MÉTODOS DE ORDENAÇÃO DE DADOS INTERNOS

Heapsort & Shellsort

INTEGRANTES

- Evandro Padilha Barroso Filho
- Joelson Pereira Lima
- Matheus Palheta Barbosa
- Vinícius Edwards Guimarães Sousa

ROTEIRO

- **HEAPSORT**
  - *Onde e como usar;*
  - *Tipo de ordenação;*
  - *Forma de ordenação;*
  - *Complexidade;*
  - *Estabilidade;*
  - *Vantagens e desvantagens;*
  - *Exemplo de aplicação real;*
  - *Código comentado em C e em Python;*
  - *Exemplo passo a passo.*

ROTEIRO

- **SHELLSORT**
  - *Onde e como usar;*
  - *Tipo de ordenação;*
  - *Forma de ordenação;*
  - *Complexidade;*
  - *Estabilidade;*
  - *Vantagens e desvantagens;*
  - *Exemplo de aplicação real;*
  - *Código comentado em C e em Python;*
  - *Exemplo passo a passo.*

HEAPSORT

COMO E ONDE USAR

- É um algoritmo de ordenação que faz parte da família de algoritmos de ordenação por seleção;
- As operações das filas de prioridades podem ser utilizadas para implementar algoritmos de ordenação.
- O uso de *heap* para fazer a ordenação origina o método HeapSort.

TIPO DE ORDENAÇÃO

- Ordenação Parcial, ou seja, ele divide a entrada em região ordenada e região não-ordenada, e diminui a região não-ordenada extraindo o maior (ou menor) elemento e adicionando-o à região ordenada.

FORMA DE ORDENAÇÃO

- O Heapsort utiliza uma estrutura de dados chamada *heap*, para ordenar os elementos à medida que os insere na estrutura.
- A *heap* pode ser representada como uma árvore binária com propriedades especiais (ou como um vetor).

FILA DE PRIORIDADES

- Uma fila é um tipo de estrutura de dados que segue o princípio FIFO (*First In, First Out*) onde a ordem de saída dos elementos é a mesma de entrada;
- Já em uma fila de prioridade, cada elemento possui uma prioridade onde é determinado a sua ordem saída;
- Geralmente implementados como *heap.*

FORMA DE ORDENAÇÃO

- Para uma ordenação decrescente, deve ser construída uma *heap* mínima (o menor elemento fica na raiz).
- Para uma ordenação crescente, deve ser construído uma *heap* máxima (o maior elemento fica na raiz).

COMPLEXIDADE

- A complexidade de tempo é igual para todos os casos.

Melhor Caso	Pior Caso	Caso Médio
O(n log n)	O(n log n)	O(n log n)

ESTABILIDADE

- O Heapsort não é um algoritmo de ordenação estável.
- Porém, é possível adaptar a estrutura a ser ordenada de forma a tornar a ordenação estável. Cada elemento da estrutura adaptada deve ficar no formato de um par (elemento original, índice original). Assim, caso dois elementos sejam iguais, o desempate ocorrerá pelo índice na estrutura original.

VANTAGENS E DESVANTAGENS

Vantagens

- O comportamento do Heapsort é sempre O(n log n), qualquer que seja a entrada.
- Recomendado para aplicações que não podem tolerar eventualmente um caso desfavorável.

Desvantagens

- O anel interno do algoritmo é bastante complexo se comparado com o do Quicksort.
- Não é recomendado para arquivos com poucos registros, por causa do tempo necessário para construir o *heap*.

EXEMPLO DE APLICAÇÃO REAL

- Heapsort não é muito usado na prática, mas pode ser útil em sistemas embarcados em tempo real onde há pouco espaço disponível.

CÓDIGO COMENTADO EM C

```c
// função principapl do heapSort
void heapSort(int vetor[], int tam)
{
    // corrige o vetor para heap
    for (int i = tam / 2 - 1; i >= 0; i--)
        constroiHeap(vetor, tam, i);
    // corrige as sub arvores do heap de baixo pra cima
    for (int i=tam-1; i>=0; i--)
    {
        // move a raiz atual para o fim
        troca(vetor, 0, i);
        // executa o constroiHeap na subarvore deste elemento;
        constroiHeap(vetor, i, 0);
    }
}
```

```c
//rearranja os elementos de um vetor (ou subvetor) para que ele se torne
void constroiHeap(int vetor[], int tam, int i)
{
    int maior = i; // o elemento raiz que estamos verificando
    int esq = 2*i + 1; // o filho esquerdo dele no heap
    int dir = 2*i + 2; // o filho direito dele no heap
    // verifica se o filho esquerdo é maior que a raiz
    if (esq < tam && vetor[esq] > vetor[maior])
        maior = esq;
    // verifica se o filho direito é maior que a raiz
    if (dir < tam && vetor[dir] > vetor[maior])
        maior = dir;
    // se o maior elemento nao é a raiz
    if (maior != i)
    {
        troca(vetor, i, maior);
        // recursivamente corrige o nó alterado
        constroiHeap(vetor, tam, maior);
    }
    mostrarVetor(vetor, tam);
}
```

```c
//Função para trocar dois valores do vetor
void troca(int vetor[], int i, int j){
    int temp = vetor[i];
    vetor[i] = vetor[j];
    vetor[j] = temp;
}

//função para imprimir o vetor
void mostrarVetor(int vetor[], int tam)
{
    for (int i=0; i<tam; ++i)
        printf("%d, ", vetor[i]);
    printf("\n");
}
```

CÓDIGO COMENTADO EM PYTHON

```python
def heapsort(vetor):
    tam = len(vetor)
    for i in range(int(tam/2)-1, -1, -1):
        constroi_heap(vetor, tam, i)

    for i in range(tam-1, -1, -1):
        vetor[i], vetor[0] = vetor[0], vetor[i]
        constroi_heap(vetor, i, 0)
    return vetor
```

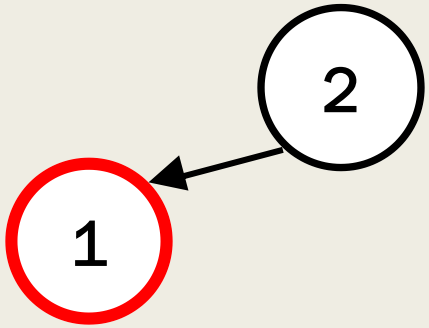
```python
def constroi_heap(vetor, tam, i):
    maior = i
    esq = 2*i +1
    dir = 2*i +2
    if esq < tam and vetor[esq] > vetor[maior]:
        maior = esq
    if dir < tam and vetor[dir] > vetor[maior]:
        maior = dir

    if maior != i:
        vetor[i], vetor[maior] = vetor[maior], vetor[i]
        constroi_heap(vetor, tam, maior)
```


