

INSTITUTO FEDERAL
Sul-rio-grandense
Campus Gravataí

HEAPSORT

&

CYCLESORT

Adriano Ocampo, Artur Silva e João Victor Dreissig

ORDENAÇÃO: OTIMIZAÇÃO FUNDAMENTAL EM ALGORITMOS

- **Definição:** Transformação de um conjunto de dados desordenado no mesmo conjunto reestruturado para satisfazer uma relação de ordem específica.
- **Aplicação:** Predominantemente em estruturas de dados lineares (vetores, listas).
- **Importância:** É um problema fundamental, pois dados ordenados permitem: Buscas (ex: Busca Binária) e fusões drasticamente mais eficientes.
- **Crucial para:** Otimização de Sistemas de Informação, Bancos de Dados e Algoritmos de Processamento de Dados.

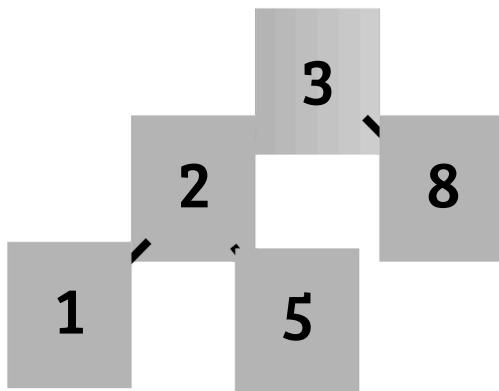
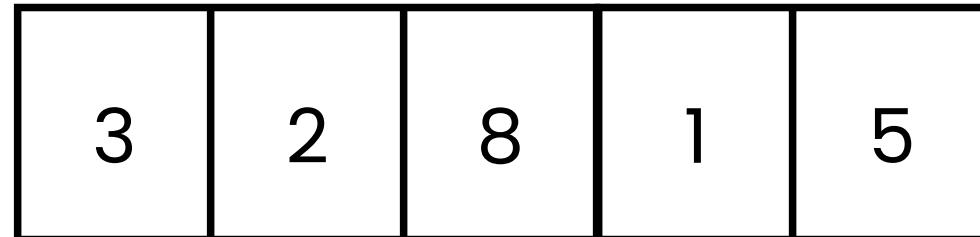
HEAPSORT

Algoritmo de Ordenação Eficiente

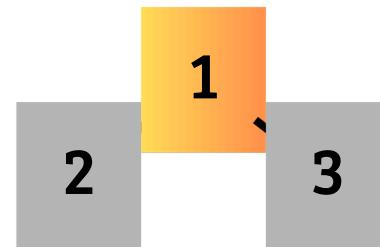
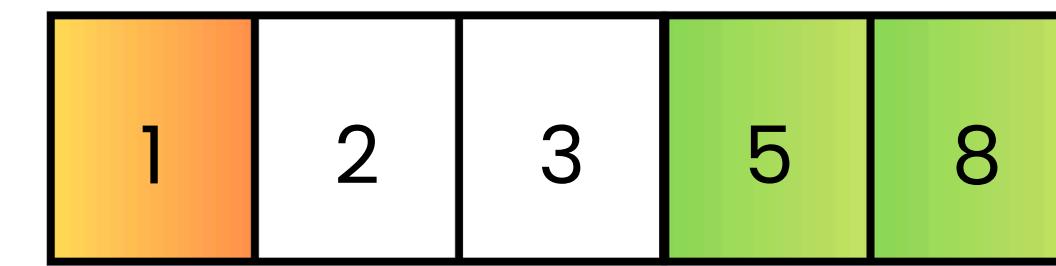
- **O que é:** Ordenação por comparação que usa uma heap binária;
- **Vantagem Principal:** Complexidade de tempo garantida de $O(n \log n)$ (melhor e pior caso) – Desempenho muito consistente;
- **Uso de Memória:** É in-place, ideal para recursos limitados;
- **Desvantagem:** É instável, ou seja, não preserva a ordem relativa de elementos com chaves iguais.

