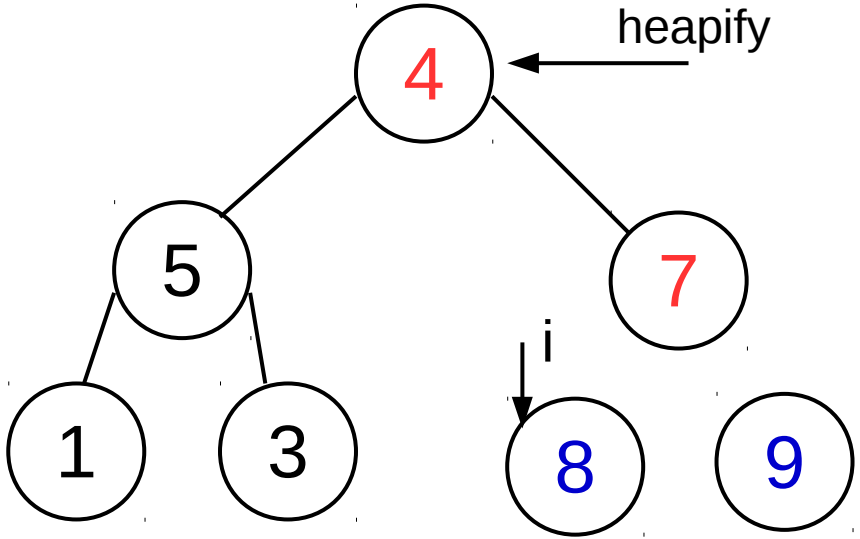
Heap Sort

8	5	7	1	3	4	9



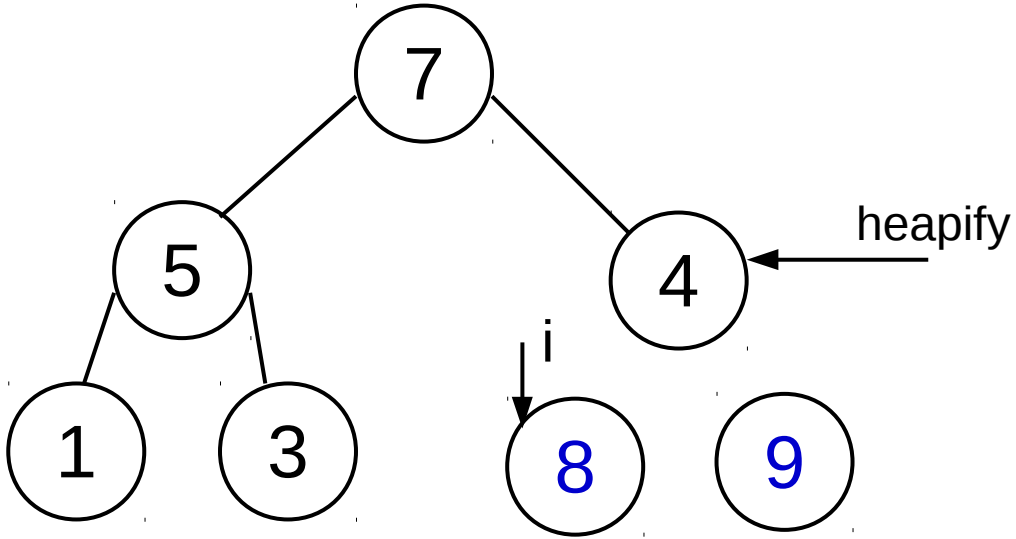
Heap Sort

4	5	7	1	3	8	9



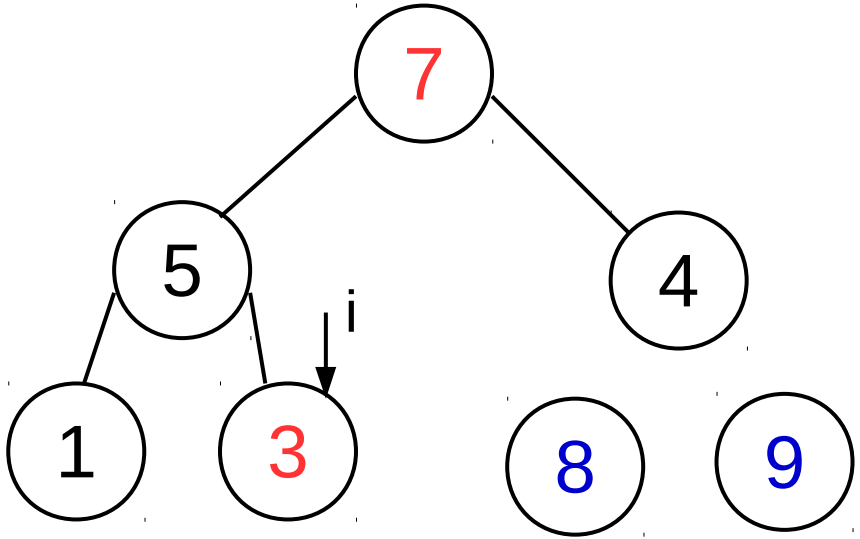
Heap Sort