2
1
8
5
7

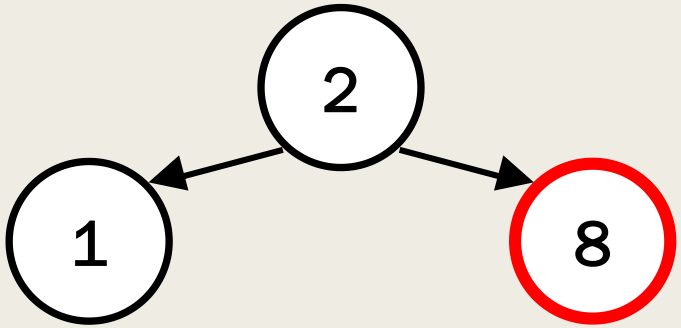

2
1
8
5
7


2

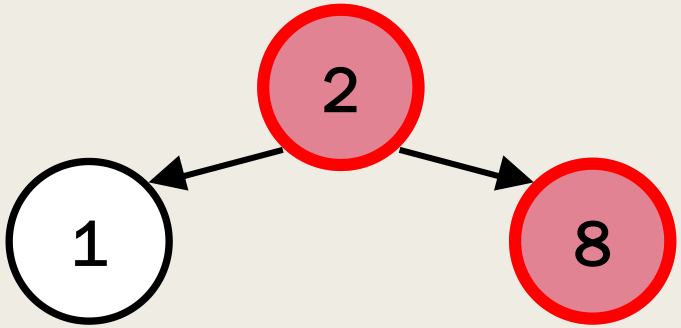

2
1
8
5
7

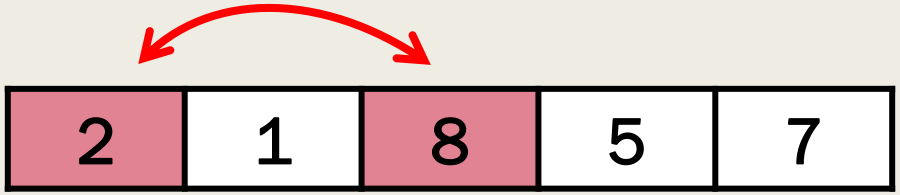

2
1

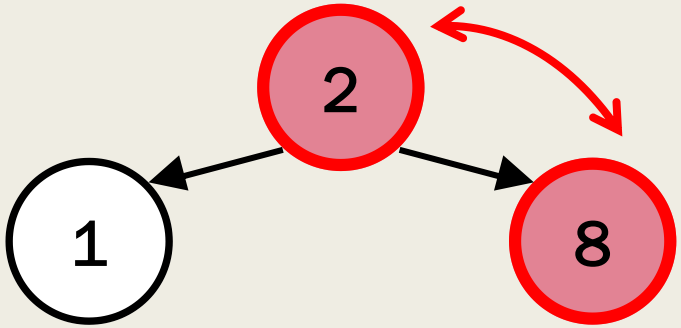

2
1
8
5
7

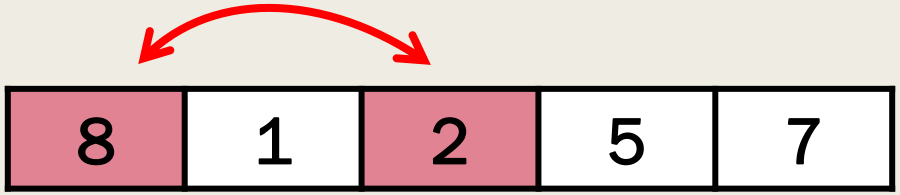

2
1
8

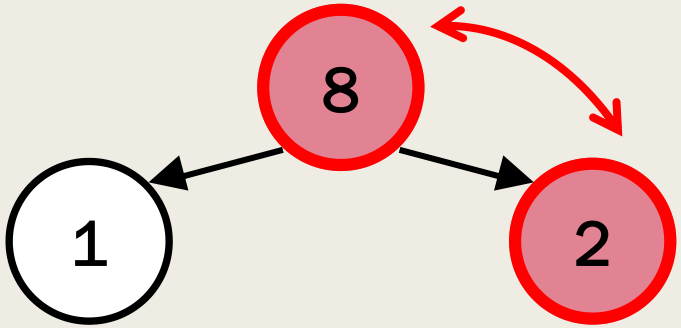

2
1
8
5
7

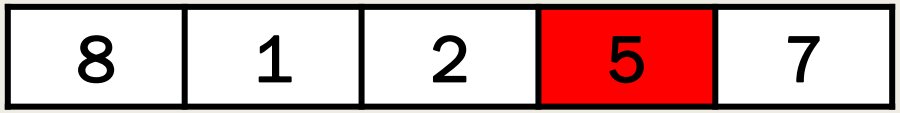

2
1
8

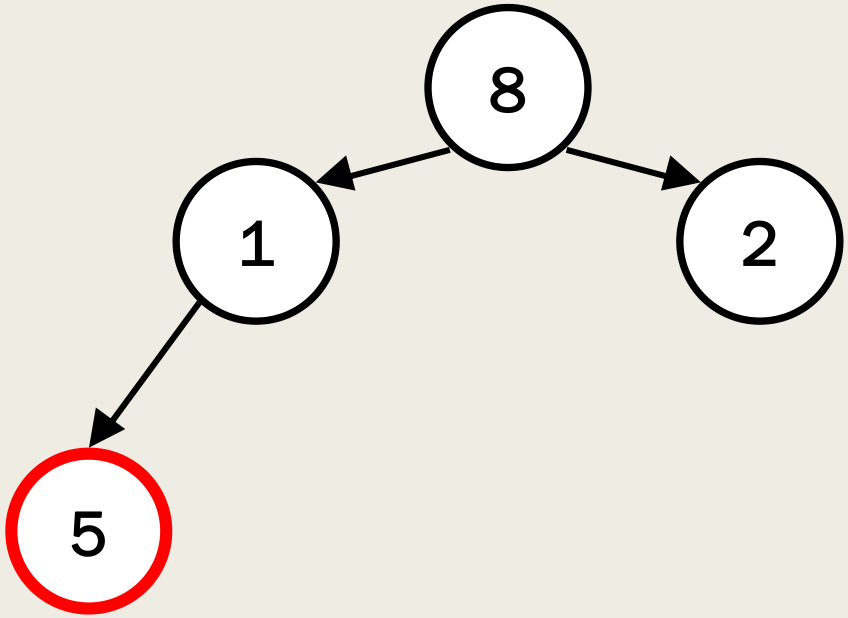

2
1
8
5
7

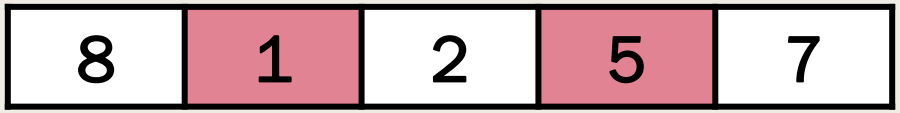

2
1
8

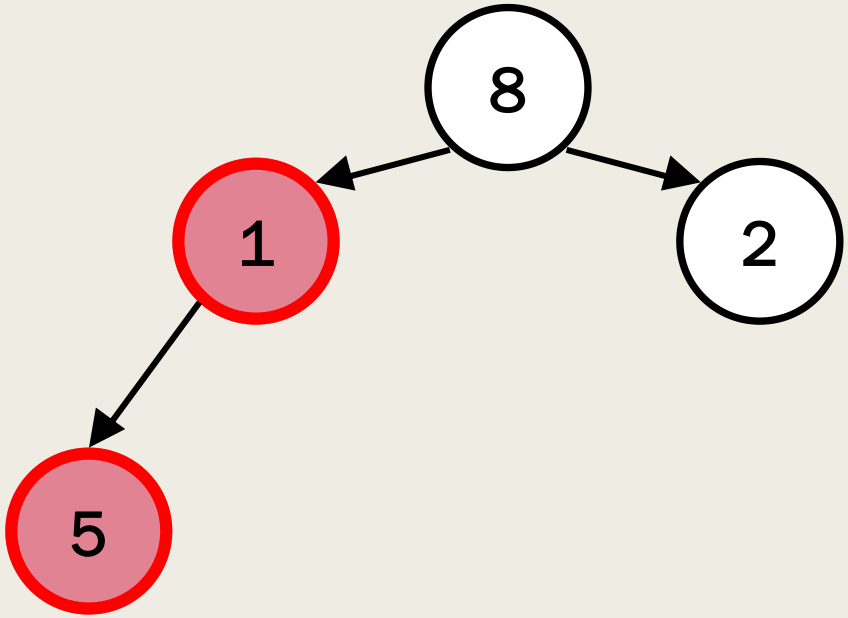

8
1
2
5
7

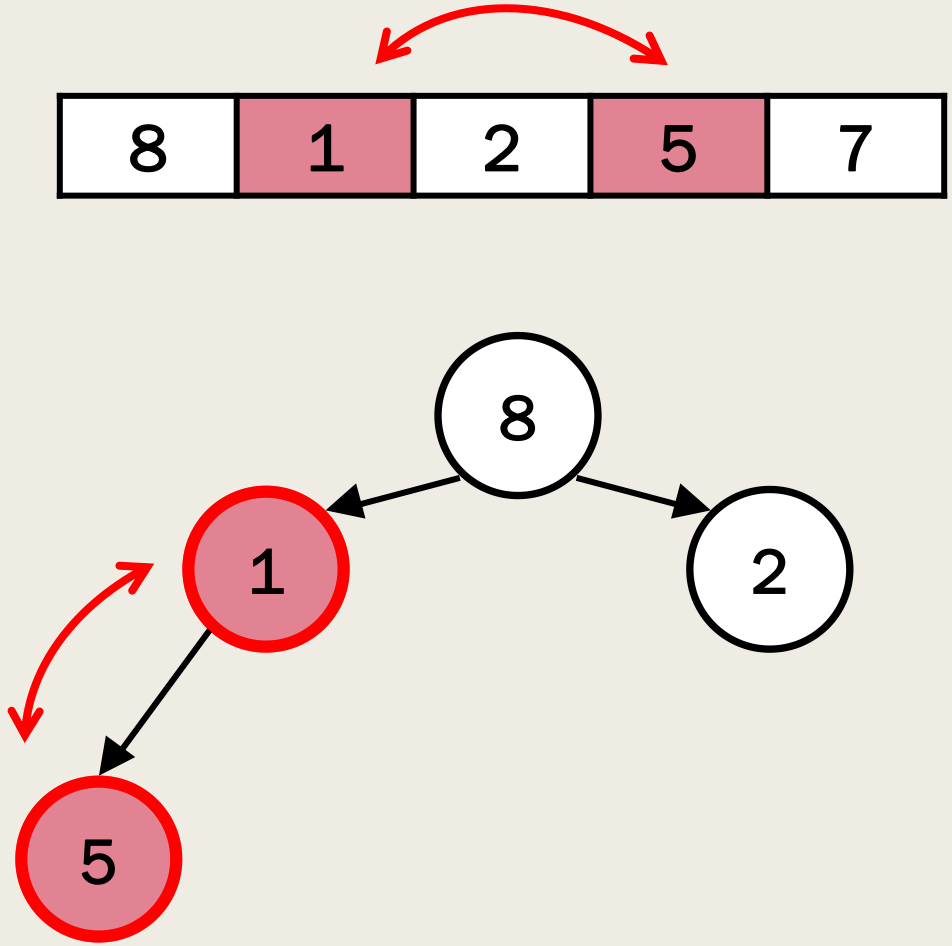

8
1
2

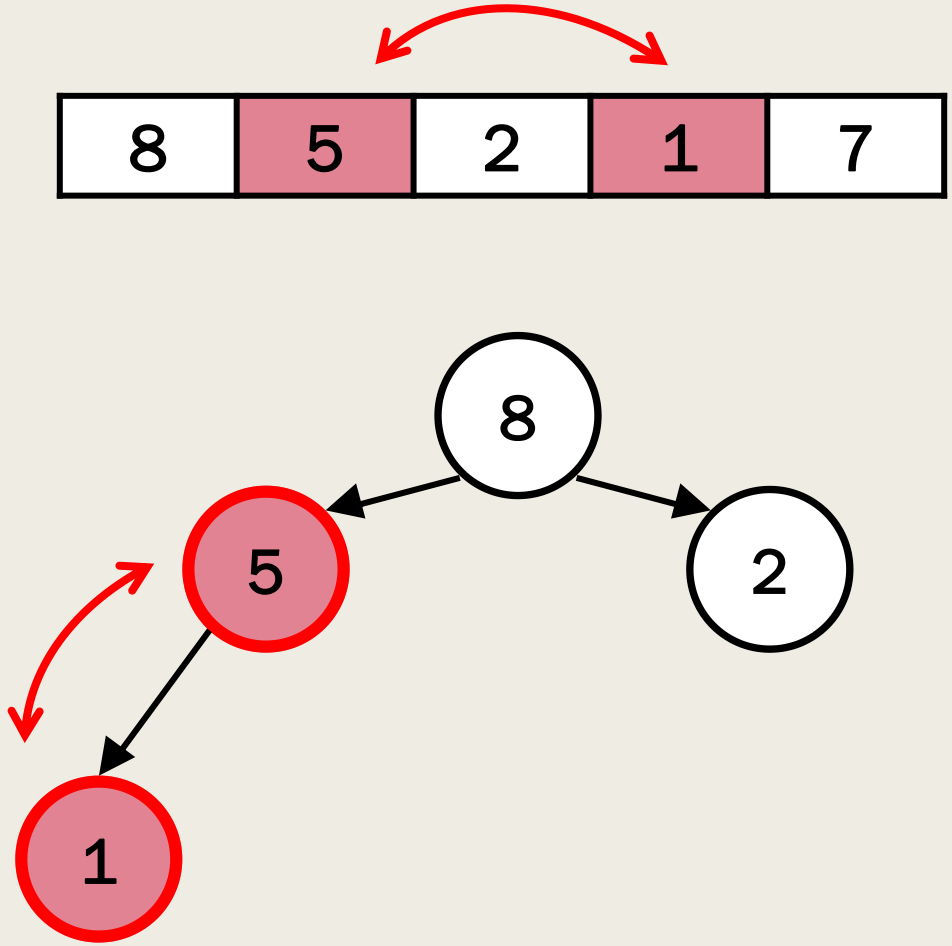

8
1
2
5
7

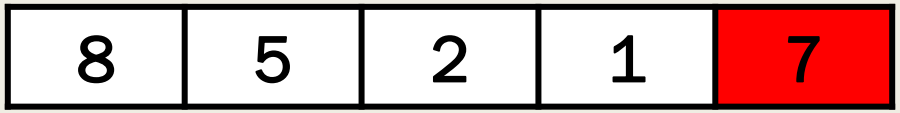

8
1
2
5

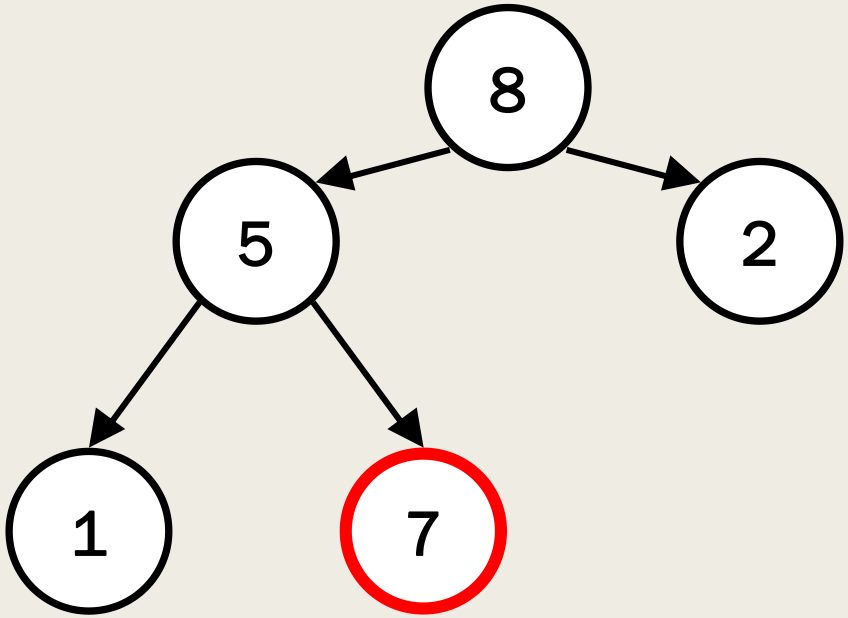

8
1
2
5
7

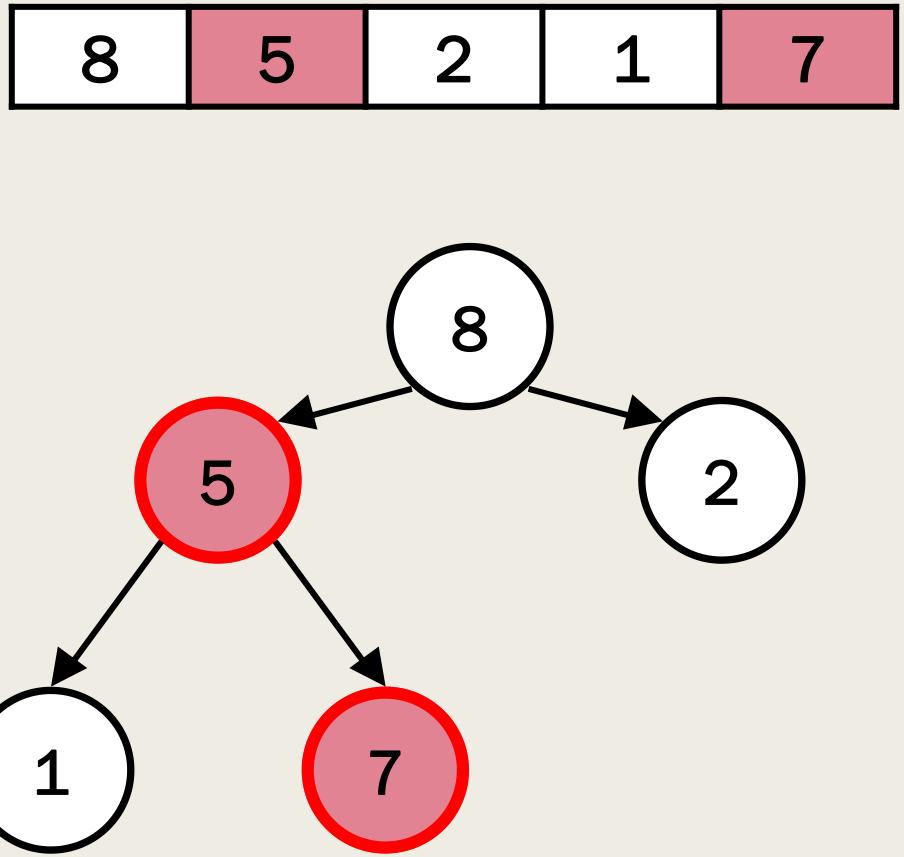

8
1
2
5

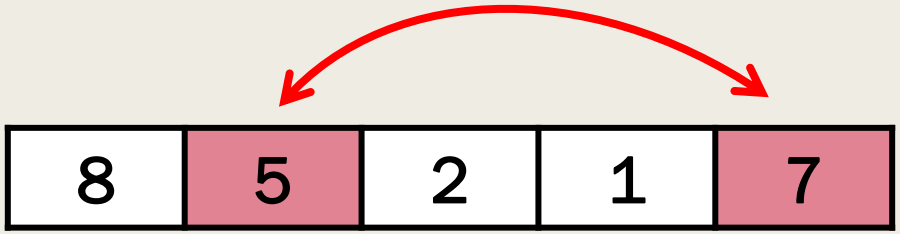

8
1
2
5
7
8
1
2
5

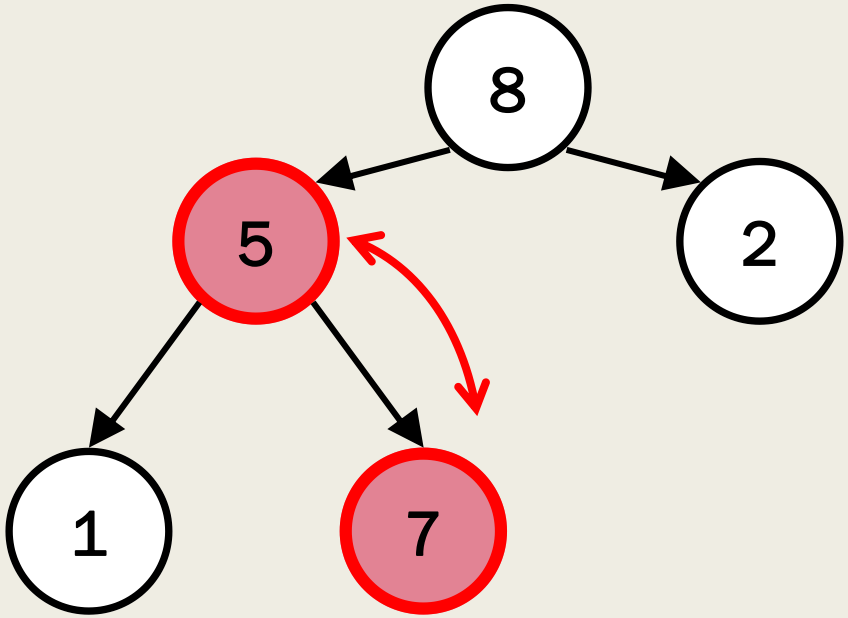

8
5
2
1
7
8
5
2
1

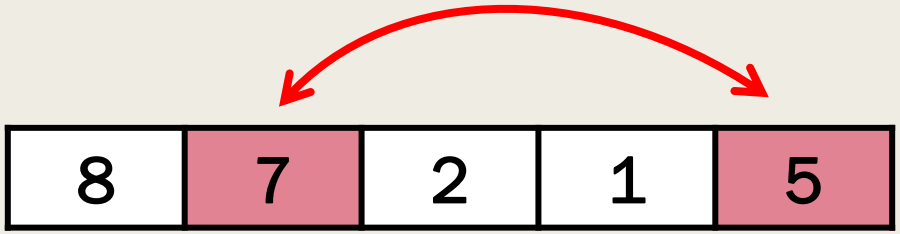

8
5
2
1
7

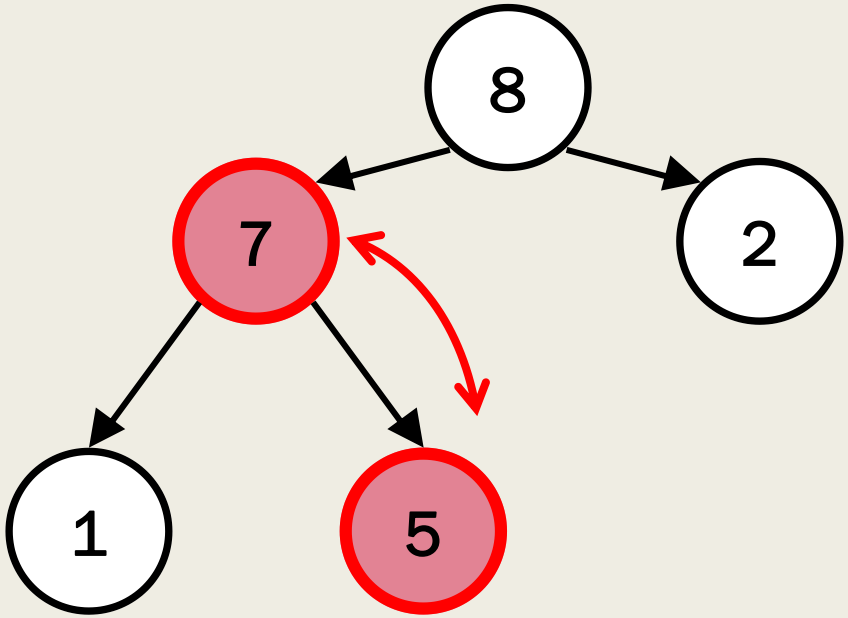

8
5
2
1
7


8
5
2
1
7
8
5
2
1
7

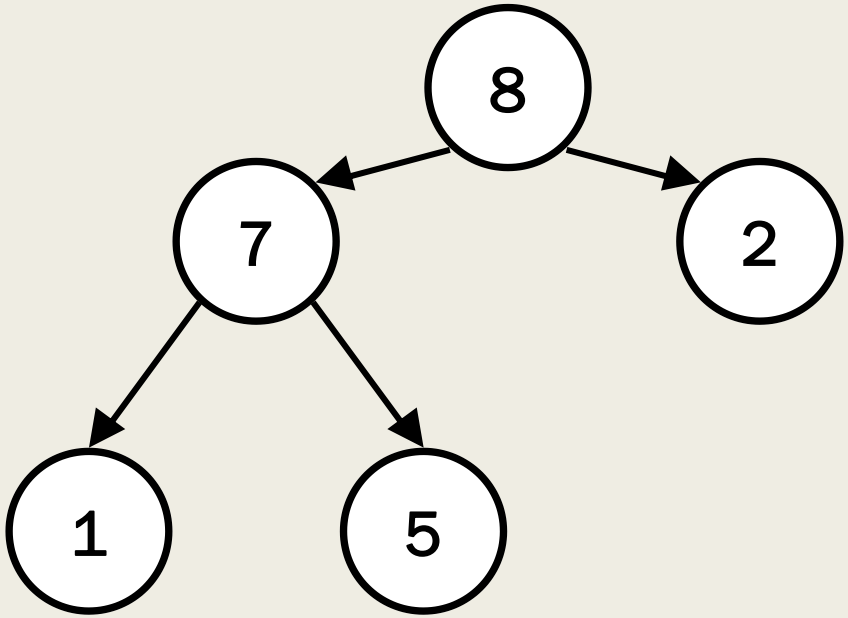

8
5
2
1
7

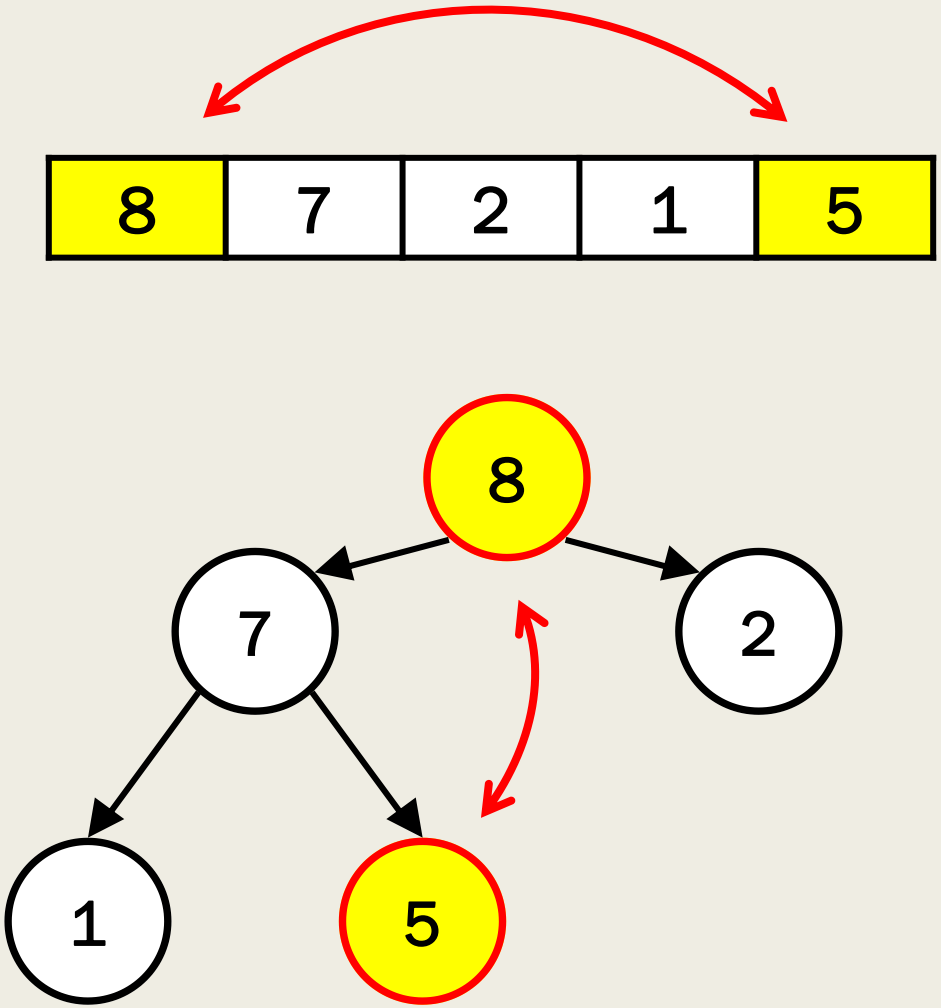

8
5
2
1
7

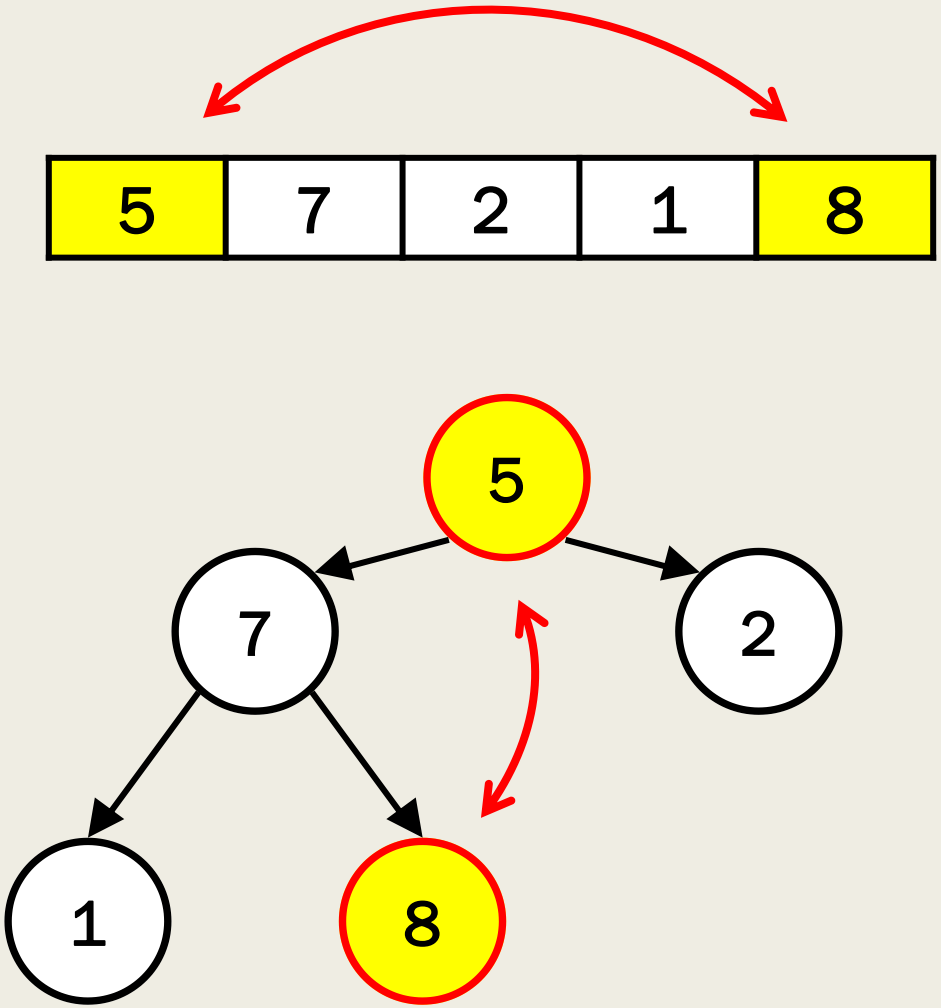

8
7
2
1
5

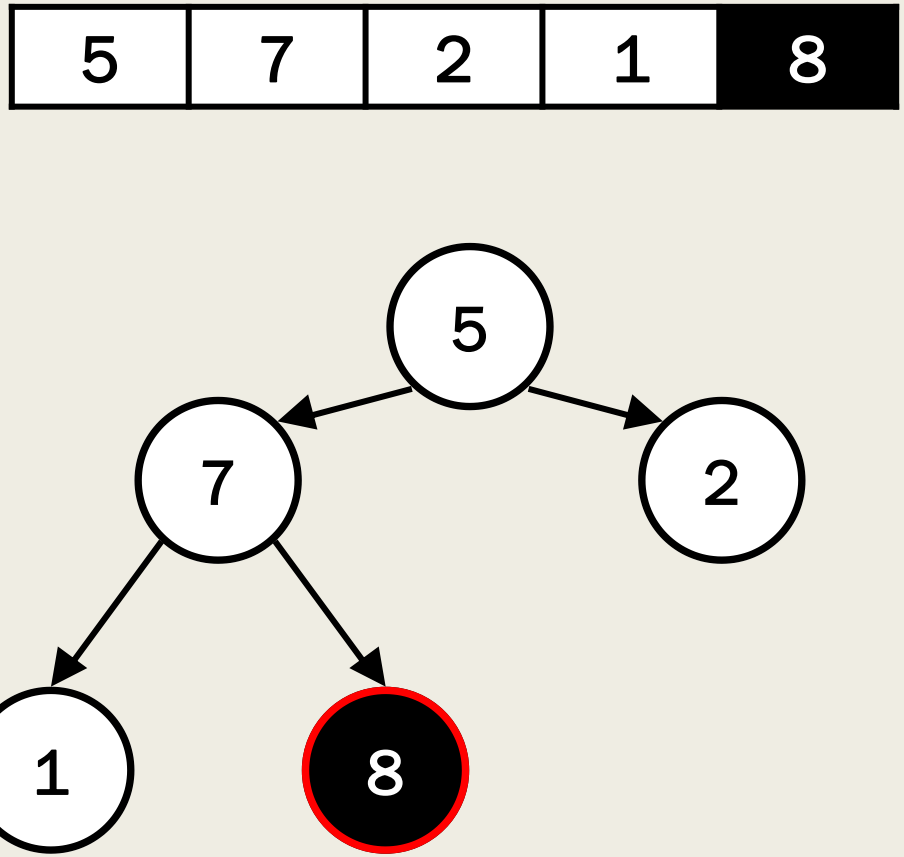

8
7
2
1
5

