

Estrutura de Dados

Heap e fila de prioridade

GRUPO:

BRENNO DE FARO VIEIRA

HUMBERTO DA CONCEIÇÃO

JOSE FREIRE FALCÃO

NEWTON SOUZA SANTANA JUNIOR

PROFESSOR: ALBERTO COSTA NETO

Heap

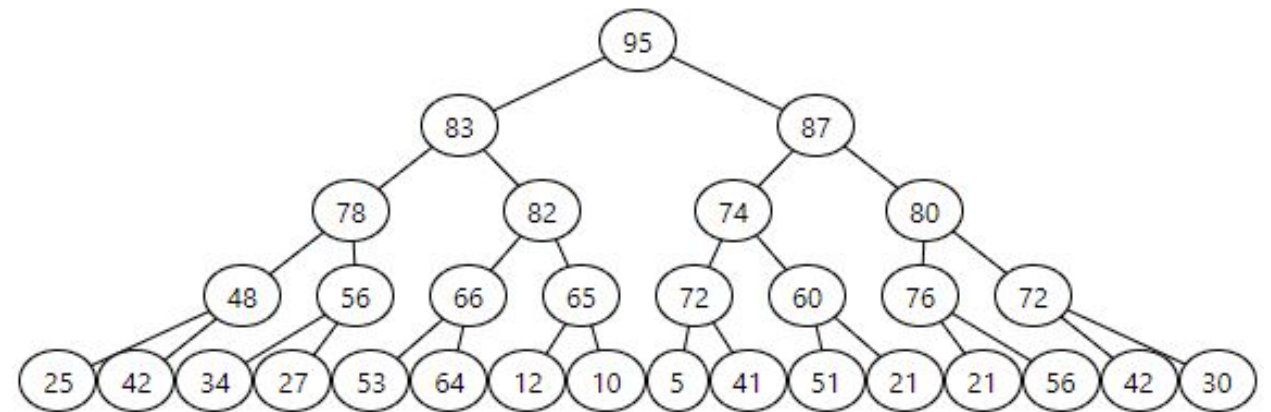
▣ O que é?

Heap é uma estrutura de dados baseada em árvore que respeita alguns critérios:

- ▶ Se o nó A possui um filho B, então $\text{chave}(A) \geq \text{chave}(B)$, caso seja um max-heap; caso seja um min-heap, então $\text{chave}(A) \leq \text{chave}(B)$.
- ▶ A remoção do heap sempre acontece pelo elemento posicionado na raiz.
- ▶ É uma árvore binária quase-completa, ou seja, todos os seus níveis estão completos, com exceção do último dependendo do caso.

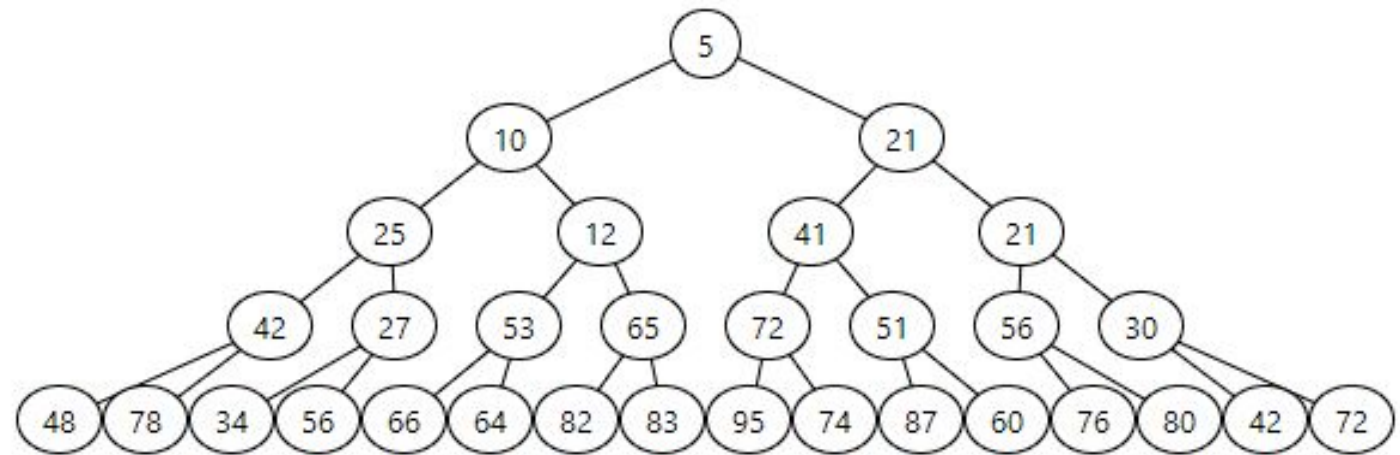
Heap

- ▶ Exemplo de max-heap:



Heap

- ▶ Exemplo de min-heap:



Heap

□ Implementação em array

Devido a sua característica de ser uma árvore quase-completa, o heap é melhor implementado em um arranjo.

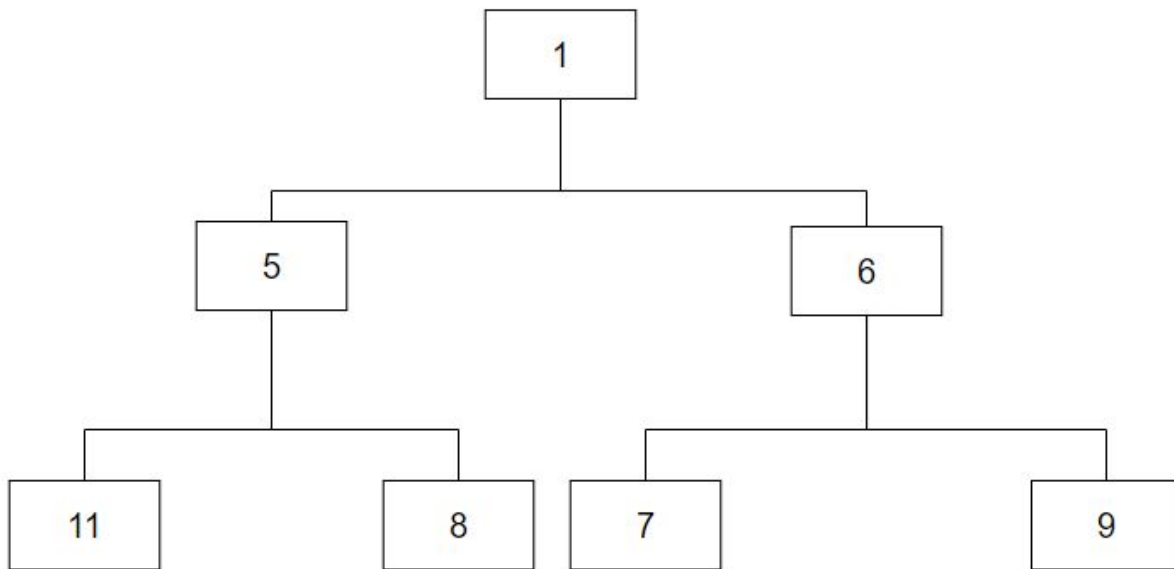
1	5	6	11	8	7	9
0	1	2	3	4	5	6

Heap

- Devido à sua implementação, é necessário saber alguns conceitos específicos do heap em um arranjo:
- ▶ O filho da esquerda de um Nó sempre estará na posição $2 \cdot \text{index} + 1$.
- ▶ O filho da direita de um Nó sempre estará na posição $2 \cdot \text{index} + 2$.

Na aplicação desse conceito, o nó não terá filhos se $2 \cdot \text{index} + 1$ ou $2 \cdot \text{index} + 2$ ultrapassar o tamanho da arranjo.

Heap



1	5	6	11	8	7	9
0	1	2	3	4	5	6

Heap

- ▶ Seguindo a mesma lógica, se um nó A tiver um nó pai B, B estará na posição $(\text{index}-1)/2$. Percebe-se que o único nó que não possui um nó pai é a raiz, por isso, ao calcular a posição do nó pai da raiz o resultado será -1, o que é uma posição inexistente no arranjo.

1	5	6	11	8	7	9
0	1	2	3	4	5	6

Heap

► Inserção

Para explicar a inserção, usaremos inicialmente exemplos.