7	5	4	1	3	8	9



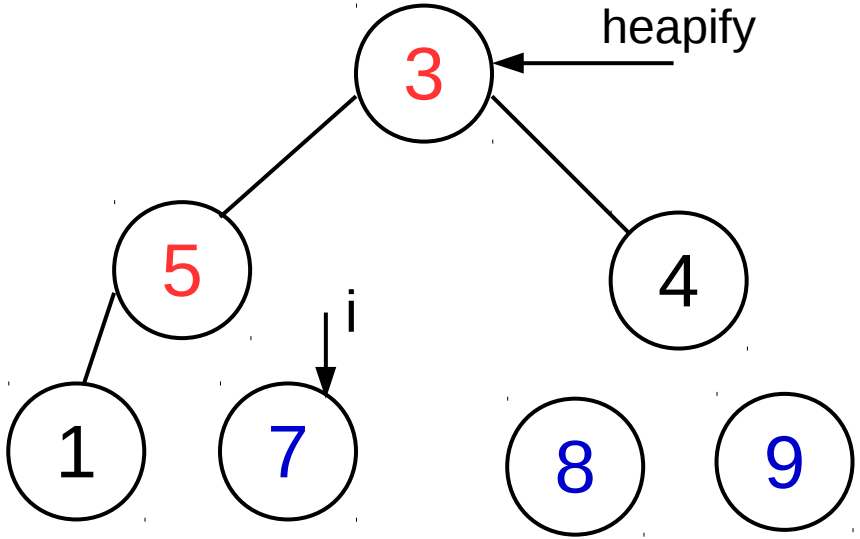
Heap Sort

7	5	4	1	3	8	9



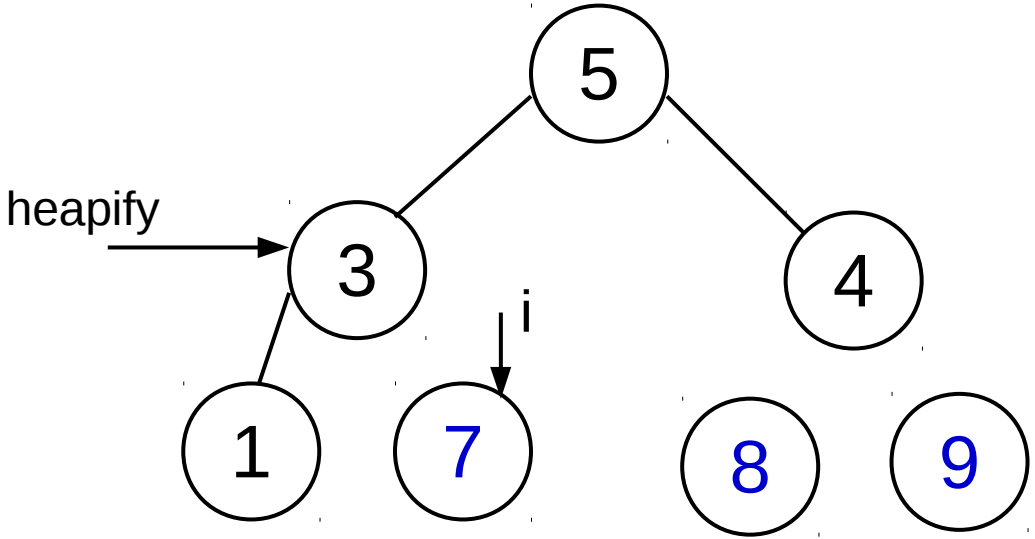
Heap Sort

3	5	4	1	7	8	9



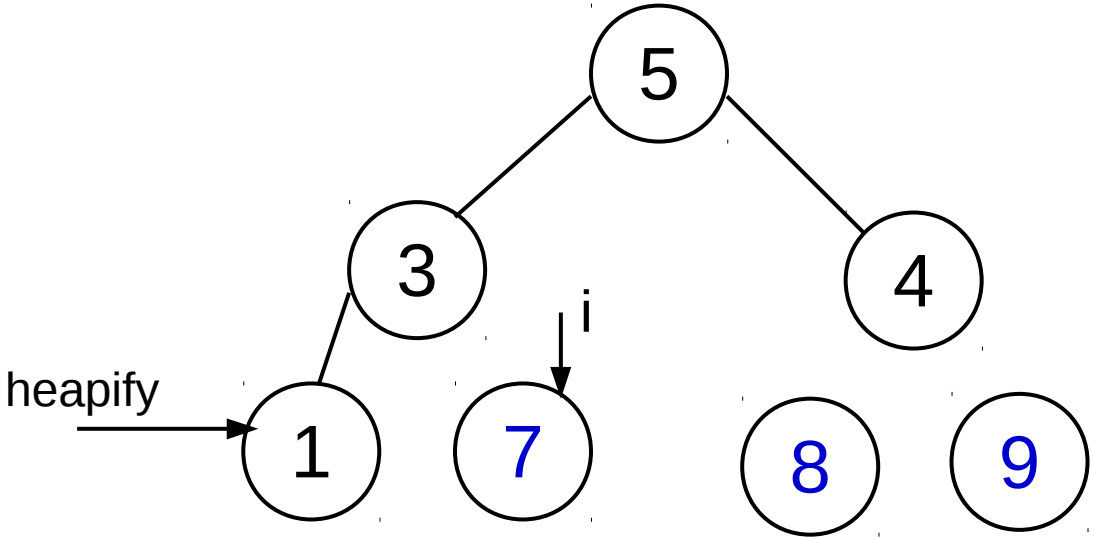