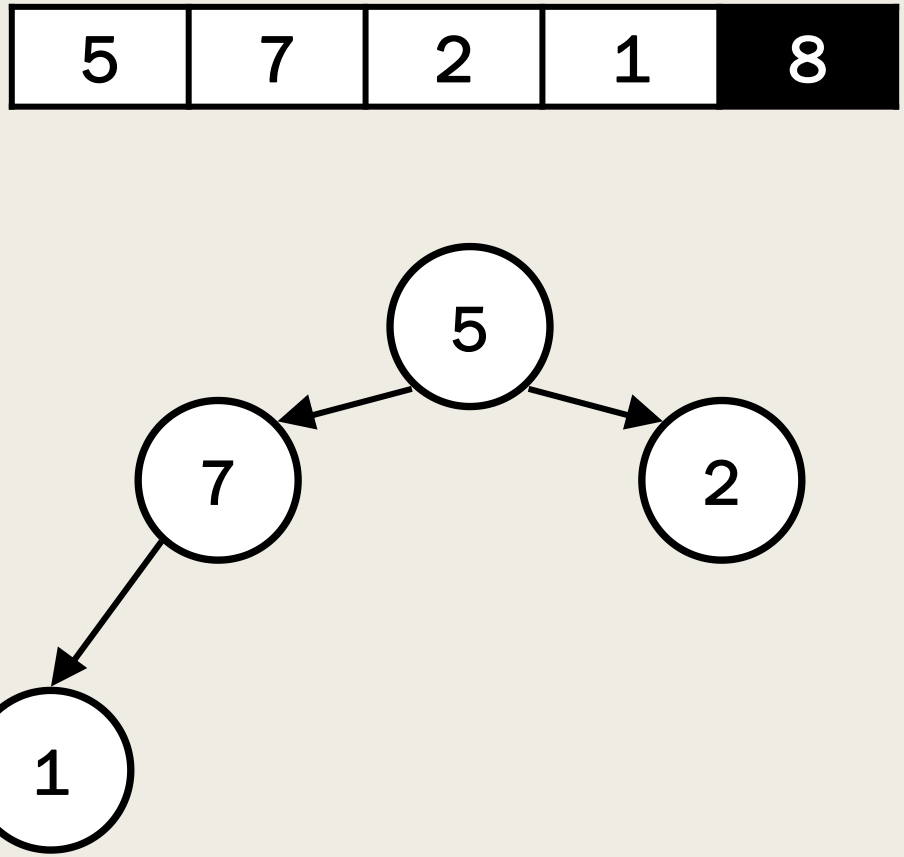
8	7	2	1	5

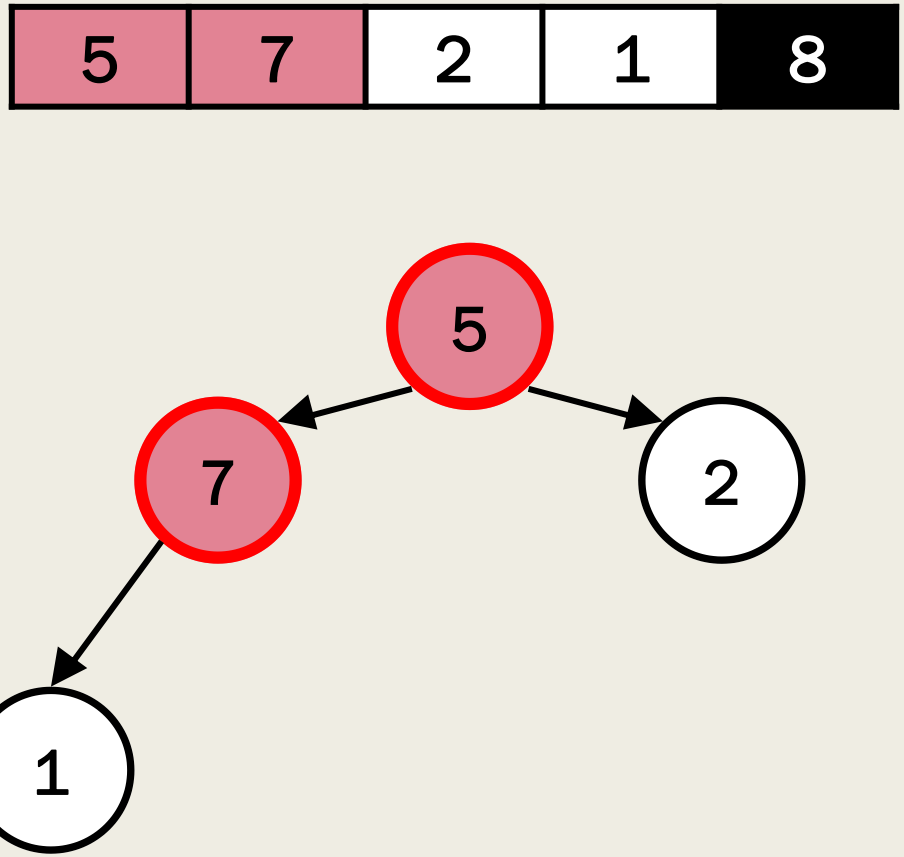


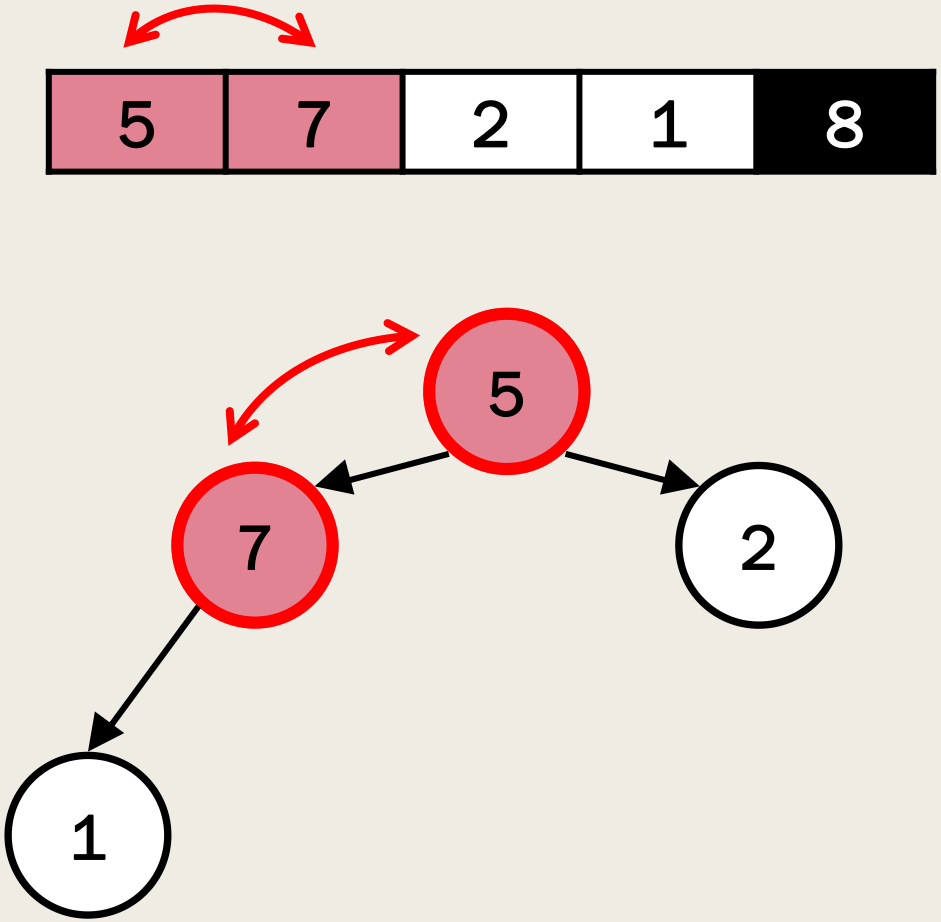

8
7
2
1
5
8
7
2
1
5

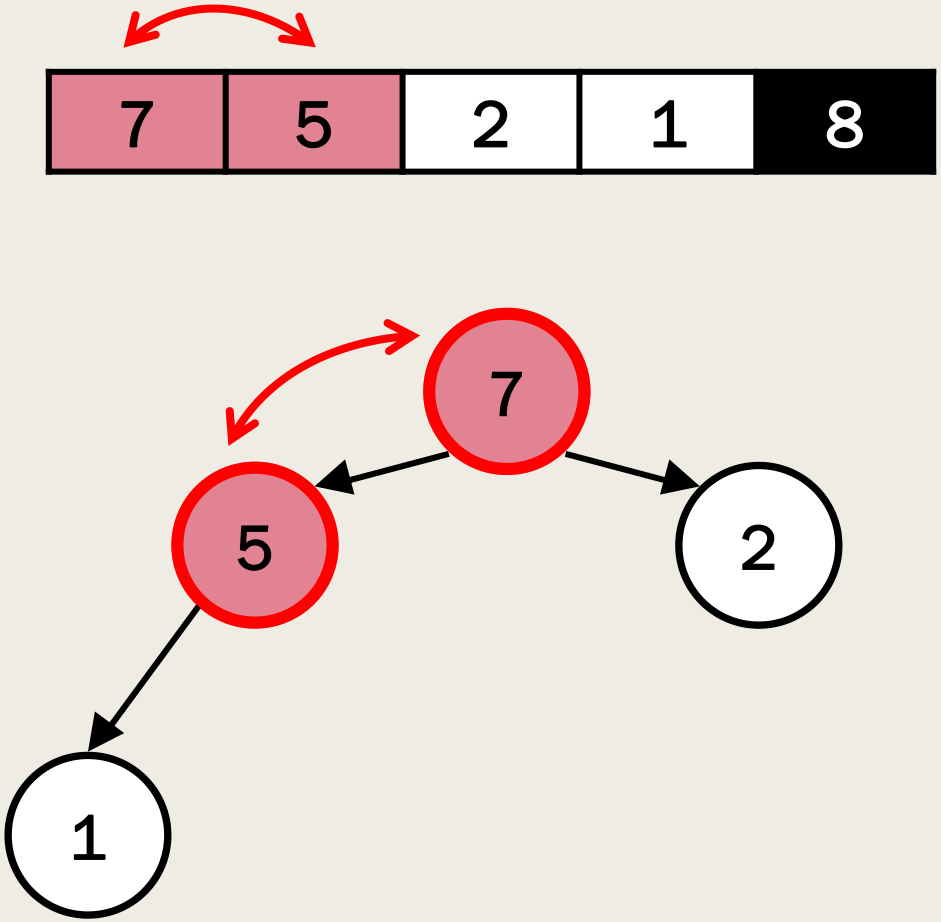

5
7
2
1
8
5
7
2
1
8

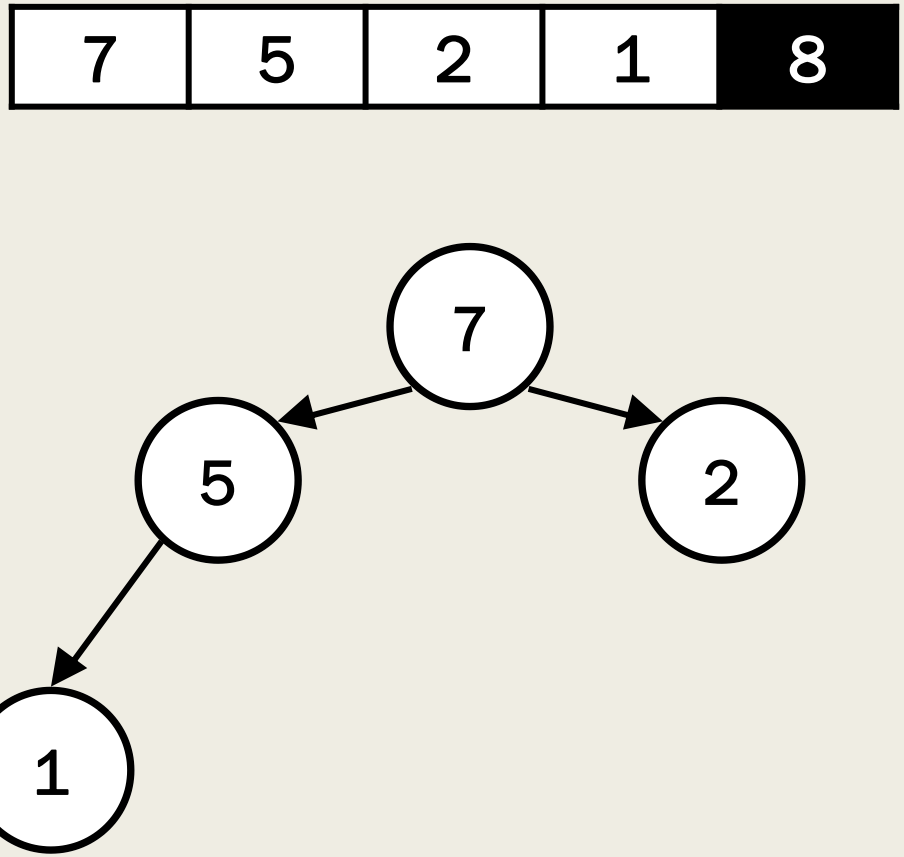

5
7
2
1
8
5
7
2
1
8

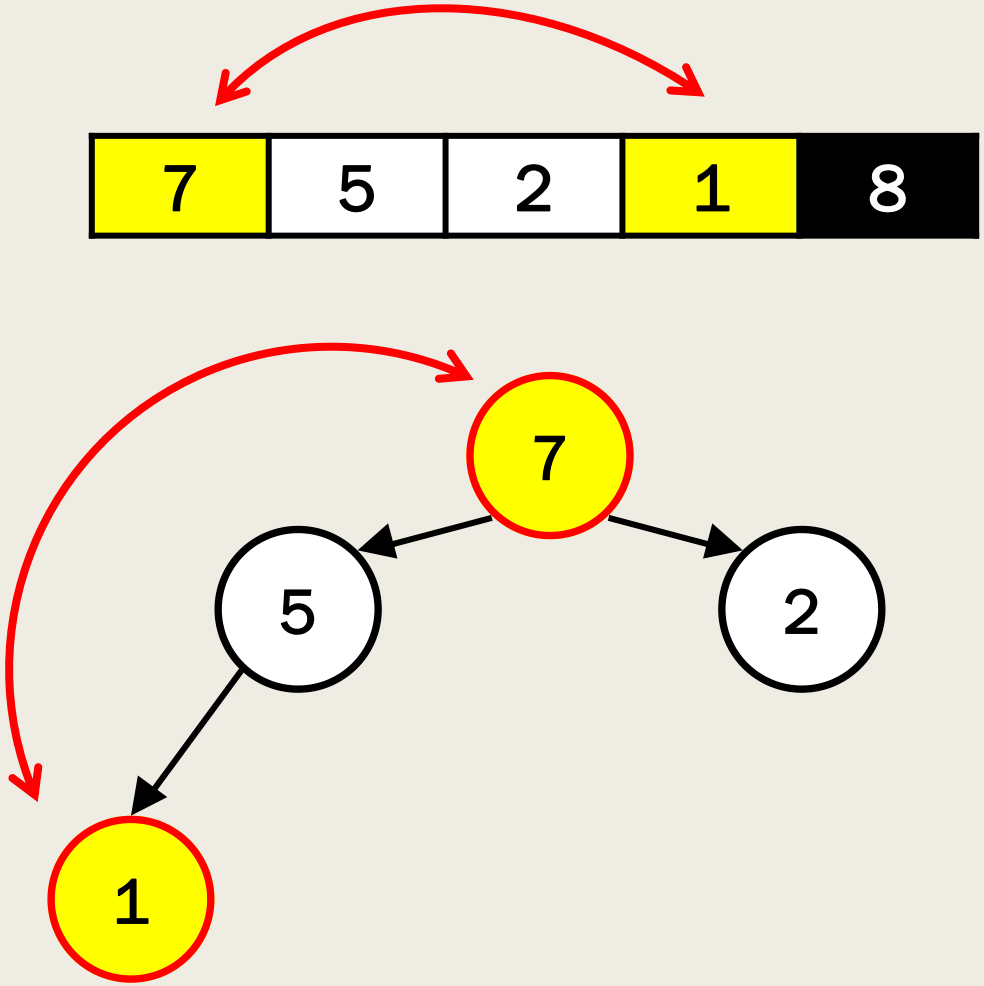

5
7
2
1
8
5
7
2
1

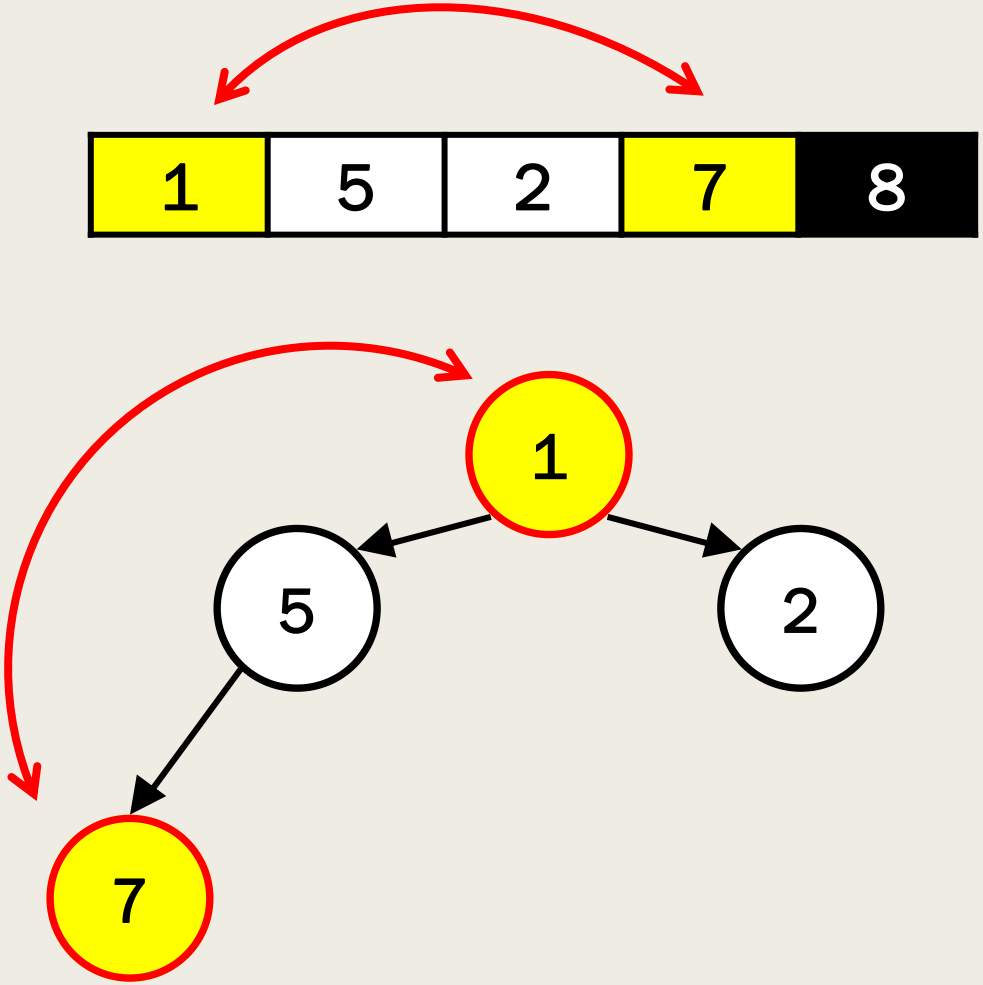

5
7
2
1
8
5
7
2
1

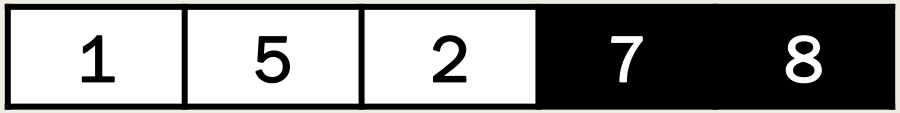

5
7
2
1
8
5
7
2
1

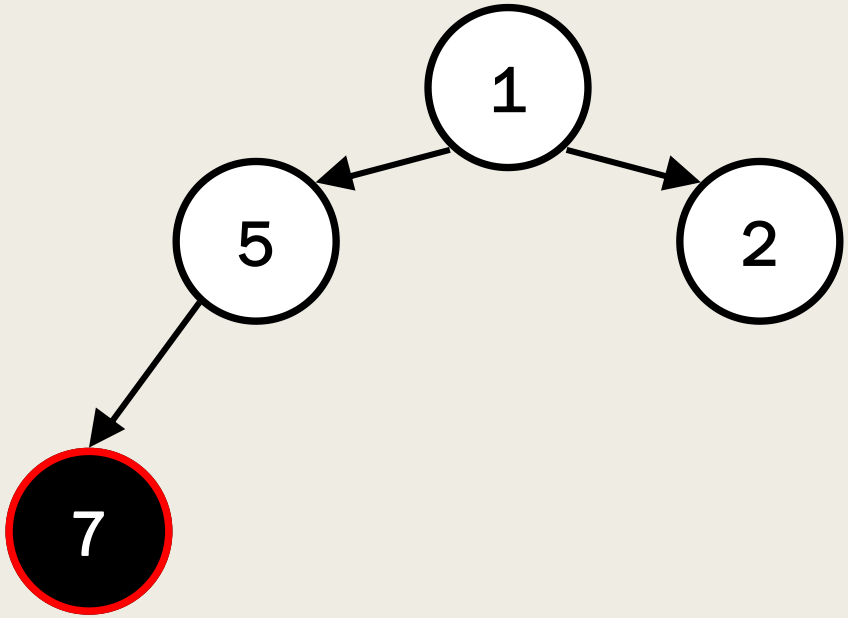

7
5
2
1
8
7
5
2
1

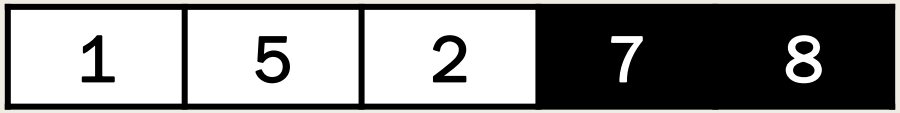

7
5
2
1
8
7
5
2
1

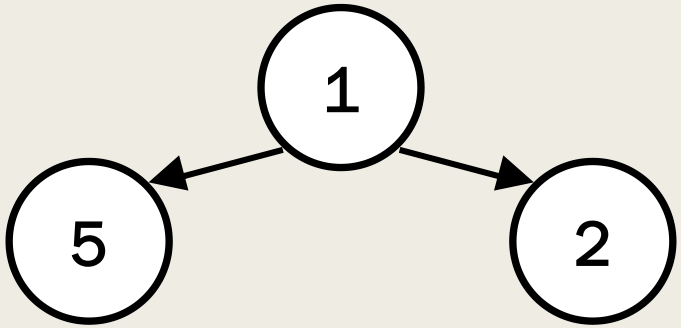

7
5
2
1
8
7
5
2
1

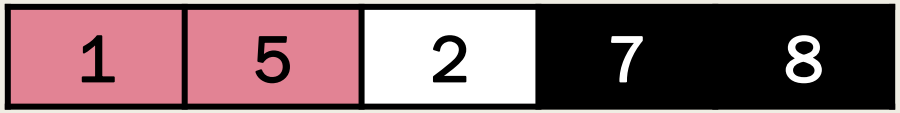

1
5
2
7
8
1
5
2
7

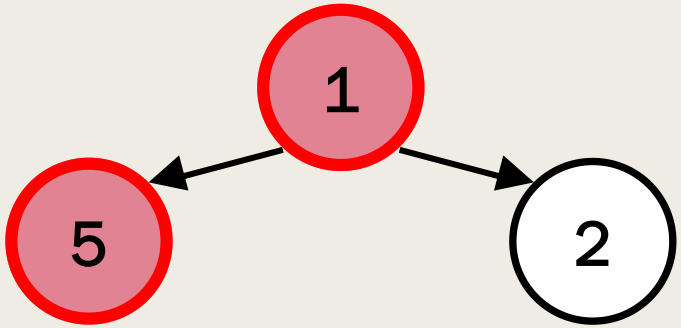

1
5
2
7
8

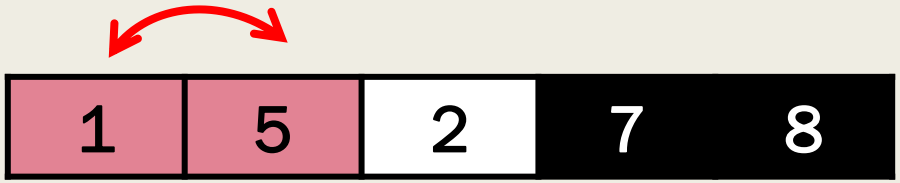

1
5
2
7

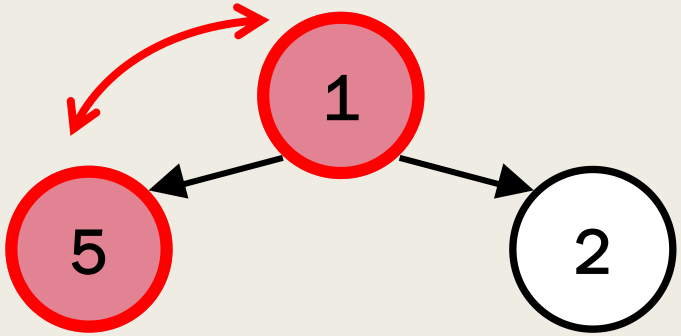

1
5
2
7
8

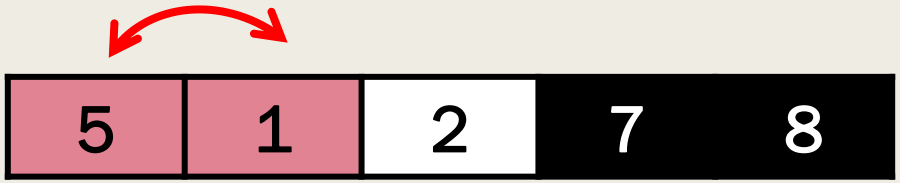

1
5
2

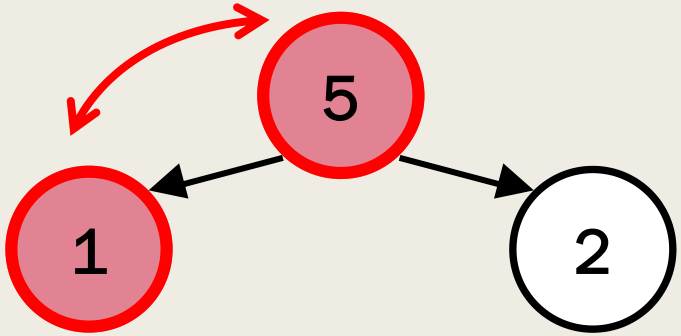

1
5
2
7
8

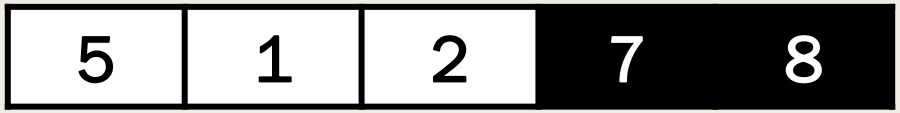

1
5
2

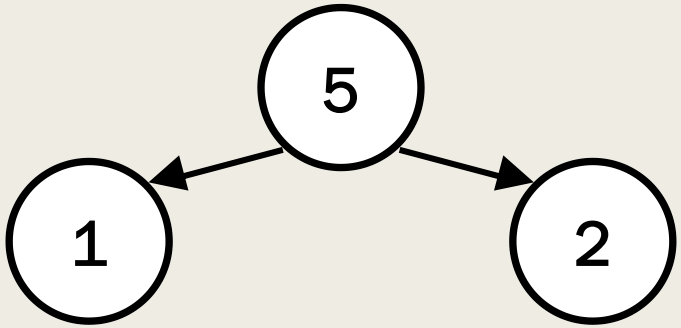

1
5
2
7
8

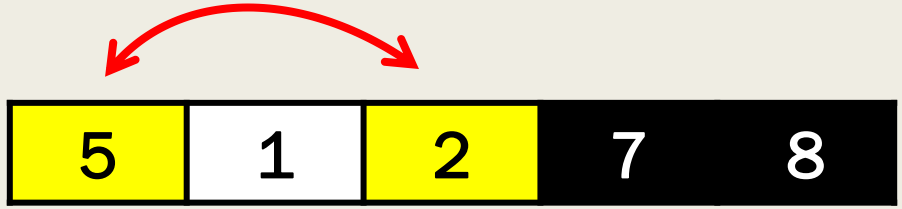

1
5
2

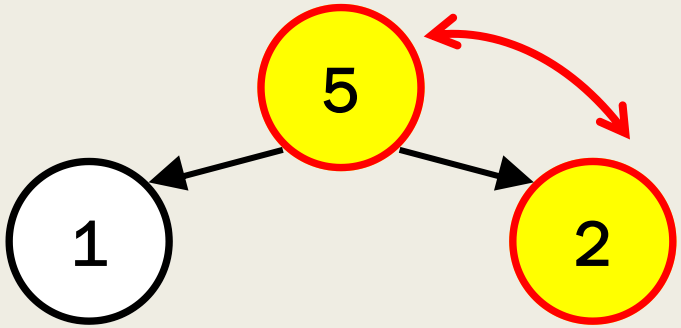

5
1
2
7
8

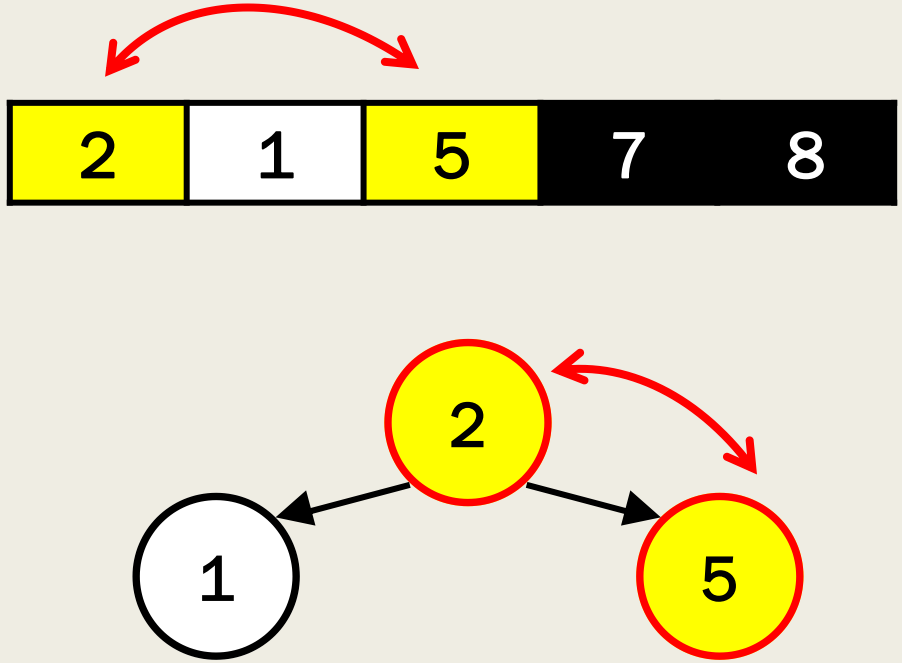

5
1
2


5
1
2
7
8