Considere esta árvore:

1	3	2	6	4	5
0	1	2	3	4	5

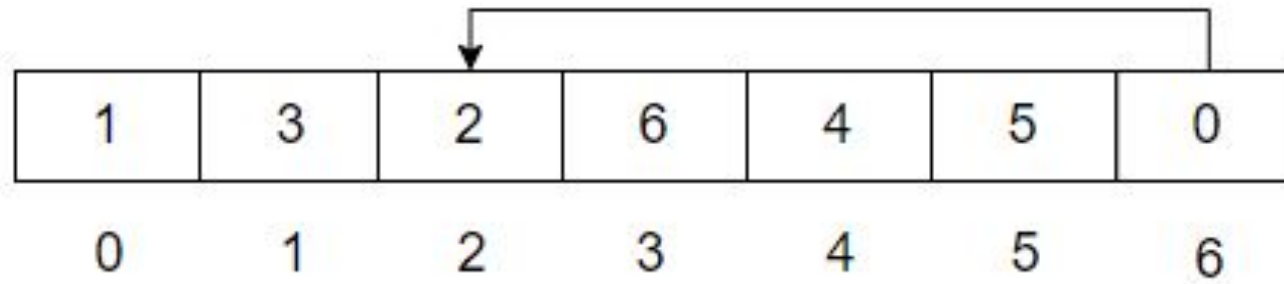
Heap

Percebe-se que a posição 6 está livre, logo o tamanho do heap será 6, pois existem apenas 6 elementos. Dessa forma, ao inserir o elemento 0, a inserção ocorrerá na posição correspondente ao tamanho da estrutura, ou seja, na posição 6. Após a inserção, o tamanho do heap atualizará para 7. O arranjo ficará assim:

1	3	2	6	4	5	0
0	1	2	3	4	5	6

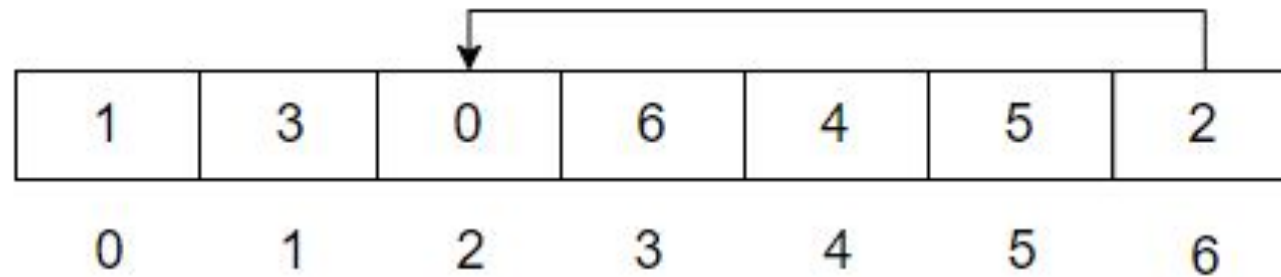
Heap

Contudo, a inserção do 0 nessa posição desrespeitará a regra do min-heap, em que o nó pai sempre será menor que o nó filho. Dessa maneira, é necessário corrigir esse problema. Para isso, faremos sucessivas comparações com o nó pai em busca da posição correta do 0.



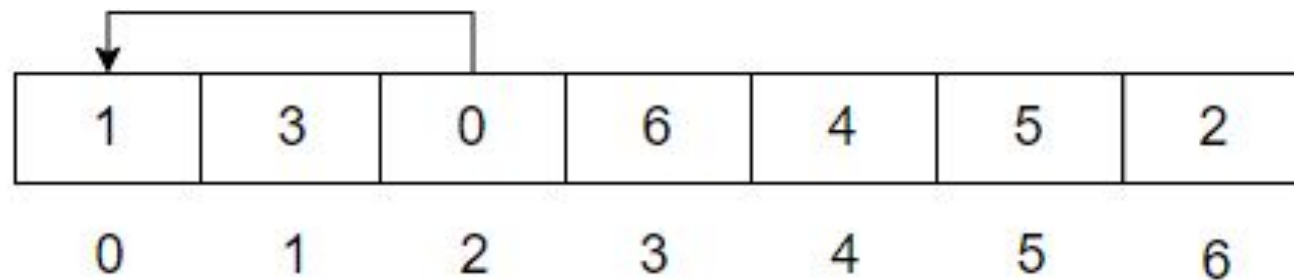
Heap

Como o pai é maior que o filho trocaremos pai e filho de posição, ficando assim:



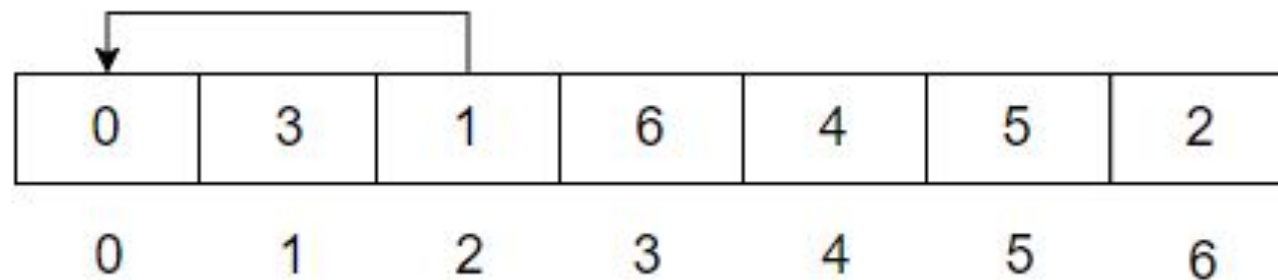
Heap

Após a troca, repetimos o processo até 0 ficar na posição correta:



Heap

Ocorrerá a troca novamente, pois o pai é maior que o filho.



Heap

As comparações continuariam, contudo, o 0 está na raiz do heap, ou seja, não existe mais nó pai para a comparação. Com isso, podemos verificar que as regras da implementação do heap estão sendo respeitadas.

0	3	1	6	4	5	2
0	1	2	3	4	5	6

Heap

□ Eficiência do método de inserção

No pior caso, a eficiência é $O(\log n)$, pois o objeto irá percorrer toda altura da árvore.

0	3	1	6	4	5	2
0	1	2	3	4	5	6

Heap

▣ Remoção

A remoção no heap é sempre feita no nó raiz.

Para manter as propriedades da heap, trocamos o ultimo nó pelo nó raiz e subtraímos 1 do tamanho do heap. Além disso, é necessário fazer sucessivas comparações com os nós filhos para colocar o nó trocado na posição correta.

Heap

► **Exemplo:**

Considere o seguinte heap:

0	3	1	6	4	5	2
0	1	2	3	4	5	6

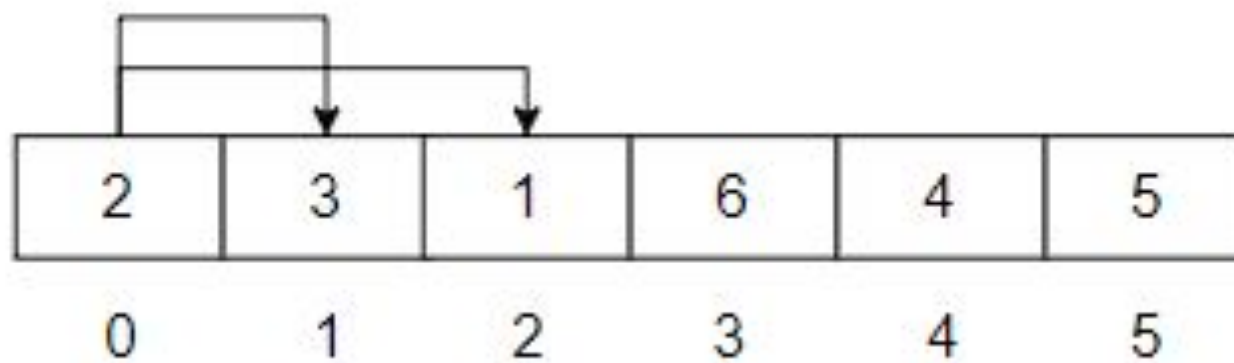
Heap

Para remover, inicialmente trocamos o último nó pelo nó raiz.

2	3	1	6	4	5	0
0	1	2	3	4	5	6

Heap

Subtraímos 1 do tamanho do heap e comparamos o nó raiz com os nós filhos.



Heap

Trocamos com o nó filho da direita, pois o filho possui menor valor, e continuamos comparando.



Heap

Não ocorrerá a troca e finalizará assim:

1	3	2	6	4	5
0	1	2	3	4	5

Heap

Ø Eficiência do método de remoção

No pior caso, a eficiência é $O(\log n)$, pois o objeto irá percorrer toda altura da árvore.

1	3	2	6	4	5
0	1	2	3	4	5

Heap

□ Principais usos:

► Filas de prioridade

Vai ser descrito posteriormente.

► Algoritmo heapsort

Basicamente, insere todos os elementos de um array informado dentro do heap e, após o fim das inserções, faz-se um laço para percorrer o heap de trás para frente; então, para cada elemento, há uma troca com o primeiro (de maior prioridade); diminui-se o tamanho e faz-se a correção do heap; no fim de tudo, retoma-se o tamanho original. Possui uma complexidade de tempo de $O(n \log n)$.

Heap

❌ Vantagens:

- ▶ Busca direta ao elemento de maior prioridade

O uso da heap é sempre ligado a facilitar a obtenção do elemento com maior prioridade. Devido ao suas regras de estruturação, o elemento de maior prioridade sempre ficara na raiz da árvore.