Heap Sort

5	3	4	1	7	8	9



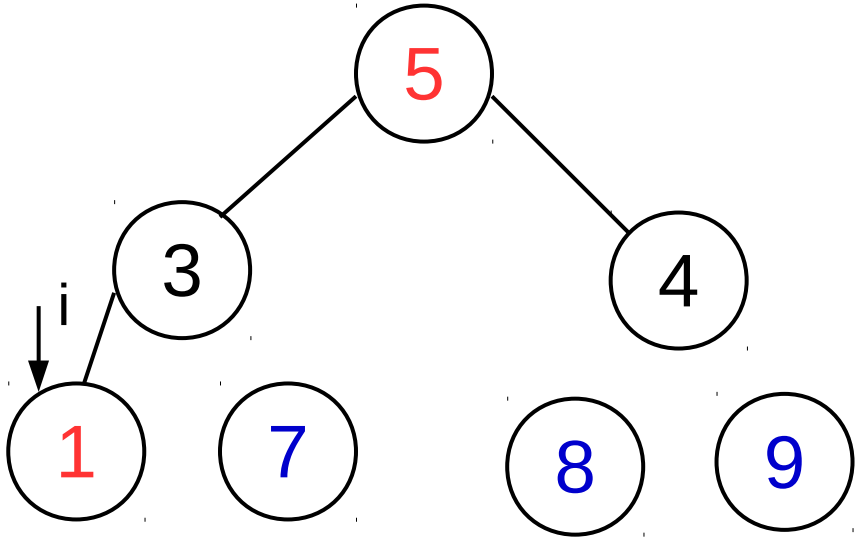
Heap Sort

5	3	4	1	7	8	9



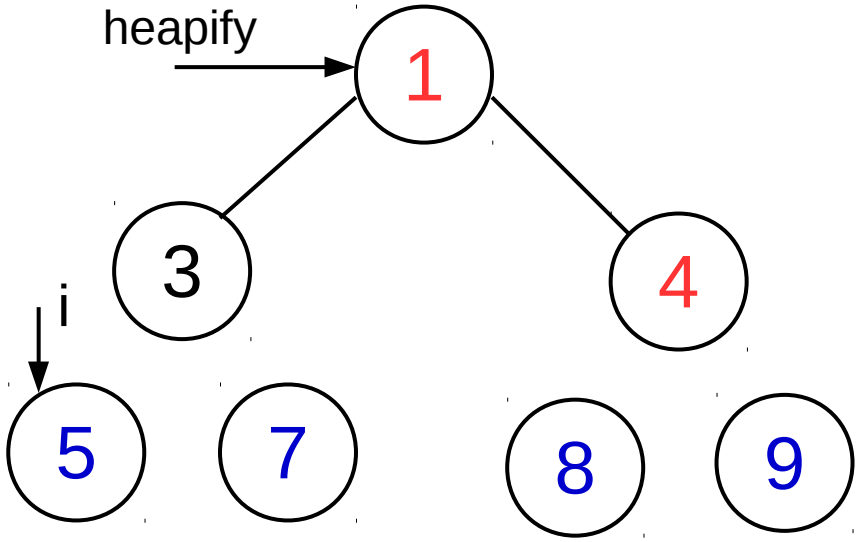
Heap Sort

5	3	4	1	7	8	9