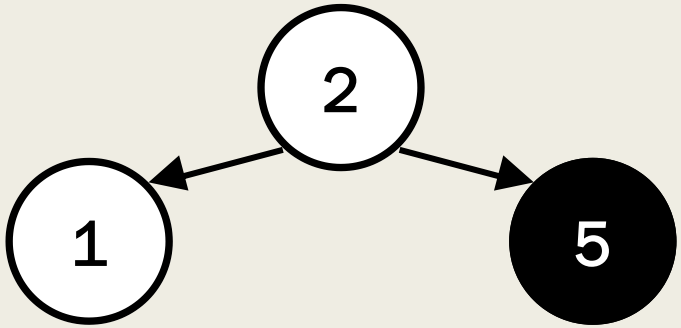

5
1
2


5
1
2
7
8

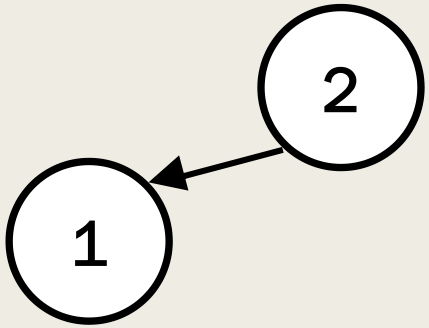

5
1
2


2
1
5
7
8
2
1
5

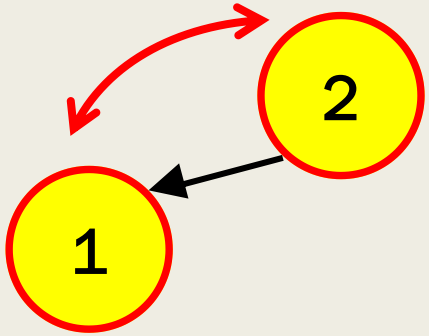

2
1
5
7
8


2
1
5

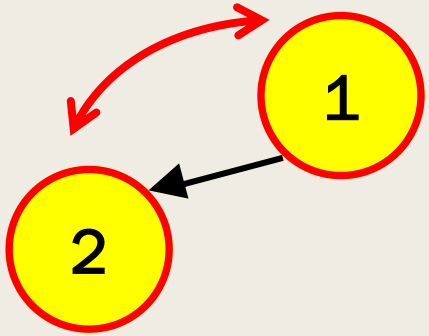

2
1
5
7
8


2
1

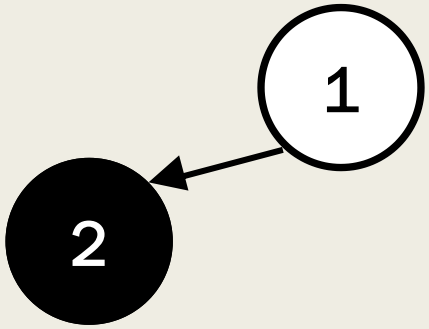

2
1
5
7
8


2
1


1
2
5
7
8

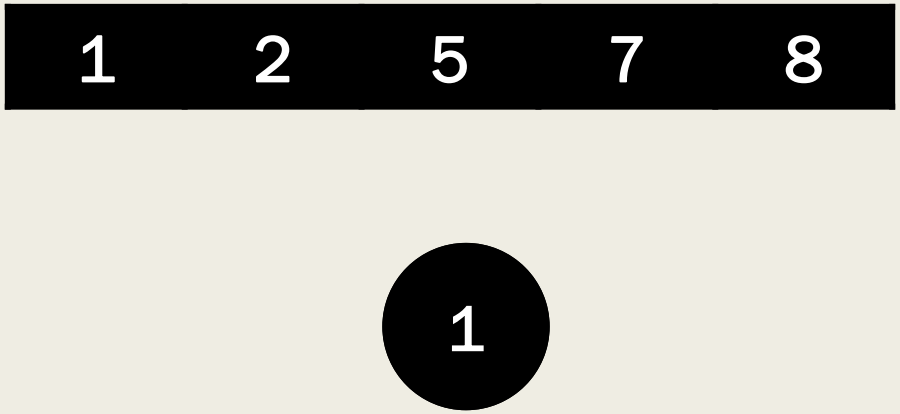

1
2


1
2
5
7
8


1
2


1
2
5
7
8


1


1 2 5 7 8
1

AVALIAÇÃO COM 100,000 NÚMEROS - C

	TESTE 01	TESTE 02	TESTE 03
CRESCENTE	0,015000	0.031000	0.031000
DECRESCENTE	0,015000	0.015000	0.015000
CRESCENTE DECRESCENTE	0,015000	0.022000	0.031000
DECRESCENTE CRESCENTE	0,015000	0.015000	0.020000
DESORDENADO	0,027000	0.031000	0.025000

AVALIAÇÃO COM 100,000 NÚMEROS - PYTHON

	TESTE 01	TESTE 02	TESTE 03
CRESCENTE	1,192533	1,174774	1,216801
DECRESCENTE	1,082931	1,094917	1,084163
CRESCENTE DECRESCENTE	1,140608	1,166528	1,148628
DECRESCENTE CRESCENTE	1,109372	1,117068	1,127719
DESORDENADO	1,174774	1,180946	1,187492

SHELLSORT

ONDE E COMO USAR

- É um algoritimo de ordenação derivado do Insertion Sort;
- Esse método começa ordenando pares de elementos distantes e, progressivamente reduzindo o *gap* entre os elementos a serem comparados.