HEAPSORT

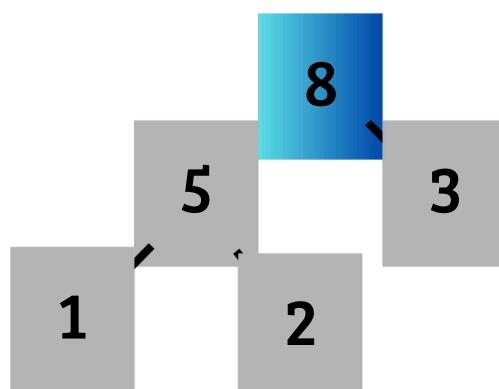
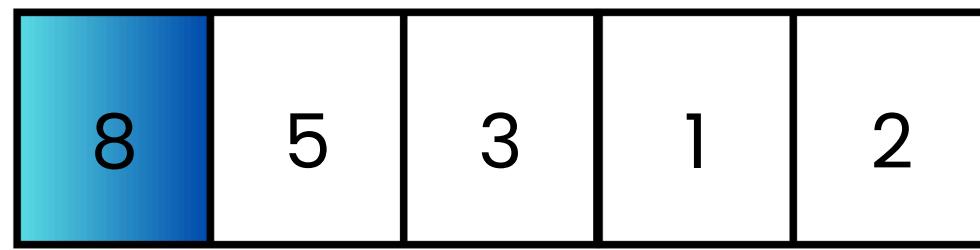
vctor



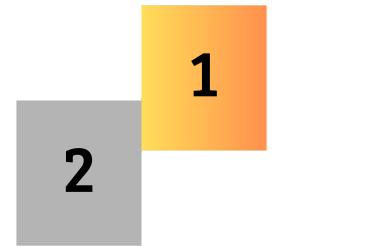
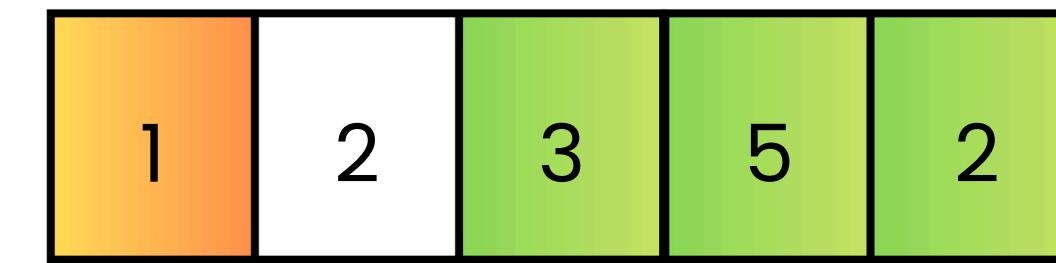
heapify



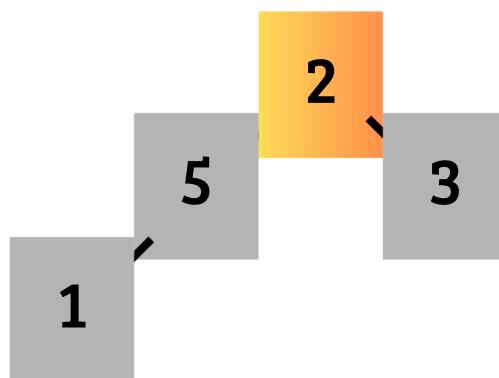
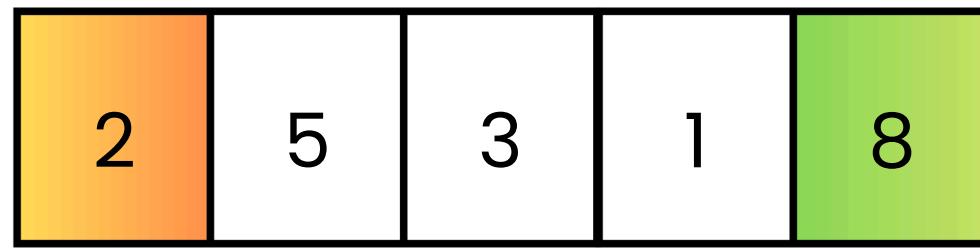
heapify