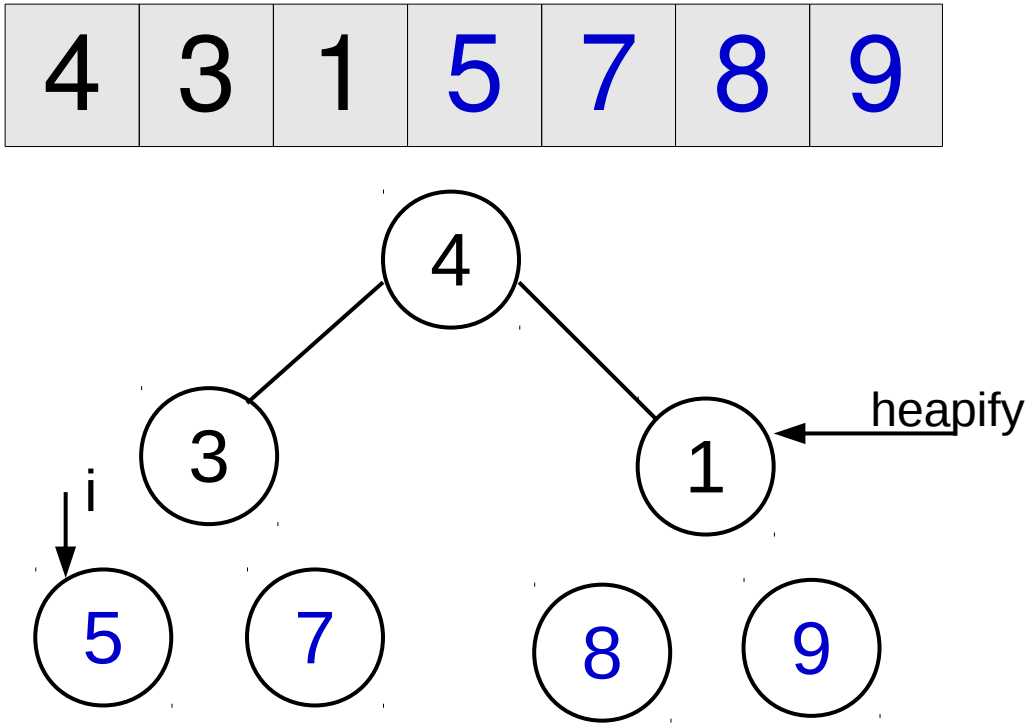


Heap Sort

1	3	4	5	7	8	9

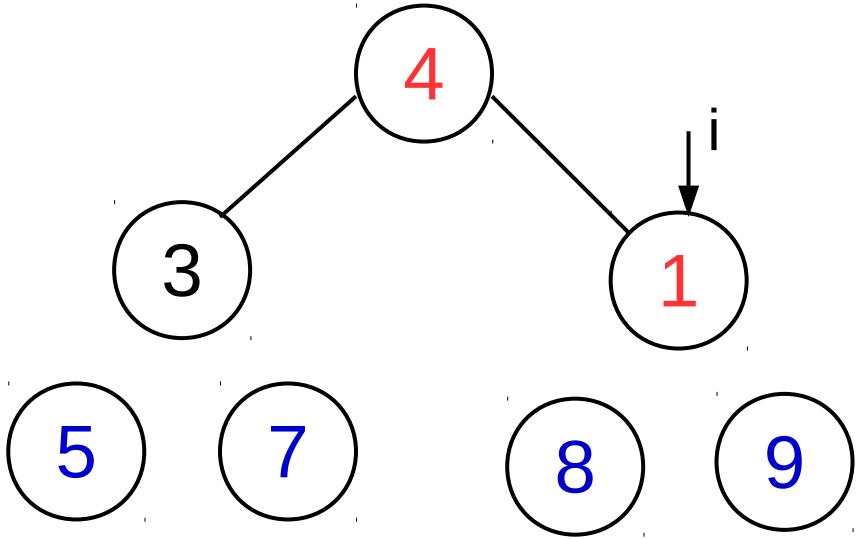


Heap Sort

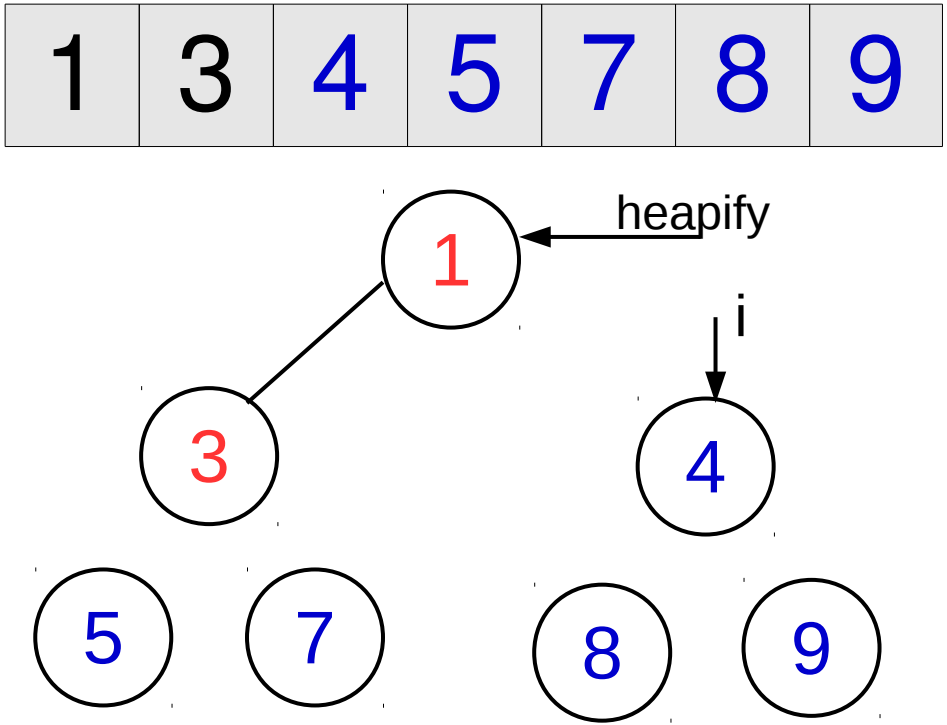


Heap Sort

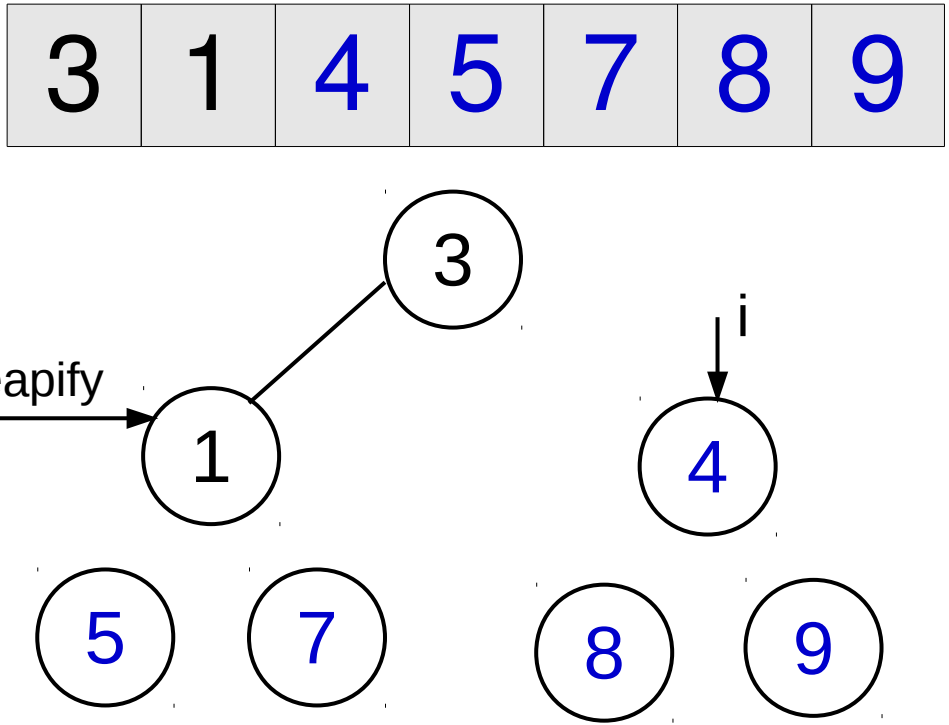