- Começando com elementos distantes, ele pode mover elementos fora do lugar para a posição mais rapidamente do que uma simples troca de elementos vizinhos;

TIPO DE ORDENAÇÃO

- Ordenação Parcial, ou seja, ele divide a entrada em região ordenada e região não-ordenada, e diminui a região não-ordenada extraindo o maior (ou menor) elemento e adicionando-o à região ordenada.

FORMA DE ORDENAÇÃO

- Ordenação por inserção por decrementos decrescentes.
- O valor do gap começa alto e vai diminuindo a cada iteração.

COMPLEXIDADE

- O tempo de execução do Shellsort depende muito da sequência de *gap* que é usado. Determinar a complexidade de tempo continua sendo um problema.

Melhor Caso	Pior Caso	Caso Médio
O(n log_2 n)	O(n log_2 n)	Depende da sequência de gap

ESTABILIDADE

- O Shellsort não é um algoritmo de ordenação estável.
- Elementos de mesmo valor não tem suas posições originais mantidas.

VANTAGENS E DESVANTAGENS

VANTAGENS

- Ótimo para arquivos de tamanho moderado;
- Sua implementação é simples e requer uma quantidade de código pequena

DESVANTAGENS

- O tempo de execução do algoritmo é sensível à ordem inicial do arquivo;

EXEMPLO DE APLICAÇÃO REAL

- Como ele pode ser implementado usando pouco código e não usa pilha de chamadas, algumas implementações da função *qsort* na biblioteca padrão do C feita para sistemas embarcados o utiliza ao invés do Quicksort.