Heap

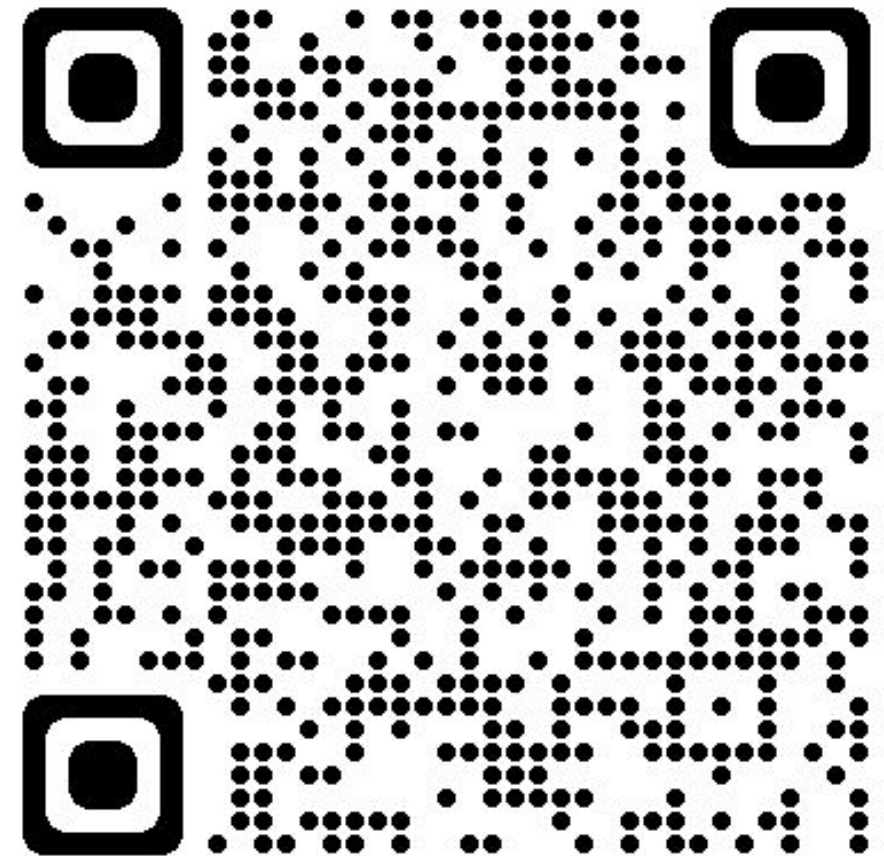
❑ Desvantagem

- ▶ Busca a algum elemento diferente do nó raiz

A estrutura de dados heap não é muito utilizada para busca de elementos devido a sua organização não favorecer essa operação.

Heap Sort

Visualização do processo de Heap Sort



► HeapSort

Heap

❑ Vantagens e Desvantagens

- ❑ O HeapSort é melhor que o QuickSort?

HeapSort

COMPLEXIDADE

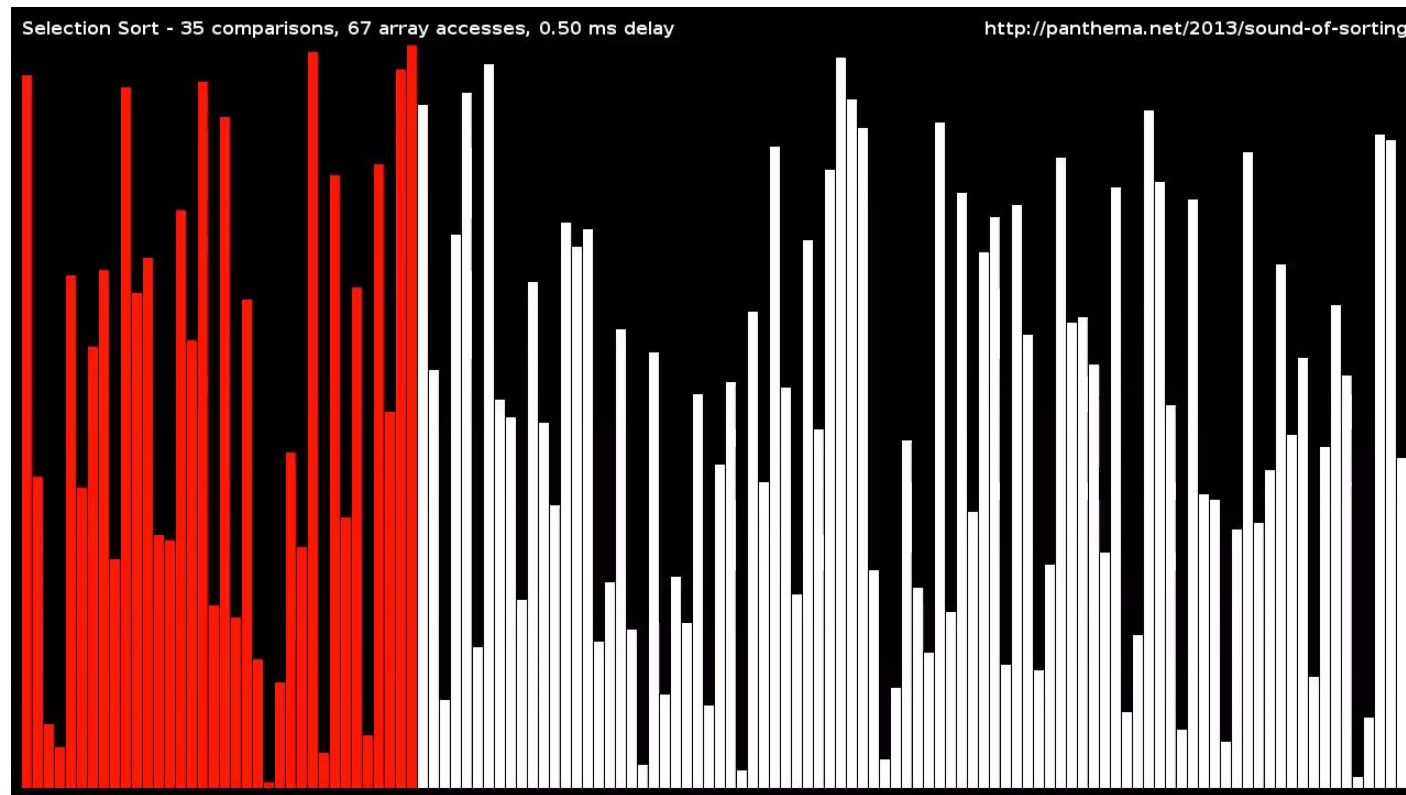
Complexidade média	$O(n \times \log n)$
Melhor caso	$O(n \times \log n)$
Pior caso	$O(n \times \log n)$
Complexidade espacial	$O(1)$

QuickSort

COMPLEXIDADE

Complexidade média	$O(n \times \log n)$
Melhor caso	$O(n \times \log n)$
Pior caso	$O(n^2)$
Complexidade espacial	$O(n)$

Vídeo do algoritmo de Ordenação



Fila de Prioridade

❑ Definição

- ❑ O que é uma fila de prioridade?
- ❑ Como funciona uma fila de prioridade?
- ❑ Diferença para uma fila comum
- ❑ Quais são as principais características de uma fila de prioridade?
 - ❑ Os elementos com maior prioridade são atendidos ou removidos primeiro da fila.



Fila de Prioridade

❑ Implementação

- ❑ Como implementar uma fila de prioridade?
- ❑ Quais são as diferentes formas de implementar uma fila de prioridade?
 - Usando lista encadeada
 - Usando um Heap
- ❑ Quais são as vantagens e desvantagens de cada forma de implementação?

Fila de Prioridade

❑ Implementação usando listas encadeadas

- Lista encadeada simples
- Lista duplamente encadeada

❑ A partir das listas encadeadas podemos ter 2 modos de implementar:

- Lista não Ordenada:

A inserção ocorre sempre no final de uma fila comum, essa estratégia é chamada de filas não ordenadas.

- Lista Ordenada:

Antes que um elemento seja inserido na fila, é necessário identificar a posição correta de acordo com a sua prioridade

Fila de Prioridade

- Implementação com uma **lista não ordenada**.

O Elemento é inserido sempre no **final da lista**, com complexidade **$O(1)$** e no momento da sua retirada a lista será percorrida totalmente para encontrar o elemento correto para remover, portanto a sua complexidade nessa etapa é **$O(n)$** .

- Implementação com uma **lista ordenada**.

O caso é semelhante ao que veremos a seguir, nesse caso, no processo de inserção o elemento já **é inserido na posição correta**, portanto é necessário percorrer a lista toda primeiro, obtendo uma complexidade **$O(n)$** .

Já no processo de remoção, basta remover o elemento de maior prioridade, que será sempre o primeiro, portanto a sua complexidade é **$O(1)$** .

Fila de Prioridade

- Implementação com uma **Heap**.

Na **inserção**, o novo elemento é adicionado no final do heap e depois reorganizado para manter a propriedade do heap. Isso envolve comparar o novo elemento com seu pai e trocá-los, se necessário, repetindo o processo até que o elemento esteja na posição correta. A complexidade de tempo da inserção é **$O(\log n)$** , onde n é o número de elementos no heap.

Na **remoção**, o elemento de maior ou menor prioridade (dependendo do tipo de heap) é removido da raiz do heap. Em seguida, o último elemento é movido para a raiz e reorganizado para manter a propriedade do heap, comparando-o com seus filhos e trocando-o, se necessário, até que esteja na posição correta. A complexidade de tempo da remoção também é **$O(\log n)$** devido ao número máximo de comparações e trocas, onde n é o número de elementos no heap.