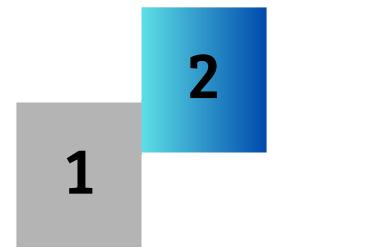
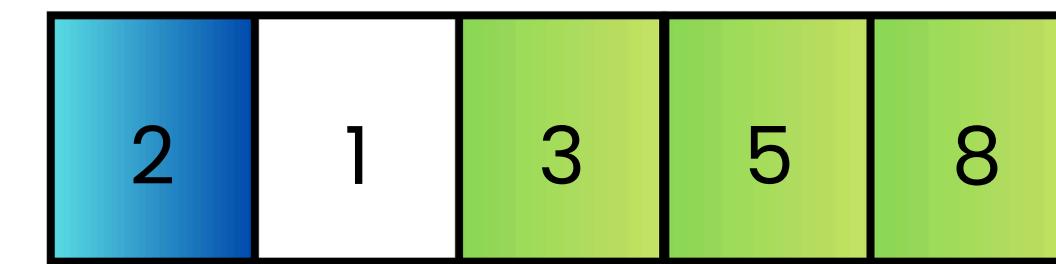
heapify



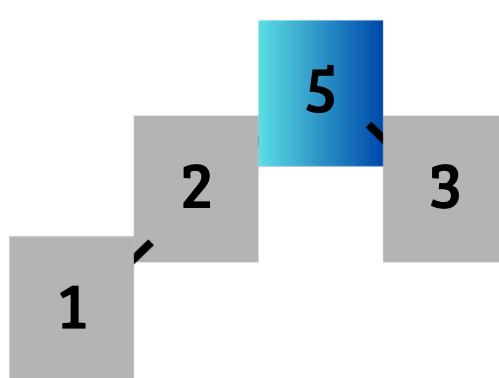
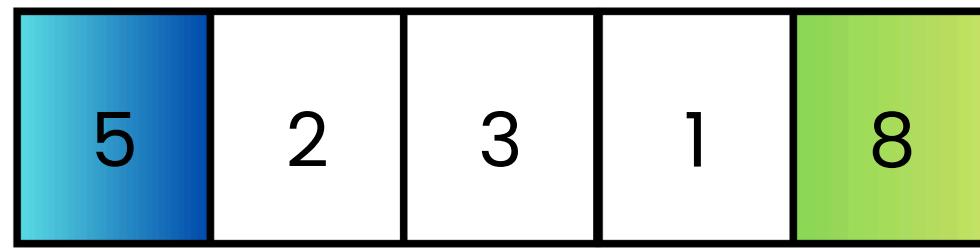
remove



remove



heapify



heapify

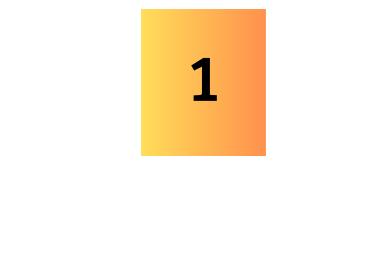
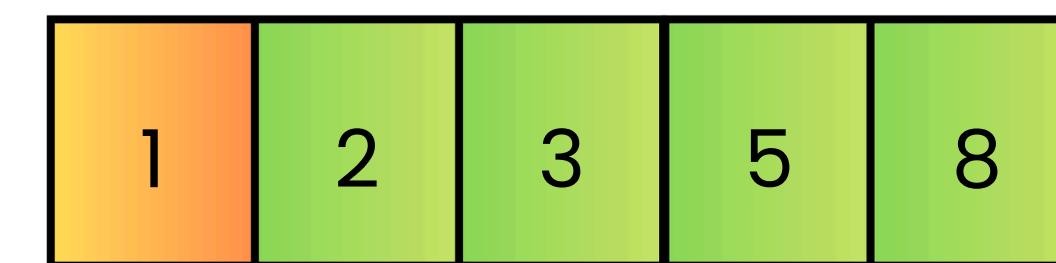


TABELA DE COMPARAÇÃO

Algoritmo	Complexidade (Pior Caso)	Uso de Memória (Extra)	Estabilidade
Heapsort	$O(n \log n)$	$O(1)$ (In-place)	Não (Instável)
Quick Sort	$O(n^2)$	$O(\log n)$ (Recursão)	Não (Instável)
Merge Sort	$O(n \log n)$	$O(n)$	Sim (Estável)