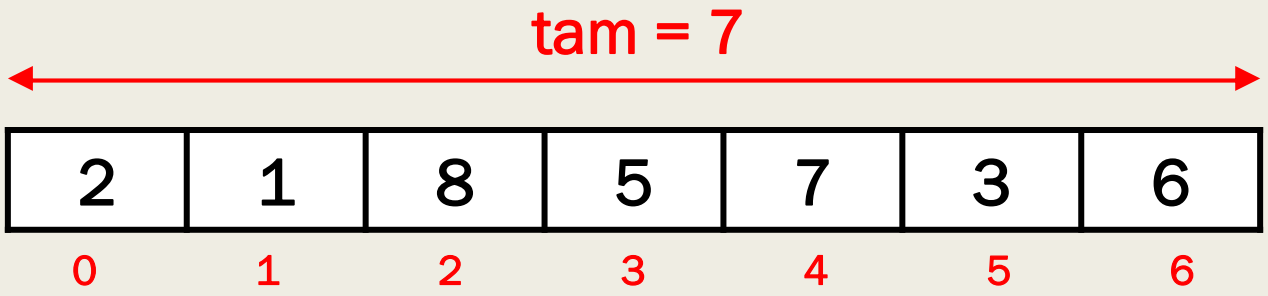
CÓDIGO COMENTADO EM C

```c
void shellSort(int vetor[], int tam) {
    int i , j , valor;

    int salto = 1;
    while(salto < tam) {
        salto = 3 * salto + 1; //inicializa o salto com um valor alto
    }
    while (salto > 0) {
        for(i = salto; i < tam; i++) { // faz uma varredura no vetor a pa
            valor = vetor[i];
            j = i;
            while (j > salto-1 && valor <= vetor[j - salto]) {
                vetor[j] = vetor[j - salto];
                j = j - salto;
            }
            vetor[j] = valor;
            mostrarVetor(vetor, tam);
        }
        salto = salto/3;
    }
}
```

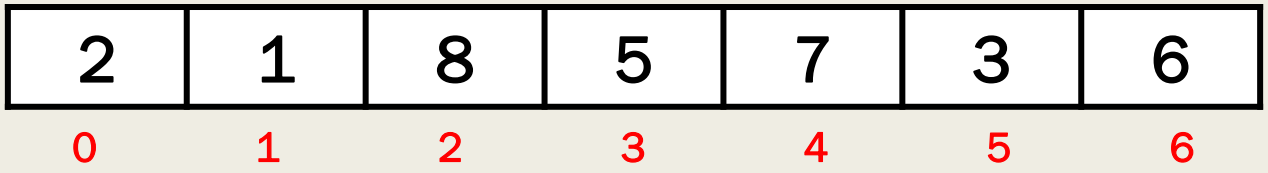
CÓDIGO COMENTADO EM PYTHON

```python
def shellsort(vetor):
    tam = len(vetor)
    salto = 1
    while(salto < tam):
        salto = 3 * salto + 1
    while salto > 0:
            for i in range(salto, tam):
                c = vetor[i]
                j = i
                while j > salto-1 and c <= vetor[j - salto]:
                    vetor[j] = vetor[j - salto]
                    j = j - salto
                vetor[j] = c
            salto = int(salto / 3)
```