Big-O Complexity Chart

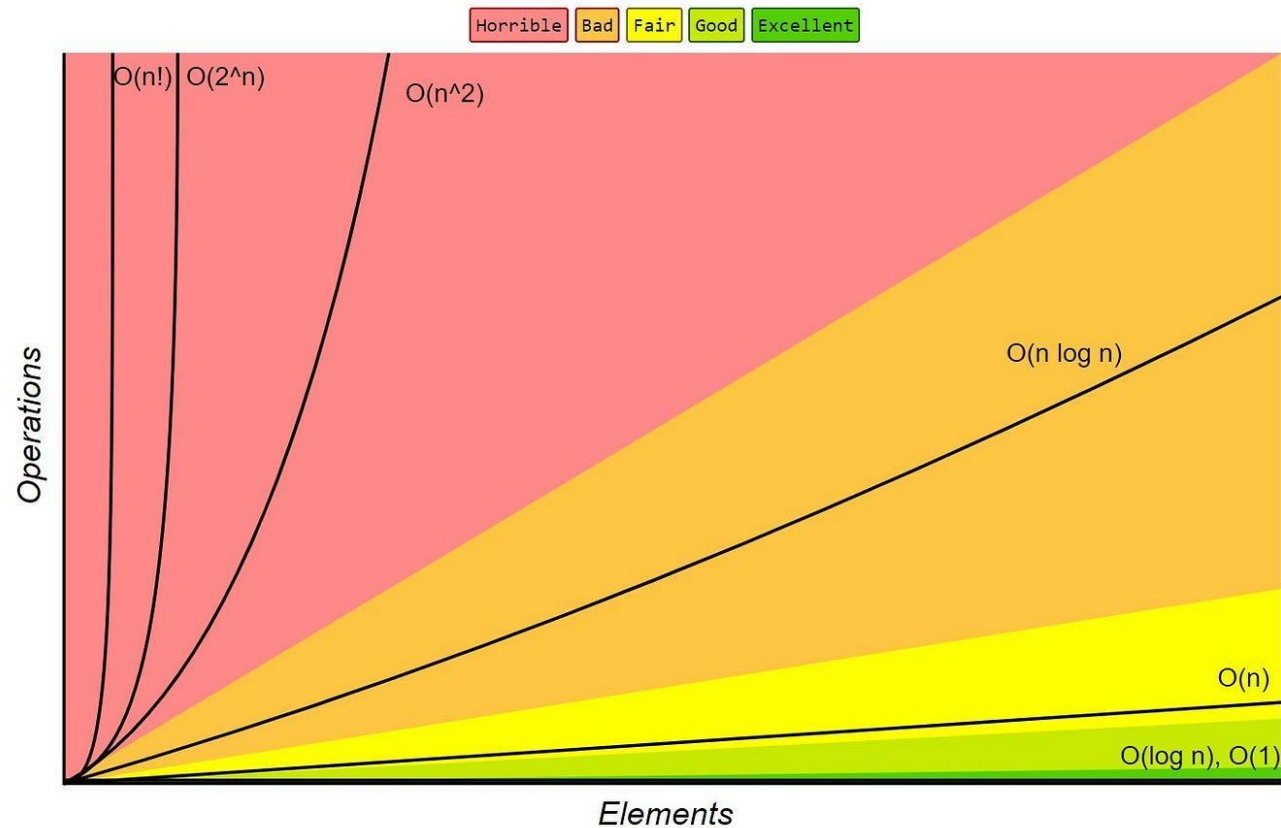


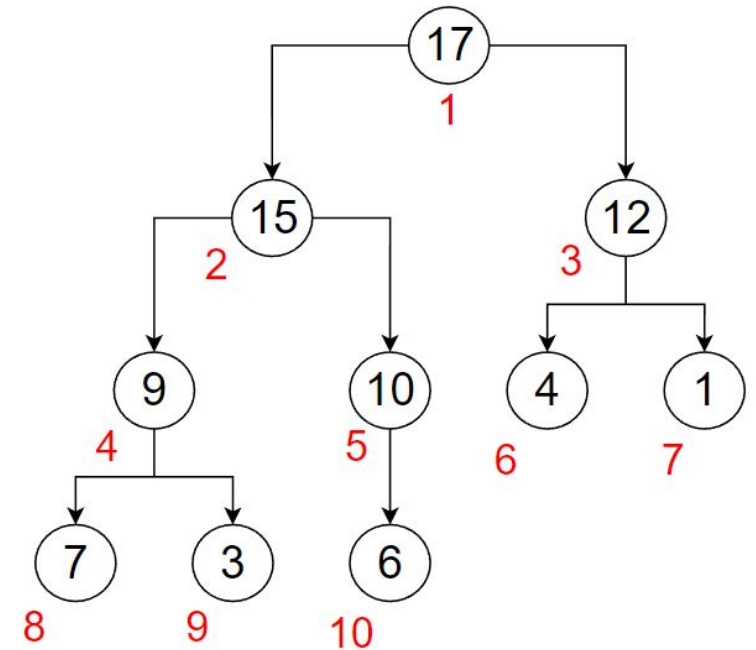
Grafico de
complexidade

Fila de Prioridade

❑ Implementação usando um Heap (Inserção)

Utilizando como exemplo o seguinte Heap:

17	15	12	9	10	4	1	7	3	6
1	2	3	4	5	6	7	8	9	10

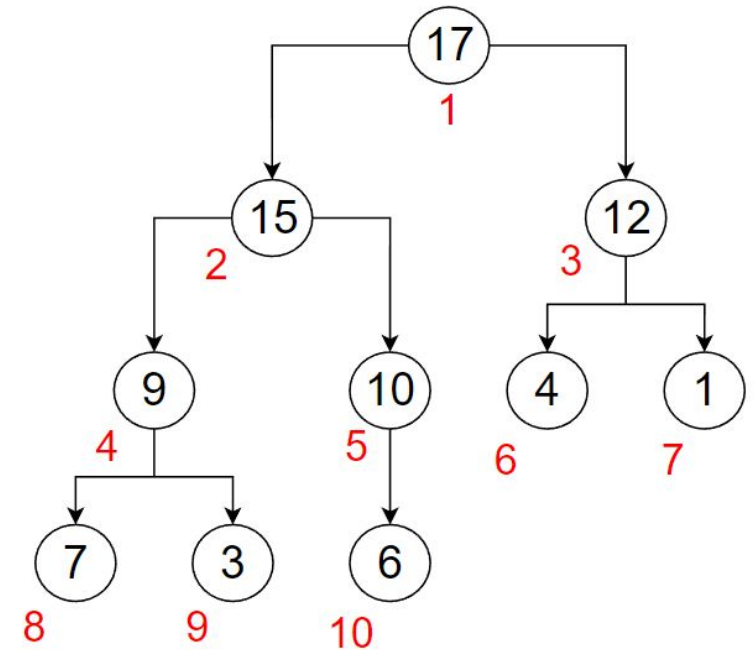


Fila de Prioridade

❑ Implementação usando um Heap (Inserção)

A operação é igual a explicada anteriormente.

17	15	12	9	10	4	1	7	3	6
1	2	3	4	5	6	7	8	9	10



Fila de Prioridade

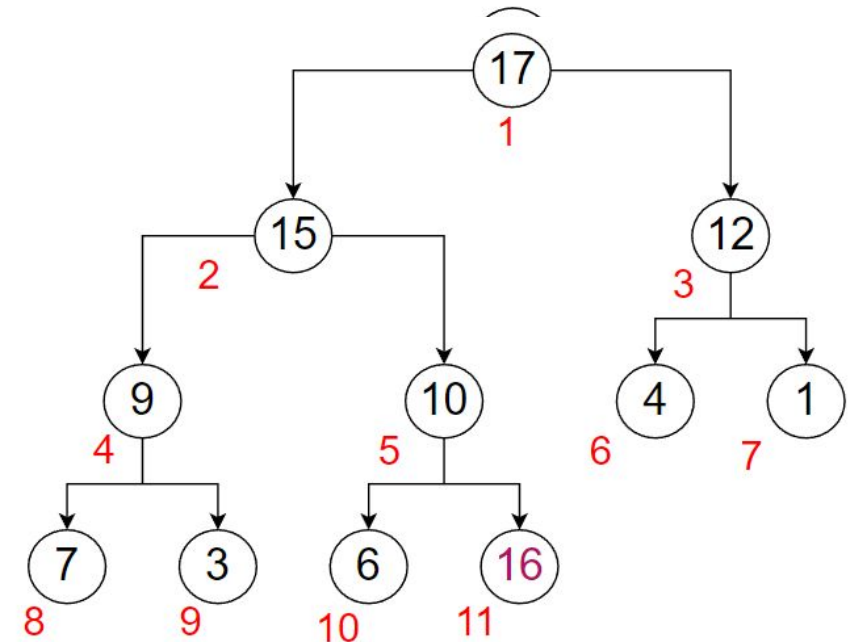
❑ Implementação usando um Heap (Inserção)

Realizando a **inserção** do elemento 16

16

Primeiramente inserimos o elemento no **final da heap**

17	15	12	9	10	4	1	7	3	6	16
1	2	3	4	5	6	7	8	9	10	11



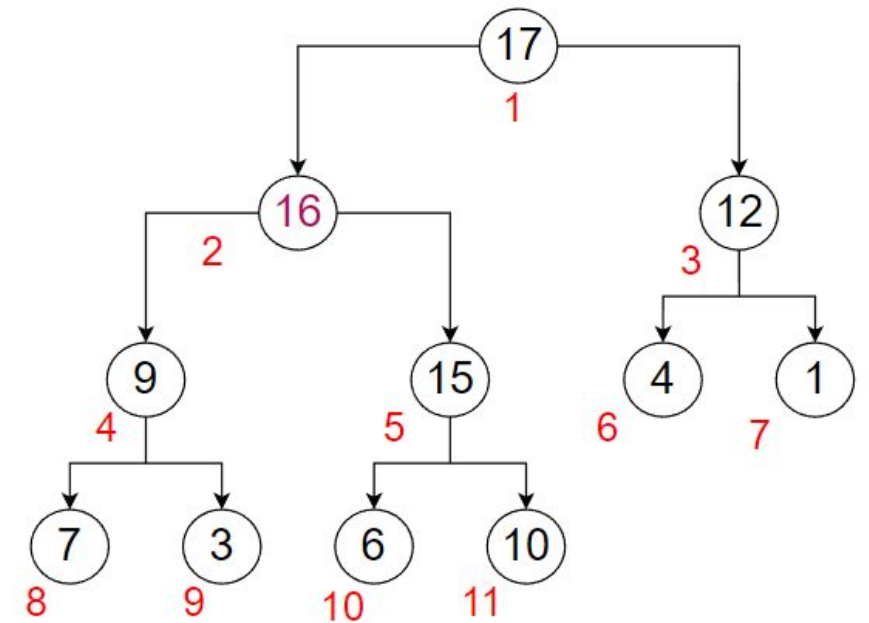
Fila de Prioridade

❑ Implementação usando um Heap (Inserção)