RESULTADO DOS TESTES

HeapSort

Vetor de 1.000

20 execuções

Vetor aleatório

média: **0.000000seg.**

desvio padrão: **0.000000 seg.**

Vetor crescente

média: **0.000200seg.**

desvio padrão: **0.000410 seg.**

Vetor decrescente

média: **0.000000seg.**

desvio padrão: **0.000000 seg.**

HeapSort

Vetor de 100.000

20 execuções

Vetor aleatório

média: **0.026050seg.**

desvio padrão: **0.007451 seg.**

Vetor crescente

média: **0.020250 seg.**

desvio padrão: **0.005098 seg.**

Vetor decrescente

média: **0.019200 seg.**

desvio padrão: **0.006396 s**

HeapSort

Vetor de 1.000.000

20 execuções

Vetor aleatório

média: **0.346500seg.**

desvio padrão: **0.018648 seg.**

Vetor crescente

média: **0.225050seg.**

desvio padrão: **0.007437 seg.**

Vetor decrescente

média: **0.226550seg.**

desvio padrão: **0.008217 seg.**

AMBIENTE DE TESTE



```
testeADRIANO_ARTUR_JOAO_SORT.cpp
1 #include <stdio.h> //para usar funções de imprimir e ler
2 #include <stdlib.h> //para malloc, rand, exit
3 #include <math.h> //para calcular desvio padrão com sqrt
4 #include <time.h> //para srand
5
6 //const para definir número de repetições
7 //#define NUM_EXECUÇÃO 20
8
9 //função cria vetor aleatório
10 int* criaVetorAleatório(int tamanho) {
11     int* vetor = (int*)malloc(tamanho * sizeof(int));
12     if(vetor == NULL) {
13         printf("Erro: não foi possível alocar memória.");
14         exit(1);
15     }
16     //preenche vetor com valores aleatórios
17     for(int i = 0; i < tamanho; i++) {
18         vetor[i] = rand() % 10000;
19     }
20     //retorna o ponteiro para o vetor
21     return vetor;
22 }
23
24 //função criar vetor crescente
25 int* criaVetorCrescente(int tamanho) {
26     int* vetor = (int*)malloc(tamanho * sizeof(int));
27     if(vetor == NULL) {
28         printf("Erro: não foi possível alocar memória.");
29         exit(1);
30     }
31     //verifica se o vetor é nulo
32     if(vetor == NULL) {
33         printf("Erro: não foi possível alocar memória.");
34         exit(1);
35     }
36     //preenche vetor ordem crescente
37     for(int i = 0; i < tamanho; i++) {
38         vetor[i] = i;
39     }
40     //retorna o ponteiro para o vetor
41     return vetor;
42 }
```

C:\Users\adriano\Documents\ testeADRIANO_ARTUR_JOAO_SORT.cpp

Processos

Nome	Status	32% CPU	60% Memória	1% Disco	0% Rede
Google Chrome (26)	Ativo	0%	2.045 MB	0 MB/s	0,1 Mbps
Host for Endpoint Security	Ativo	0%	413,4 MB	0 MB/s	0 Mbps
Pesquisar (7)	Ativo	0%	92,4 MB	0 MB/s	0 Mbps
Windows Explorer	Ativo	0%	66,7 MB	0 MB/s	0 Mbps
Gerenciador de Tarefas	Ativo	7,1%	62,8 MB	0 MB/s	0 Mbps
Secure System	Ativo	0%	53,5 MB	0 MB/s	0 Mbps
Agente Milius Helpdesk	Ativo	0%	53,2 MB	0,1 MB/s	0 Mbps
Gerenciador de Janelas da Área...	Ativo	0%	47,3 MB	0 MB/s	0 Mbps
WMI Provider Host	Ativo	0%	45,8 MB	0 MB/s	0 Mbps
Configurações	Ativo	0%	44,7 MB	0 MB/s	0 Mbps
Iniciar	Ativo	0%	33,3 MB	0 MB/s	0 Mbps
Windows Terminal Host	Ativo	0%	26,1 MB	0 MB/s	0 Mbps
SnippingTool.exe	Ativo	0%	24,2 MB	0 MB/s	0 Mbps
Microsoft Edge (5)	Ativo	0%	20,6 MB	0 MB/s	0 Mbps
appmodel	Ativo	0%	19,9 MB	0 MB/s	0 Mbps