2
1
8
5
7
3
6


tam = 7
2
1
8
5
7
3
6
0
1
2
3
4
5
6

gap = tam / 2



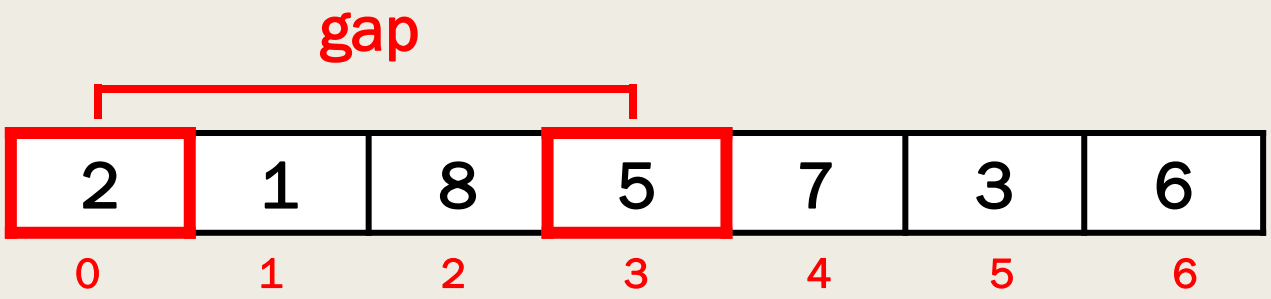
tam = 7

gap = 7 / 2 = 3

tam = 7

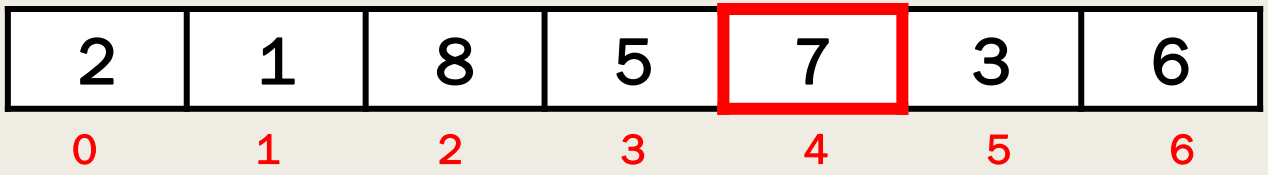
tam = 7

gap = 3



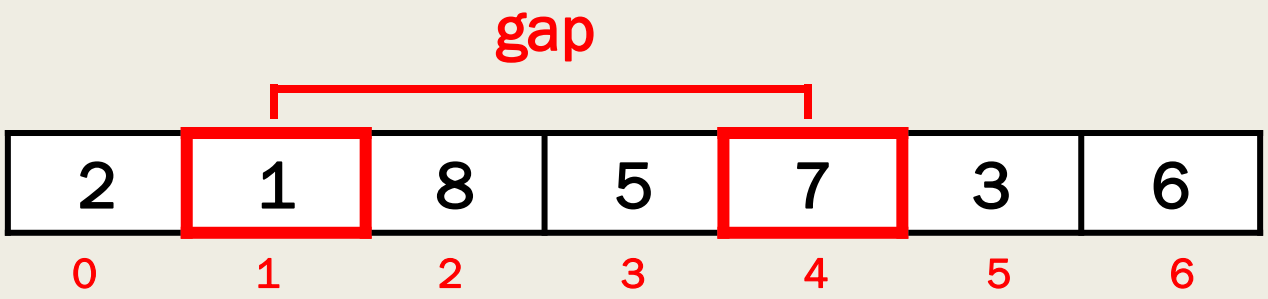
tam = 7

gap = 3



tam = 7

gap = 3

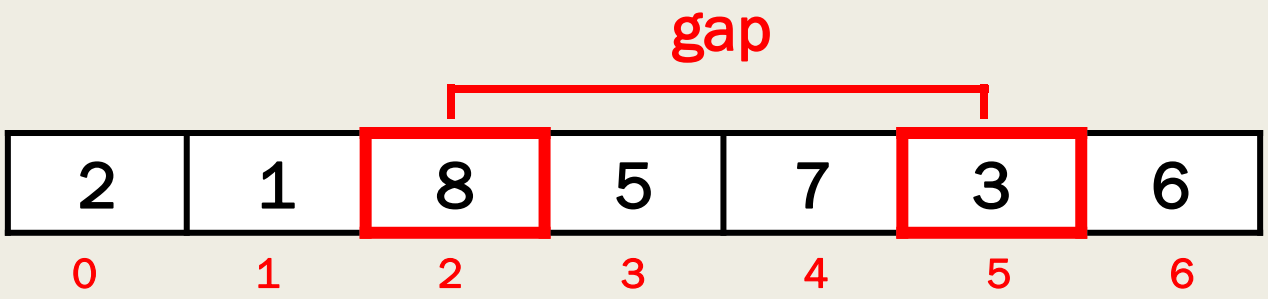


tam = 7

gap = 3

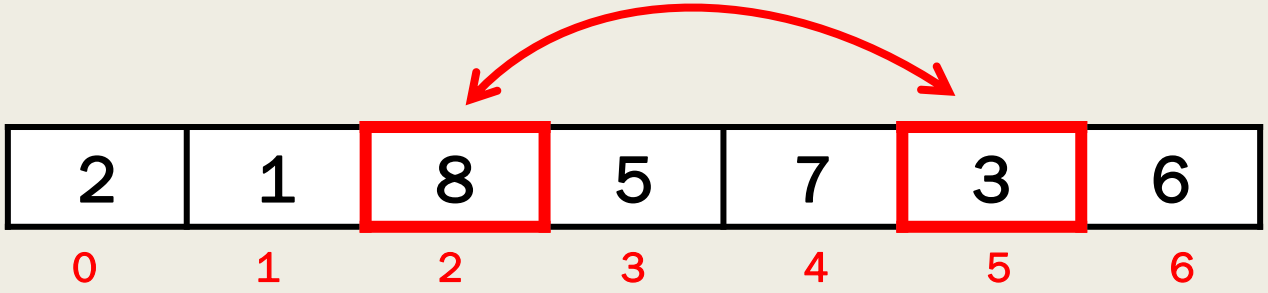
tam = 7

gap = 3



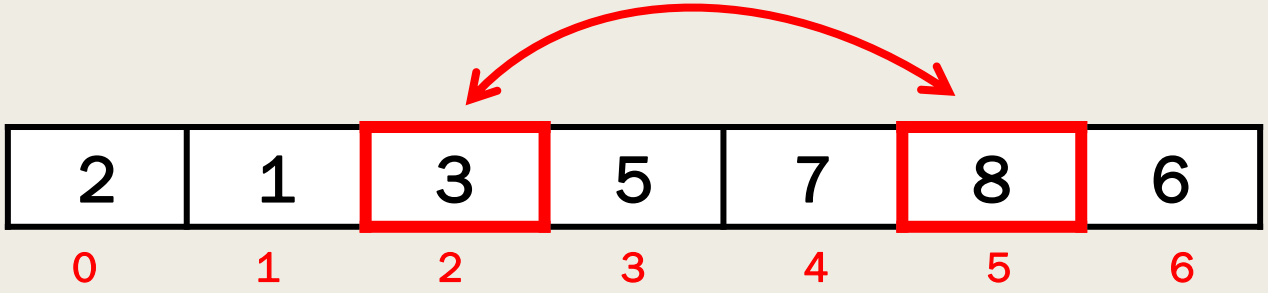
tam = 7

gap = 3



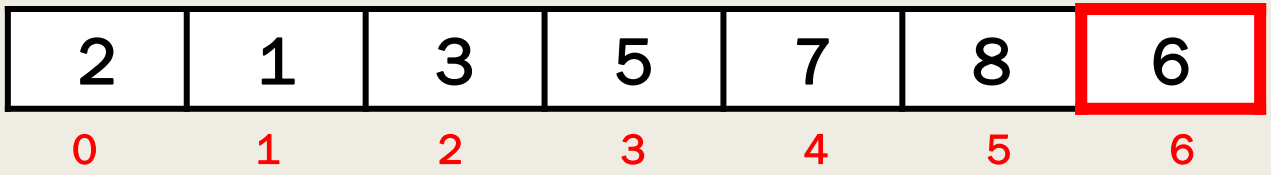
tam = 7

gap = 3



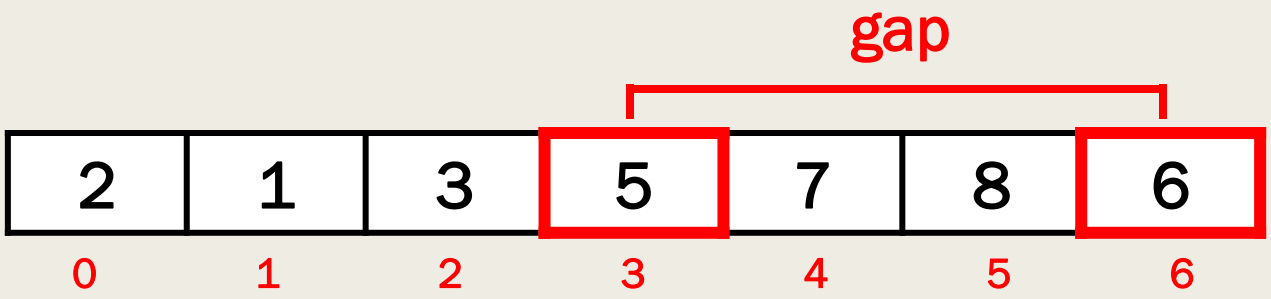
tam = 7

gap = 3



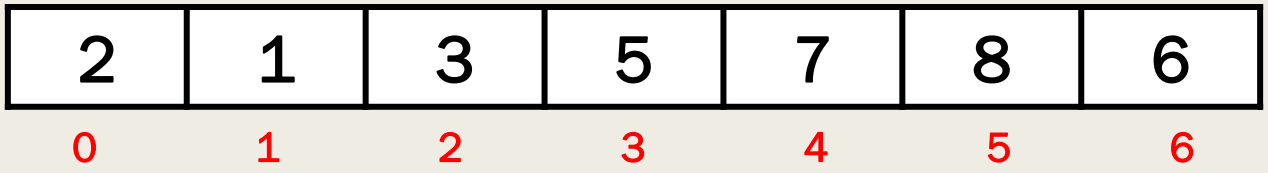
tam = 7

gap = 3



tam = 7

gap = 3

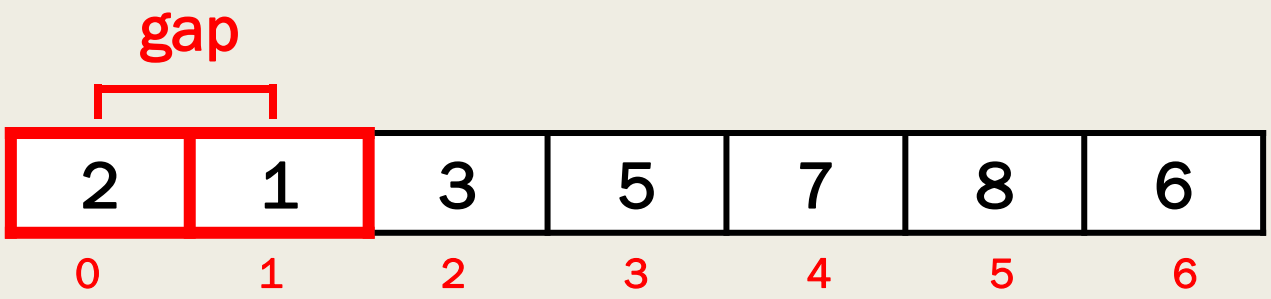


tam = 7

gap = 3 / 2 = 1

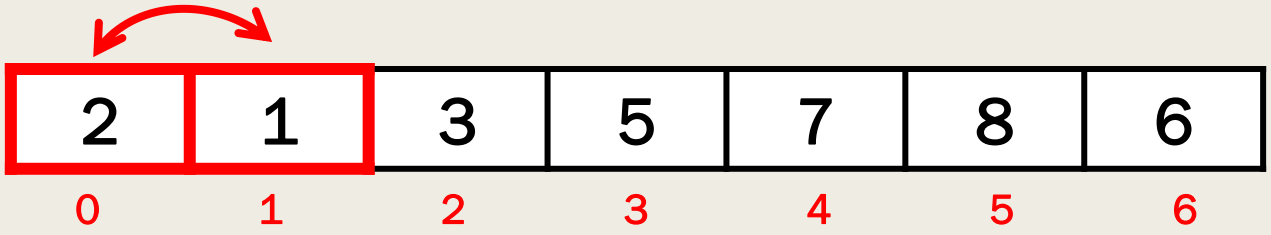
tam = 7

gap = 1



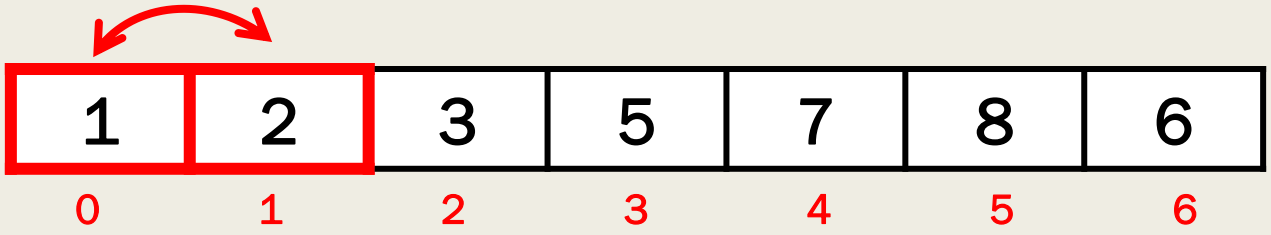
tam = 7

gap = 1



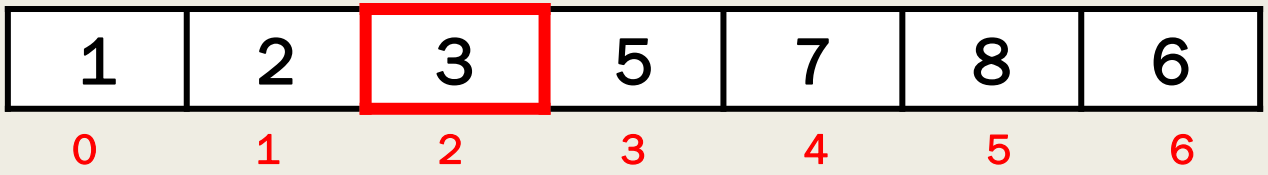
tam = 7

gap = 1



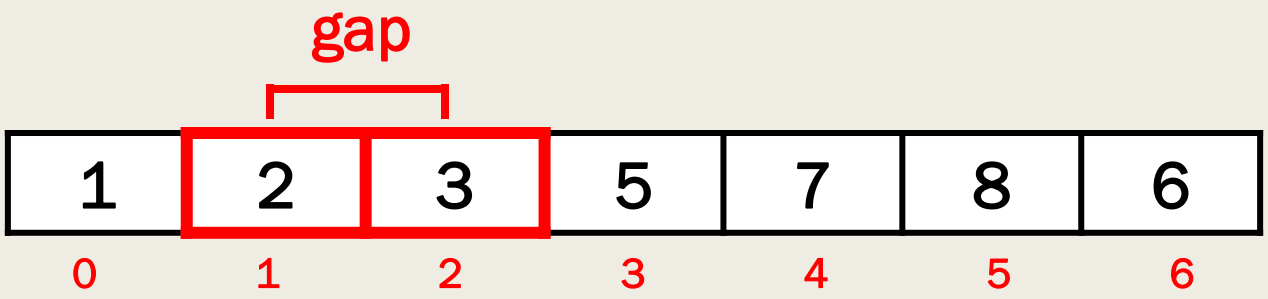
tam = 7

gap = 1



tam = 7

gap = 1

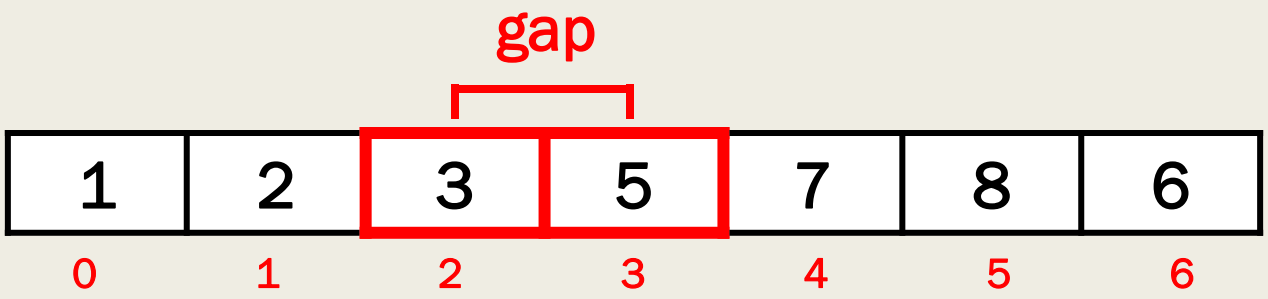


tam = 7

gap = 1

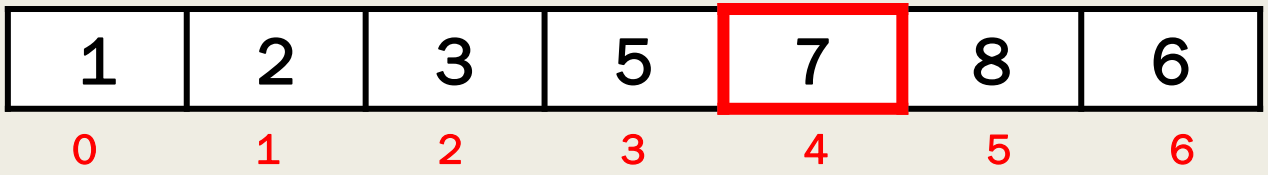
tam = 7

gap = 1



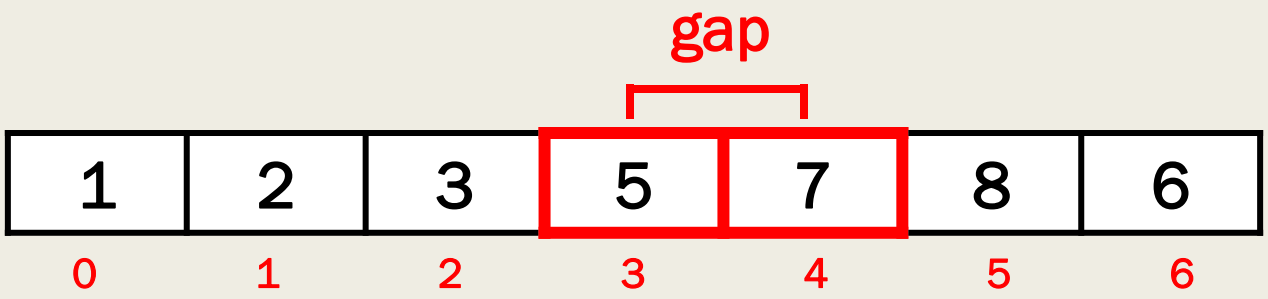
tam = 7

gap = 1



tam = 7

gap = 1

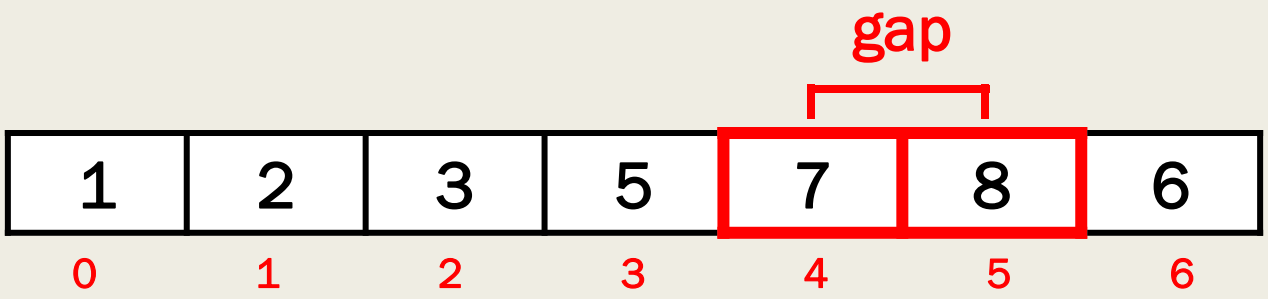


tam = 7

gap = 1

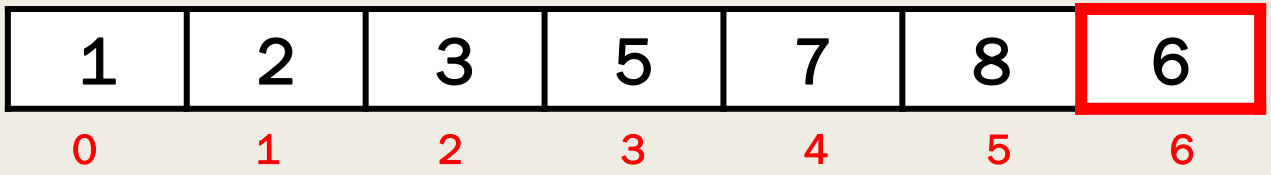
tam = 7

gap = 1



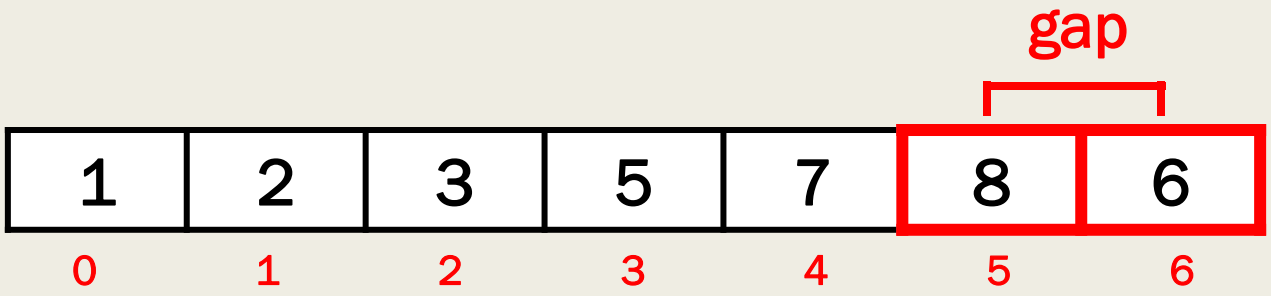
tam = 7

gap = 1



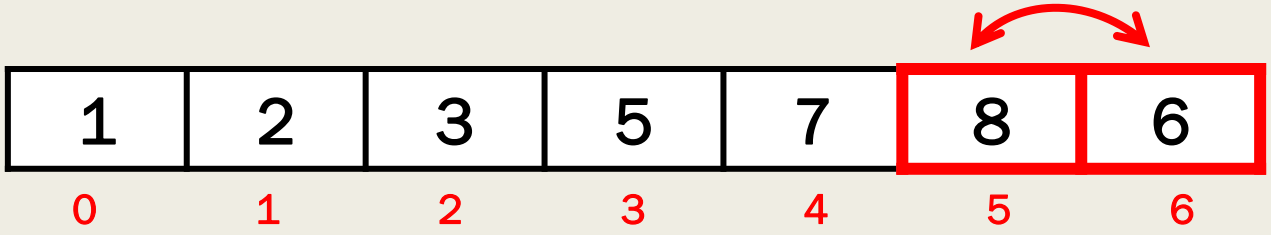
tam = 7

gap = 1



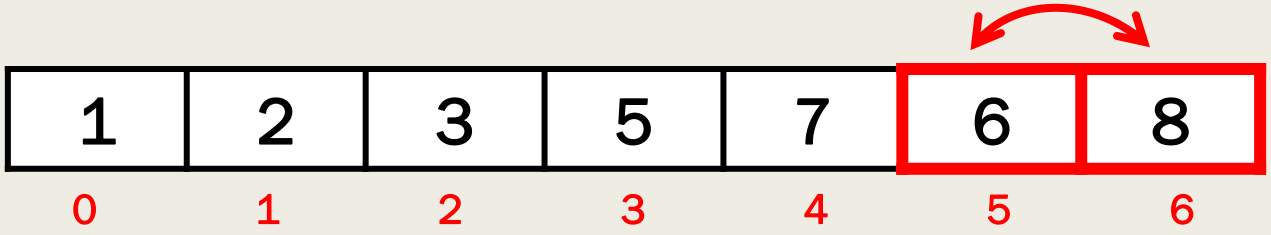
tam = 7

gap = 1