4	3	1	5	7	8	9



Heap Sort

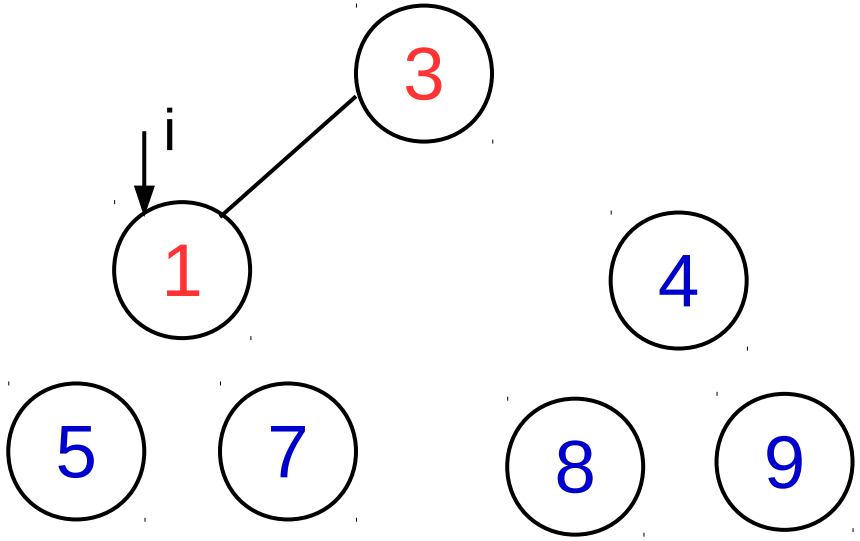


Heap Sort

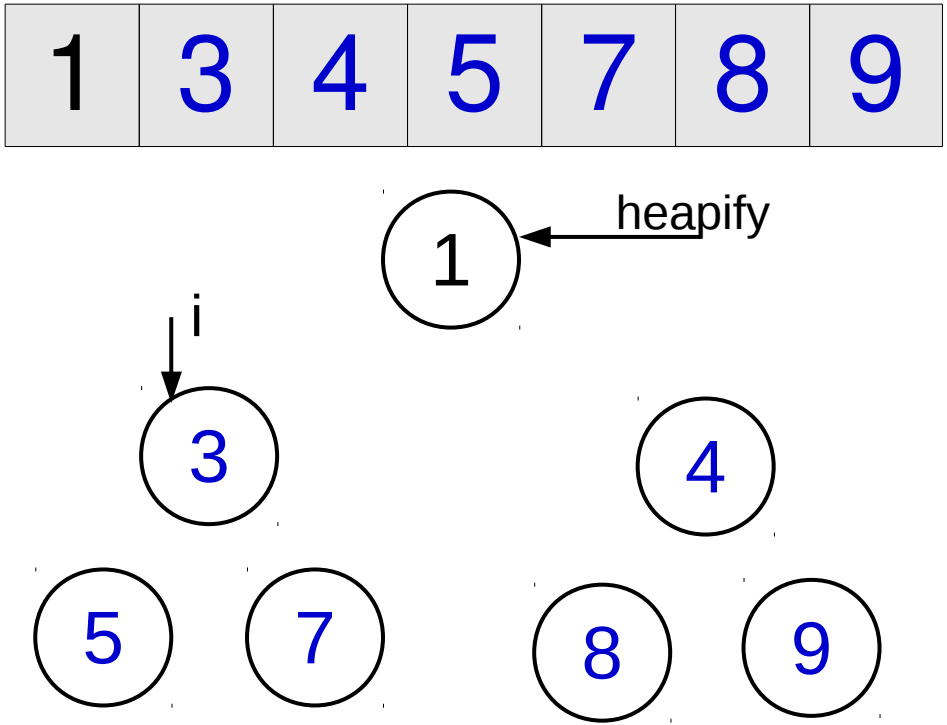


Heap Sort

3	1	4	5	7	8	9

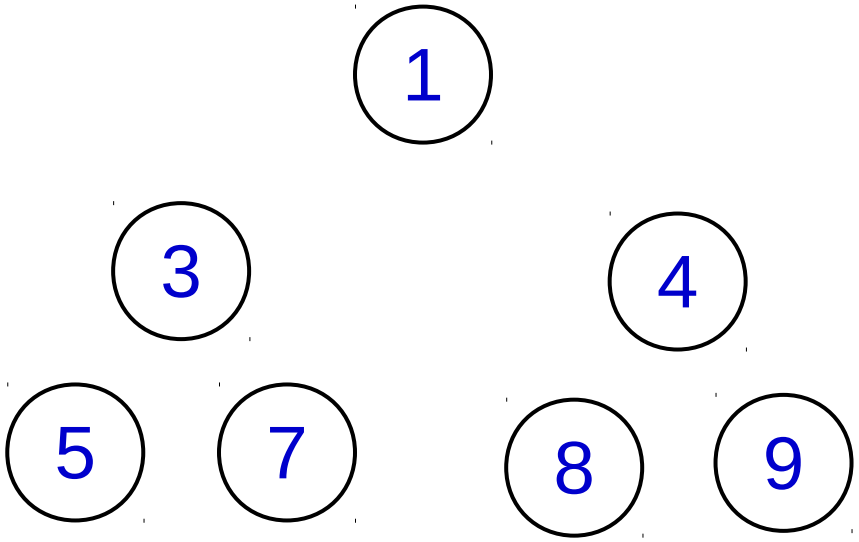