Desempenho

CPU

Utilização: 45% 3,60 GHz

Velocidade base: 3,60 GHz

Sockets: 1

Núcleos: 4

Processadores lógicos: 4

Virtualização: Habilitado

Cache L1: 256 KB

Cache L2: 1,0 MB

Cache L3: 6,0 MB

Tempo de atividade: 7:11:26:15

Nome	Status	73% CPU	56% Memória	2% Disco	0% Rede
Microsoft Edge (5)	Ativo	0%	21,9 MB	0 MB/s	0 Mbps
Windows Terminal Host	Ativo	10,5%	20,9 MB	0 MB/s	0 Mbps

CYCLESORT

Otimizado para Mínimas Escritas

- **O que é:** Ordenação baseada em comparação que funciona ao identificar e rotacionar ciclos de elementos fora de posição;
- **Vantagem Principal:** É o algoritmo teoricamente ótimo em termos de número de escritas na memória (swaps), garantindo que cada elemento seja escrito em sua posição final no máximo uma vez;
- **Uso de Memória:** É in-place $O(1)$, ideal para sistemas com recursos limitados;
- **Desvantagem Crucial:** Sua complexidade de tempo no pior caso é $O(n^2)$ – o desempenho é lento e inconsistente para grandes conjuntos de dados.