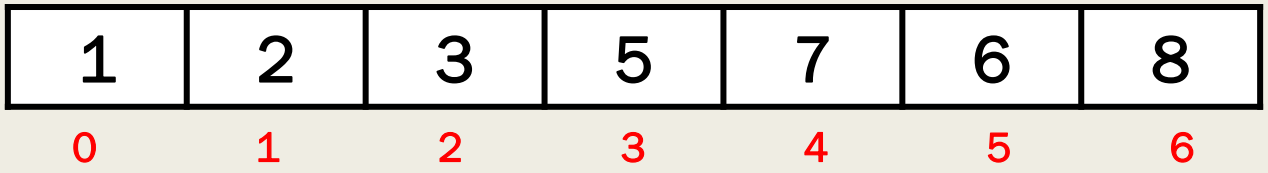
tam = 7

gap = 1



tam = 7

gap = 1



tam = 7

gap = 1

AVALIAÇÃO COM 100,000 NÚMEROS - C

	TESTE 01	TESTE 02	TESTE 03
CRESCENTE	0,015000	0.003000	0.016000
DECRESCENTE	0,000000	0.015000	0.000000
CRESCENTE DECRESCENTE	0,000000	0.000000	0.005000
DECRESCENTE CRESCENTE	0,016000	0.000000	0.000000
DESORDENADO	0,016000	0.015000	0.031000

AVALIAÇÃO COM 100,000 NÚMEROS - PYTHON

	TESTE 01	TESTE 02	TESTE 03
CRESCENTE	0,250004	0,249991	0,273056
DECRESCENTE	0,453116	0,467121	0,467121
CRESCENTE DECRESCENTE	0,431228	0,474673	0,453115
DECRESCENTE CRESCENTE	0,500004	0,453130	0,462381
DESORDENADO	1,281249	1,288572	1,262601

OBRIGADO!

To be continued

Referências Bibliográficas

- https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/priority.html
- https://en.wikipedia.org/wiki/Priority_queue#Using_a_priority_queue_to_sort
- https://en.wikipedia.org/wiki/Heap_(data_structure)
- https://en.wikipedia.org/wiki/Heapsort
- http://ptcomputador.com/P/computer-programming-languages/87235.html
- http://www.decom.ufop.br/anascimento/site_media/uploads/bcc202/aula_16_-_fila_de_prioridade_e_heapsort.pdf
- https://www.youtube.com/watch?v=MtQL_ll5KhQ
- https://www.cos.ufrj.br/~rfarias/cos121/aula_09.html

Referências Bibliográficas

- https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/shellsort.pdf
- https://pt.slideshare.net/ander-san/trabalho-shell-sort
- https://www.youtube.com/watch?v=SHcPqUe2GZM
- https://pt.wikipedia.org/wiki/Shell_sort
- https://en.wikipedia.org/wiki/Shellsort