Heap Sort

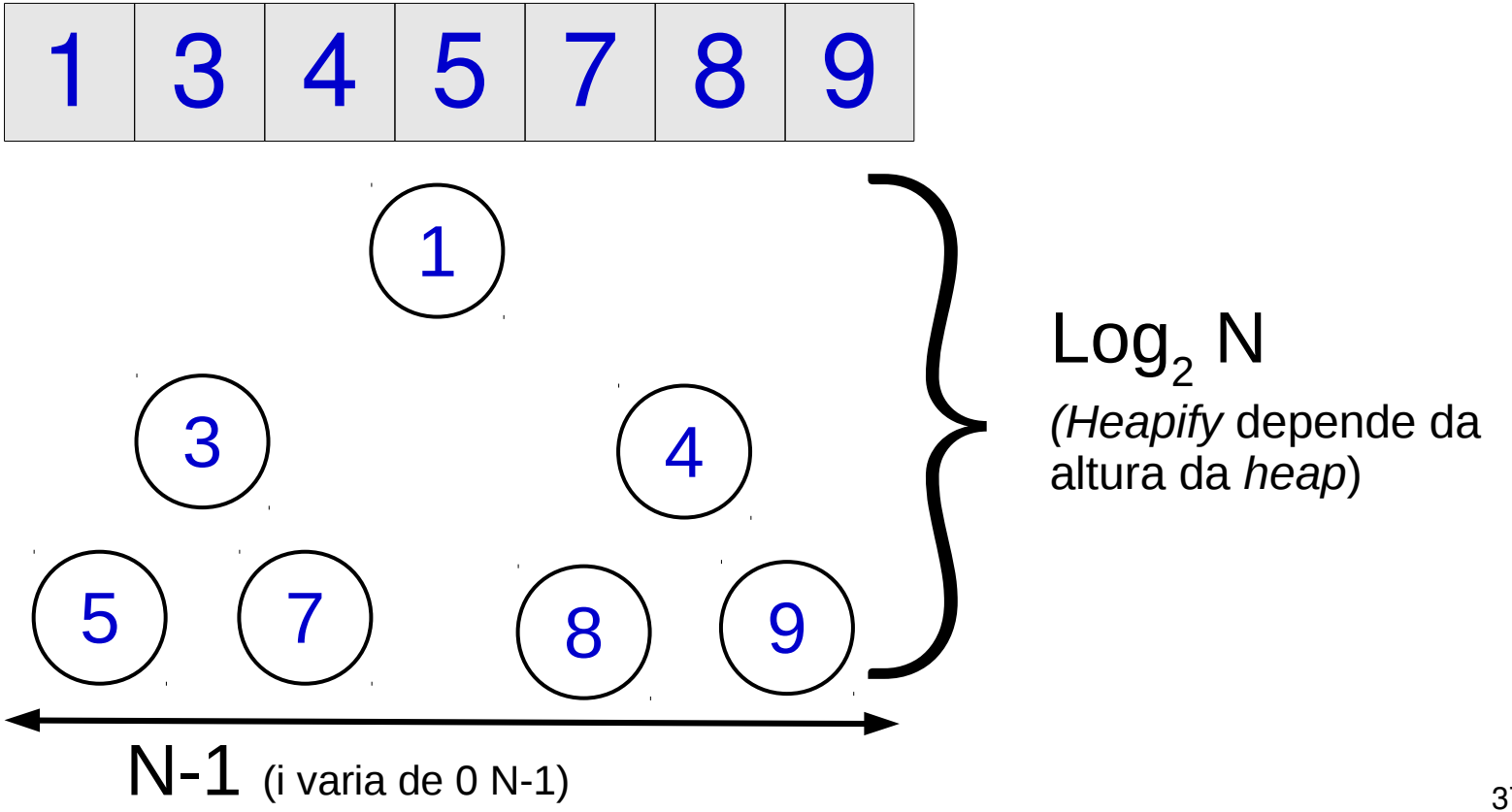


Heap Sort

1	3	4	5	7	8	9



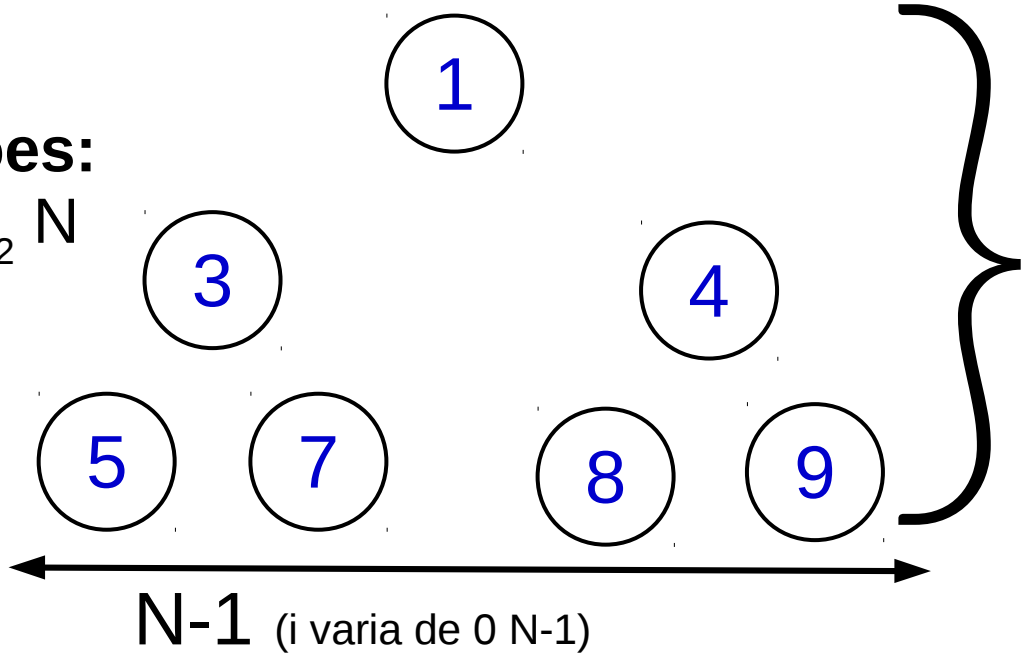
Heap Sort


1 3 4 5 7 8 9
1
3
4
5
7
8
9
Log2 N
(Heapify depende da altura da heap)
N-1 (i varia de 0 N-1)

Heap Sort

1	3	4	5	7	8	9

Total de Operações:
N-1 * $\log_2$ N



Log_2 N

(*Heapify* depende da altura da *heap*)

Bibliografia

Cormen, Thomas H. et al. Algoritmos.; [tradução Arlete Simille]. 3ª ed - Rio de Janeiro - Elsevier, 2011.

Carlos de Salles Soares Neto – Notas de Aula da Disciplina de Algoritmos I - UFMA