Realizando a **inserção** do elemento 16

Ajustando o valor 16 para seguir as **propriedades da heap**

17	16	12	9	15	4	1	7	3	6	10
1	2	3	4	5	6	7	8	9	10	11



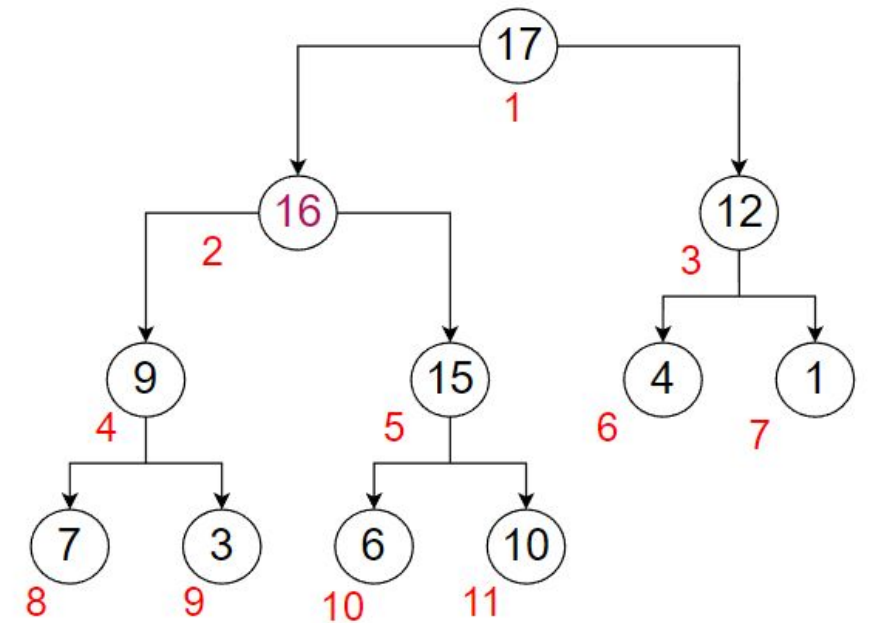
Fila de Prioridade

❑ Implementação usando um Heap (Inserção)

Realizando a **inserção** do elemento 16

Agora a Heap está corretamente ajustada.

17	16	12	9	15	4	1	7	3	6	10
1	2	3	4	5	6	7	8	9	10	11



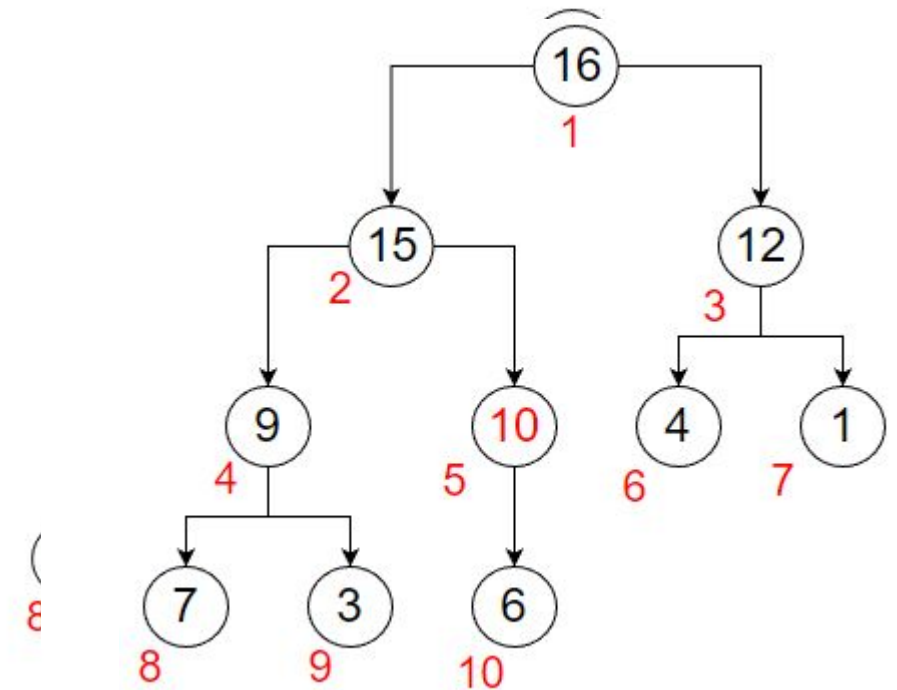
Fila de Prioridade

❑ Implementação usando um Heap (Remoção)

Realizando a **remoção** de 1 elemento

~~Remove o elemento com a melhor prioridade~~
Remove o elemento com a melhor prioridade com o último elemento da Heap

16	15	12	9	10	4	1	7	3	6
1	2	3	4	5	6	7	8	9	10



Fila de Prioridade

❑ Operações com lista encadeada

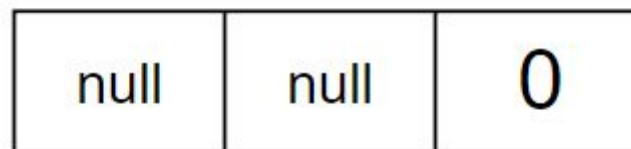
- ❑ Inserção de um elemento na fila de prioridade

Fila de Prioridade

❑ Operação de inserção em uma lista encadeada

- Inicialização da fila.

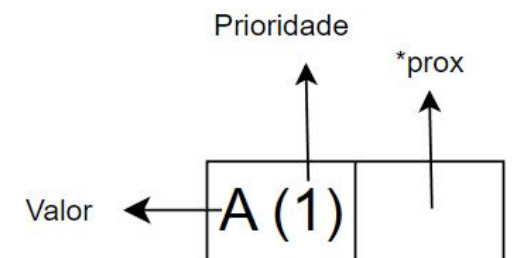
Fila



*inicio

*fim

tamanho



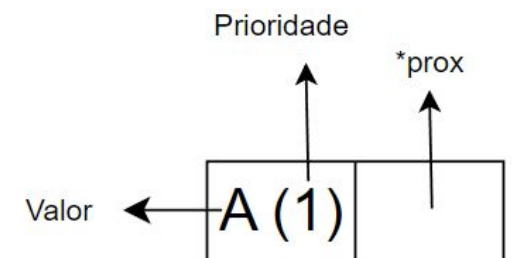
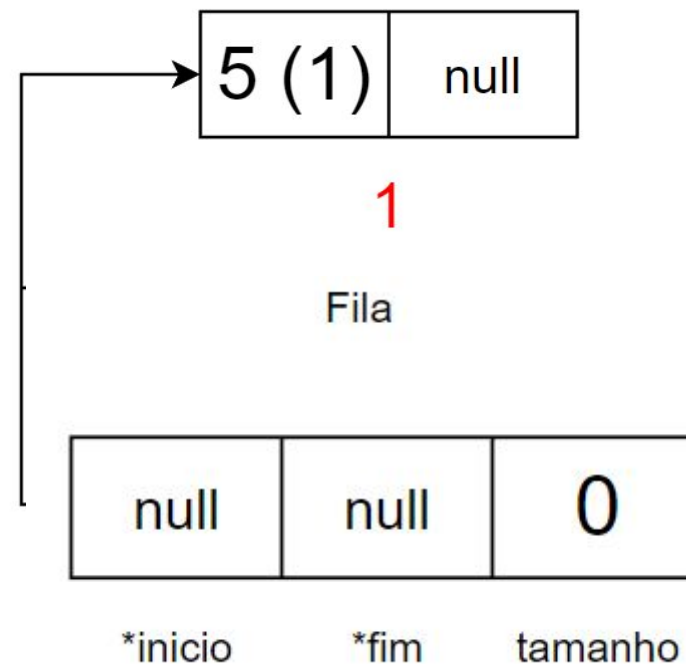
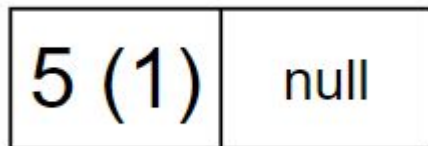
Fila de Prioridade

❑ Operação de inserção em uma lista encadeada

- Inserção do primeiro elemento na fila

Como é o **primeiro elemento da fila**, basta alterar os ponteiros de início e fim para este elemento e adicionar 1 ao tamanho.

Valor a ser inserido



Fila de Prioridade

❑ Operação de inserção em uma lista encadeada

- Inserção do segundo elemento na fila

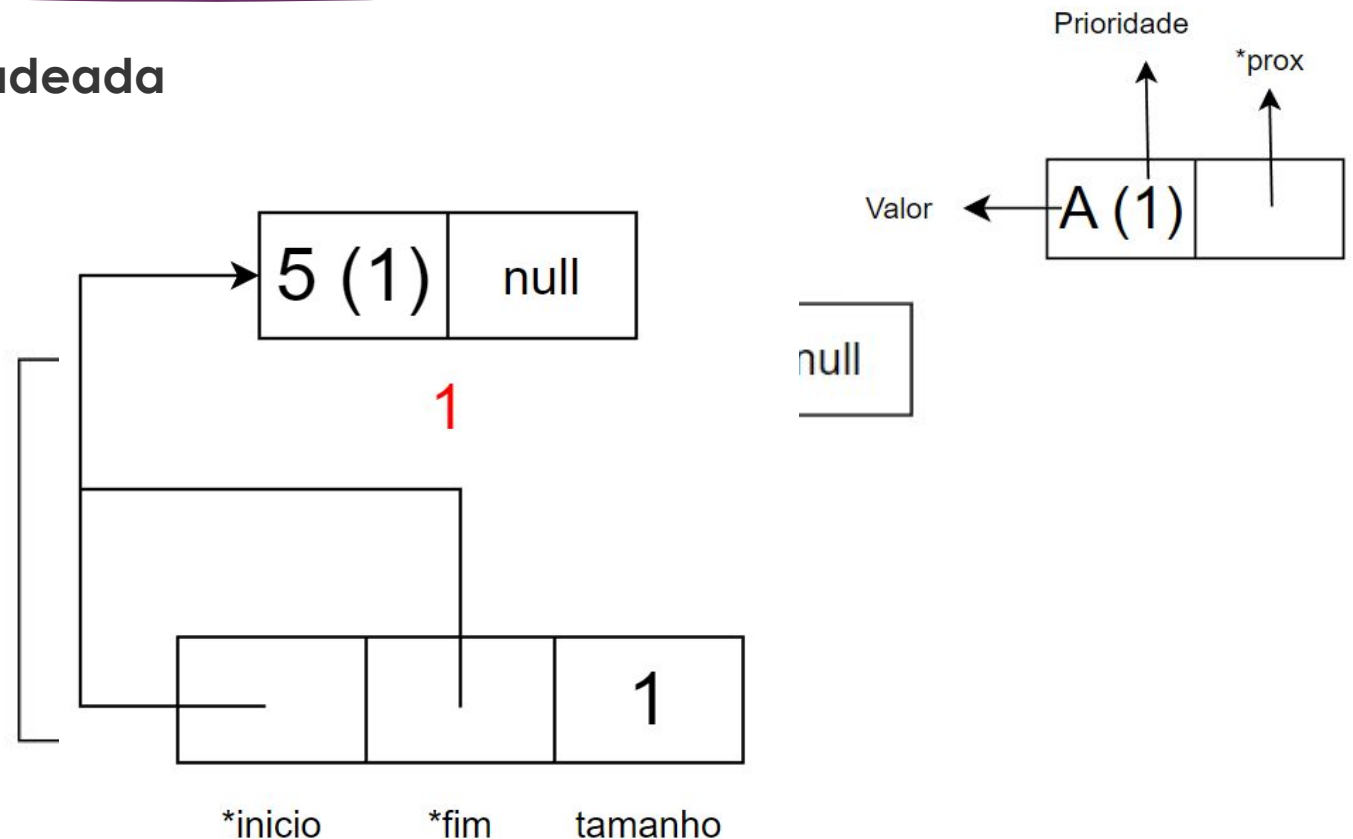
Como **já existe um elemento na fila** é preciso verificar se a sua prioridade é menor ou igual ao elemento que eu quero inserir. Como a prioridade 2 é maior do que 1 (1 é a maior prioridade) então ele deve ser inserido depois do valor 5.

Para isso é alterado o ponteiro **prox** do valor 5 para esse novo elemento;

Alterar o ponteiro **fim** para esse elemento;

Adicionar +1 ao **tamanho** da fila;

Valor a ser inserido



Fila de Prioridade

❑ Operação de inserção em uma lista encadeada

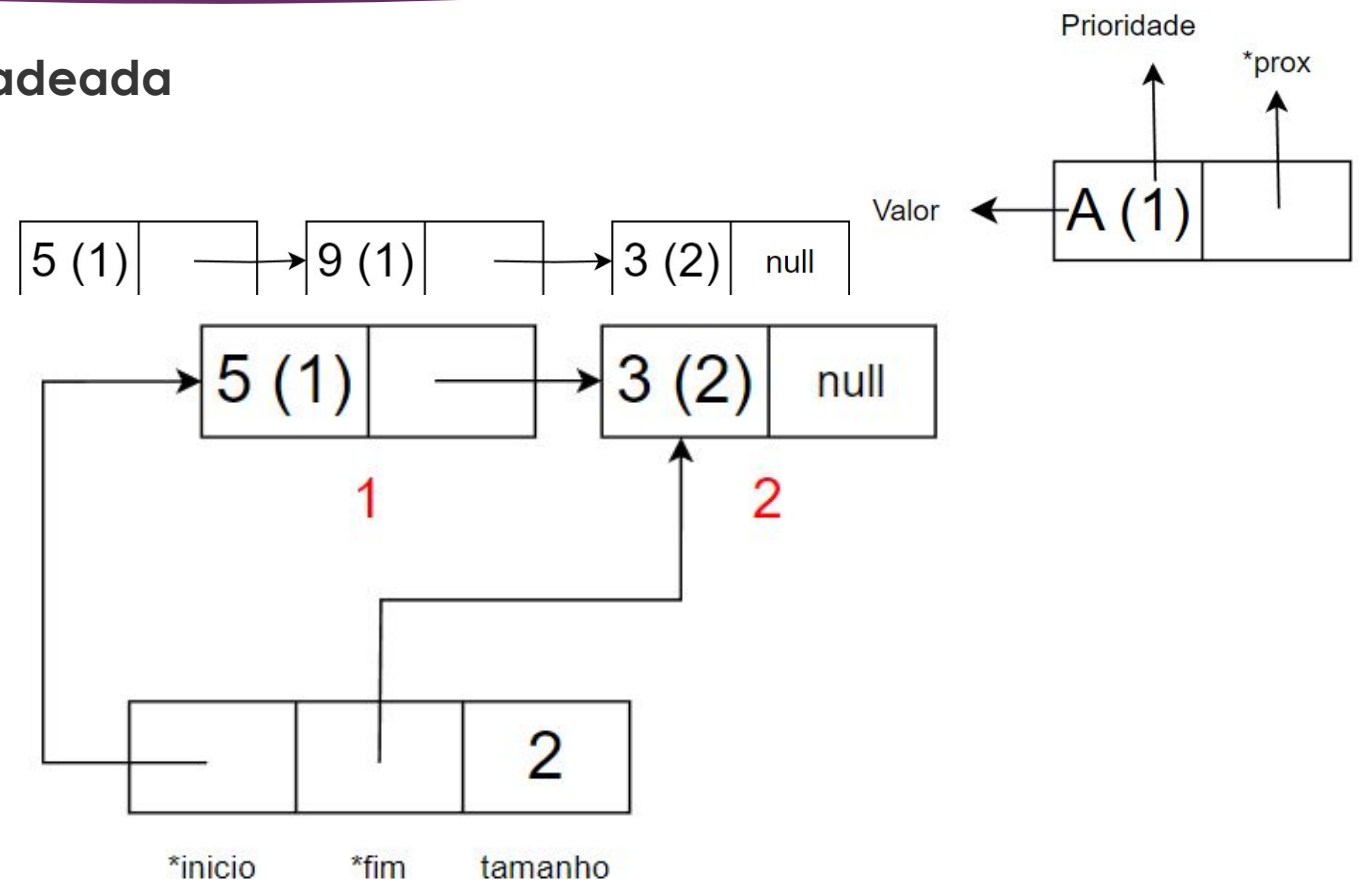
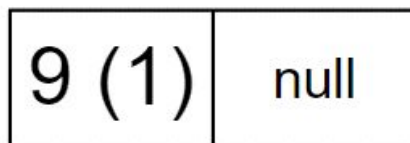
- Inserção de outro elemento na fila

Agora a fila já **possui mais de um elemento** então é necessário percorrer a mesma para achar a posição correta. Como o valor 9 possui prioridade 1, então ele será inserido logo após o 5.

Como já sabemos nas operações das listas encadeadas para inserir um elemento é necessário alterar o ponteiro **prox** do novo elemento para o valor 3, nesse caso;

E depois alterar o ponteiro do **prox** de 5 para esse novo elemento

Valor a ser inserido



Fila de Prioridade

❑ Operação de inserção em uma lista encadeada

Para as demais inserções o processo segue o mesmo.

Fila de Prioridade

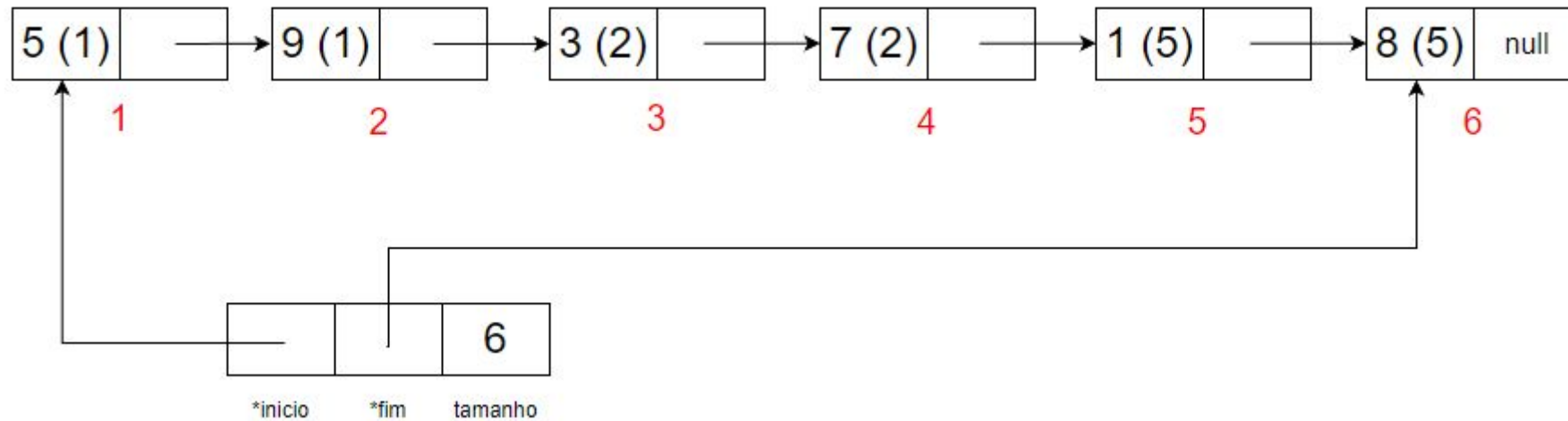
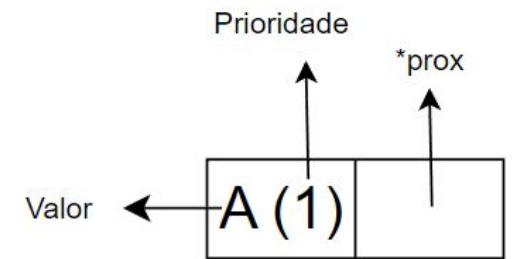
❑ Operações

- ❑ Remoção de um elemento na fila de prioridade

Fila de Prioridade

❑ Operação de remoção em uma lista encadeada

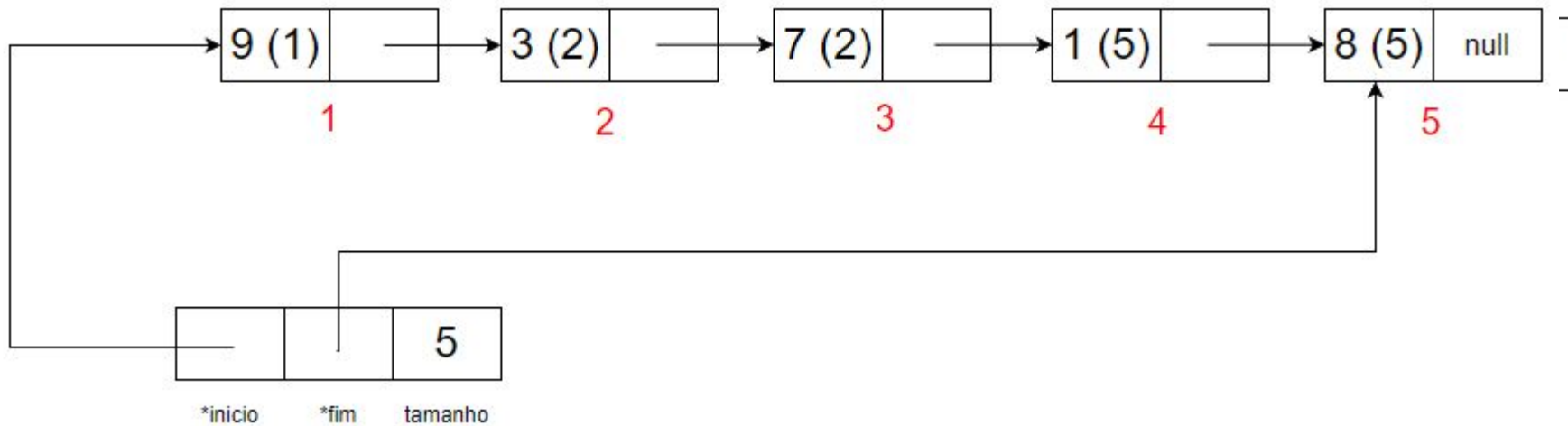
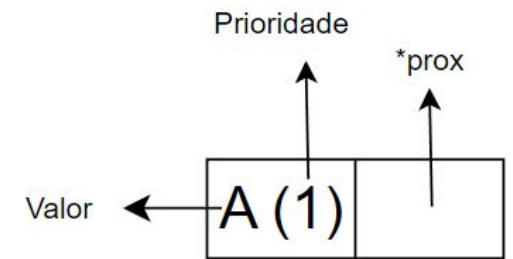
- Supondo que temos a seguinte fila de prioridade:
- Como que fazemos para realizar a **remoção**?



Fila de Prioridade

❑ Operação de remoção em uma lista encadeada

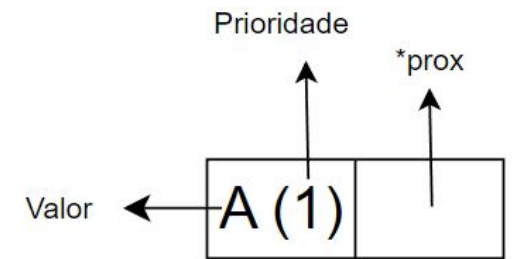
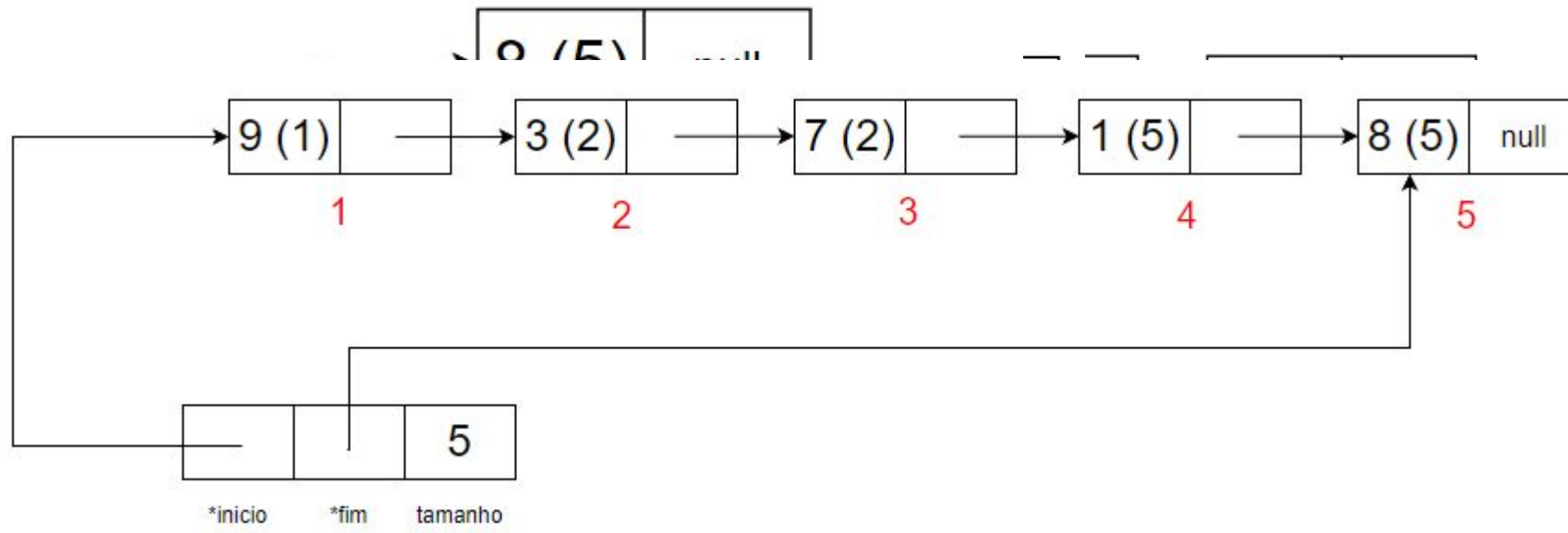
- É bastante simples.
- Basta remover o **primeiro elemento com a menor prioridade**
- Ajustar o **ponteiro** da fila
- Diminuir o **tamanho** da fila



Fila de Prioridade

❑ Operação de remoção em uma lista encadeada

- O processo se repete até a fila estar vazia



Fila de Prioridade

❑ Aplicações

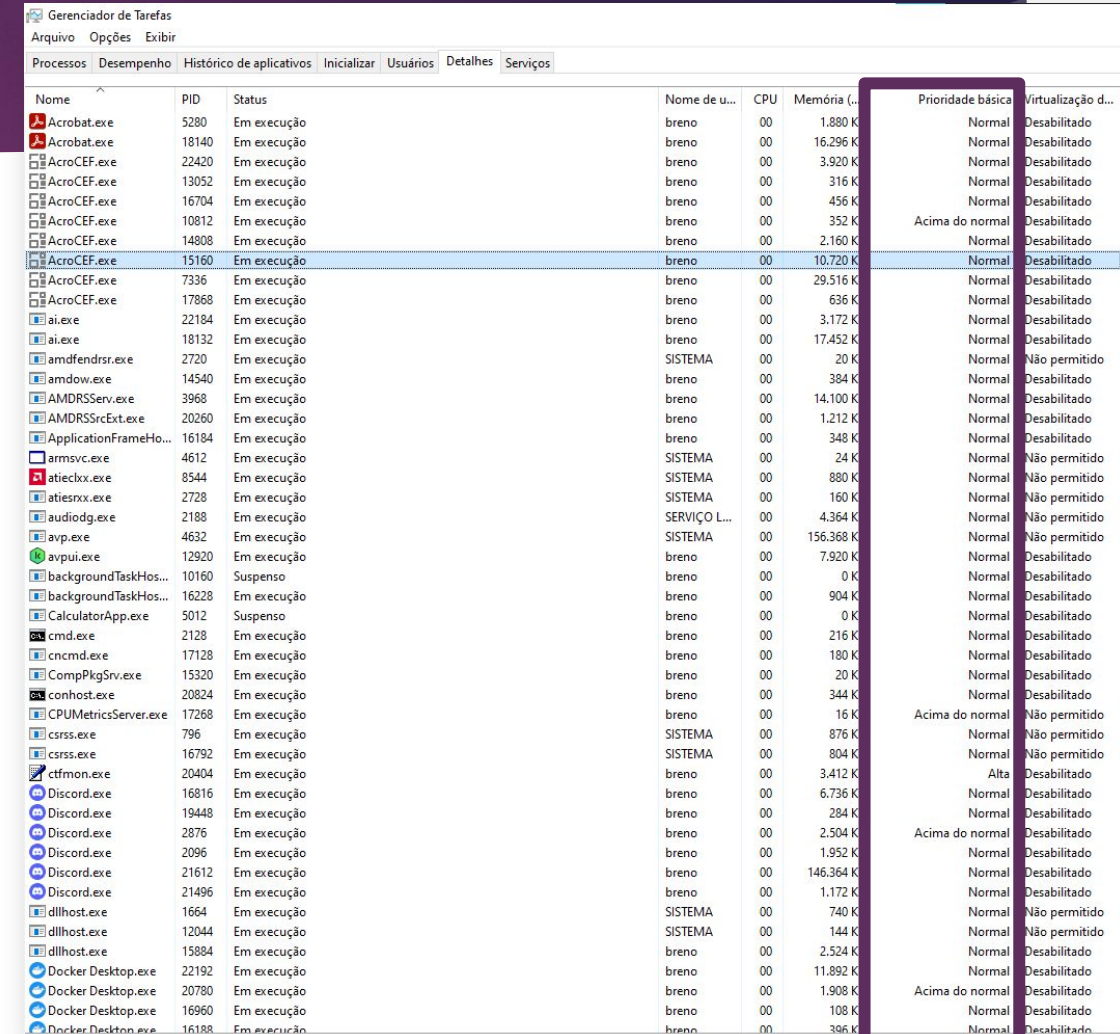
❑ Quais são alguns exemplos comuns de filas de prioridade?

- Prioridade de processos em sistemas operacionais:

Filas de prioridade são usadas para gerenciar os processos que executam em um sistema operacional, atribuindo-lhes uma prioridade baseada em critérios como tempo de chegada, tempo de execução ou importância.

- Algoritmo de Dijkstra

O algoritmo de Dijkstra é uma técnica fundamental para encontrar o caminho mais curto em um grafo ponderado. Ele é amplamente utilizado em sistemas operacionais e em muitas outras aplicações, como roteamento de rede e sistemas de GPS.



Gerenciador de Tarefas

Arquivo Opções Exibir

Processos Desempenho Histórico de aplicativos Inicializar Usuários Detalhes Serviços

Nome	PID	Status	Nome de u...	CPU	Memória (...)	Prioridade básica	Virtualização d...
Acrobat.exe	5280	Em execução	breno	00	1.880 K	Normal	Desabilitado
Acrobat.exe	18140	Em execução	breno	00	16.296 K	Normal	Desabilitado
AcroCEF.exe	22420	Em execução	breno	00	3.920 K	Normal	Desabilitado
AcroCEF.exe	13052	Em execução	breno	00	316 K	Normal	Desabilitado
AcroCEF.exe	16704	Em execução	breno	00	456 K	Normal	Desabilitado
AcroCEF.exe	10812	Em execução	breno	00	352 K	Acima do normal	Desabilitado
AcroCEF.exe	14808	Em execução	breno	00	2.160 K	Normal	Desabilitado
AcroCEF.exe	15160	Em execução	breno	00	10.720 K	Normal	Desabilitado
AcroCEF.exe	7336	Em execução	breno	00	29.516 K	Normal	Desabilitado
AcroCEF.exe	17868	Em execução	breno	00	636 K	Normal	Desabilitado
ai.exe	22184	Em execução	breno	00	3.172 K	Normal	Desabilitado
ai.exe	18132	Em execução	breno	00	17.452 K	Normal	Desabilitado
amdfendrsr.exe	2720	Em execução	SISTEMA	00	20 K	Normal	Não permitido
amdow.exe	14540	Em execução	breno	00	384 K	Normal	Desabilitado
AMDRSServ.exe	3968	Em execução	breno	00	14.100 K	Normal	Desabilitado
AMDRSSrcExt.exe	20260	Em execução	breno	00	1.212 K	Normal	Desabilitado
ApplicationFrameHo...	16184	Em execução	breno	00	348 K	Normal	Desabilitado
armsvc.exe	4612	Em execução	SISTEMA	00	24 K	Normal	Não permitido
atieclxx.exe	8544	Em execução	SISTEMA	00	880 K	Normal	Não permitido
atiesnxx.exe	2728	Em execução	SISTEMA	00	160 K	Normal	Não permitido
audiogd.exe	2188	Em execução	SERVIÇO L...	00	4.364 K	Normal	Não permitido
avp.exe	4632	Em execução	SISTEMA	00	156.368 K	Normal	Não permitido
avpui.exe	12920	Em execução	breno	00	7.920 K	Normal	Desabilitado
backgroundTaskHos...	10160	Suspensão	breno	00	0 K	Normal	Desabilitado
backgroundTaskHos...	16228	Em execução	breno	00	904 K	Normal	Desabilitado
CalculatorApp.exe	5012	Suspensão	breno	00	0 K	Normal	Desabilitado
cmd.exe	2128	Em execução	breno	00	216 K	Normal	Desabilitado
cncmd.exe	17128	Em execução	breno	00	180 K	Normal	Desabilitado
CompPkgSrv.exe	15320	Em execução	breno	00	20 K	Normal	Desabilitado
conhost.exe	20824	Em execução	breno	00	344 K	Normal	Desabilitado
CPU-Metrics-Server.exe	17268	Em execução	breno	00	16 K	Acima do normal	Não permitido
csrss.exe	796	Em execução	SISTEMA	00	876 K	Normal	Não permitido
csrss.exe	16792	Em execução	SISTEMA	00	804 K	Normal	Não permitido
ctfmon.exe	20404	Em execução	breno	00	3.412 K	Alta	Desabilitado
Discord.exe	16816	Em execução	breno	00	6.736 K	Normal	Desabilitado
Discord.exe	19448	Em execução	breno	00	284 K	Normal	Desabilitado
Discord.exe	2876	Em execução	breno	00	2.504 K	Acima do normal	Desabilitado
Discord.exe	2096	Em execução	breno	00	1.952 K	Normal	Desabilitado
Discord.exe	21612	Em execução	breno	00	146.364 K	Normal	Desabilitado
Discord.exe	21496	Em execução	breno	00	1.172 K	Normal	Desabilitado
dllhost.exe	1664	Em execução	SISTEMA	00	740 K	Normal	Não permitido
dllhost.exe	12044	Em execução	SISTEMA	00	144 K	Normal	Não permitido
dllhost.exe	15884	Em execução	breno	00	2.524 K	Normal	Desabilitado
Docker Desktop.exe	22192	Em execução	breno	00	11.892 K	Normal	Desabilitado
Docker Desktop.exe	20780	Em execução	breno	00	1.908 K	Acima do normal	Desabilitado
Docker Desktop.exe	16960	Em execução	breno	00	108 K	Normal	Desabilitado
Docker Desktop.exe	16188	Em execução	breno	00	306 K	Normal	Desabilitado

Referências:

[Visualizador de classificação - Classificação de pilha \(sortvisualizer.com\)](http://sortvisualizer.com)

[Visualizador de classificação - Classificação Rápida \(sortvisualizer.com\)](http://sortvisualizer.com)

[Introsort – Wikipedia](#)

https://www.facom.ufu.br/~abdala/DAS5102/TEO_HeapFilaDePrioridade.pdf

<https://joaoarthurbm.github.io/eda/posts/heap/>

[http://www.decom.ufop.br/anascimento/site_media/uploads/bcc202/aula_16 - fila de prioridade
_e_heapsort.pdf](http://www.decom.ufop.br/anascimento/site_media/uploads/bcc202/aula_16_-_fila_de_prioridade_e_heapsort.pdf)

<https://visualizer.siddhartha-chatterjee.com/visualize/Heap>

**FIM DOS SLIDES, AGORA
VOCÊS APLAÚDEM**



**NÃO PERGUNTEM E NOS
PAGUEM UMA CERVEJA**