CYCLESORT

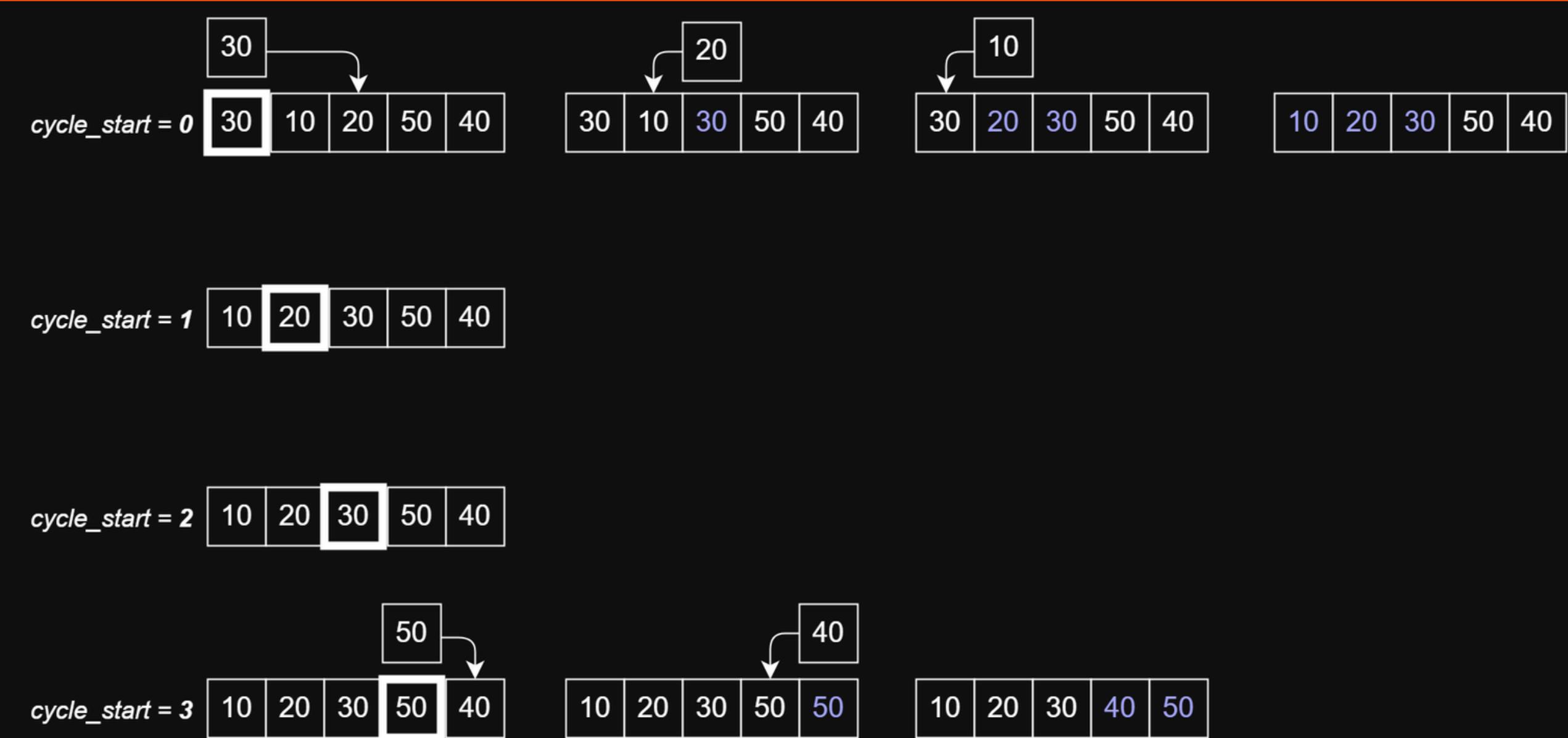


TABELA DE COMPARAÇÃO

Algoritmo	Pior Caso (Tempo)	Uso de Memória (Extra)	Estabilidade	Swaps (Escritas)
Cycle Sort	$O(n^2)$	$O(1)$ (In-place)	Não (Instável)	Ótimo (Mínimo)
Heapsort	$O(n \log n)$	$O(1)$ (In-place)	Não (Instável)	Alto

RESULTADO DOS TESTES

CycleSort		
Vetor de 1.000		
20 execuções		
Vetor aleatório	Vetor crescente	Vetor decrescente
média: 0.004700 seg.	média: 0.001550 seg.	média: 0.001450 seg.
desvio padrão: 0.007385 seg.	desvio padrão: 0.004774 seg.	desvio padrão: 0.004489 seg.

CycleSort		
Vetor de 100.000		
20 execuções		
Vetor aleatório	Vetor crescente	Vetor decrescente
média: 47.443900 seg.	média: 11.900800 seg.	média: 18.077900 seg.
desvio padrão: 1.010258 seg.	desvio padrão: 0.885235 seg.	desvio padrão: 0.246267 seg.

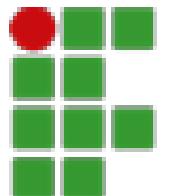
CycleSort		
Vetor de 1.000.000		
nenhuma execução foi concluída		
Vetor aleatório	Vetor crescente	Vetor decrescente
Não rodou o programa	Não rodou o programa	Não rodou o programa

MÉDIA EM SEGUNDOS
1 millhão (1194.412 segundos / 19.9 minutos

CycleSort	CycleSort	CycleSort
Vetor de 1.000	Vetor de 1.000	Vetor de 1.000
20 execuções	20 execuções	20 execuções
Vetor Crescente	Vetor Decrescente	Vetor Aleatório
Desvio Padrão: 0.00000 s	Desvio Padrão: 0.00327 s	Desvio Padrão: 0.00883 s
Média: 0.00000 s	Média: 0.00075 s	Média: 4.13480 s
CycleSort	CycleSort	CycleSort
Vetor de 100.000	Vetor de 100.000	Vetor de 100.000
20 execuções	20 execuções	20 execuções
Vetor Crescente	Vetor Decrescente	Vetor Aleatório
Desvio Padrão: 0.00883 s	Rodou apenas até o exemplo 11, com o tempo médio de 7.51155 s	Simplesmente não rodou.
Média: 4.13480 s		
CycleSort	CycleSort	CycleSort
Vetor de 1.000.000	Vetor de 1.000.000	Vetor de 1.000.000
20 execuções	20 execuções	20 execuções
Vetor Crescente	Vetor Descrescente	Vetor Aleatório
Simplesmente não rodou.	Simplesmente não rodou.	Simplesmente não rodou.

CycleSort		
Vetor de 500.000 e 700.000		
500.000		700.000
Vetor crescente		Vetor crescente
média: 260.246090 seg. (10 execuções, sem desvio padrão)		1 execução: 552.790000 seg.

MUITO
Obrigado



INSTITUTO FEDERAL
Sul-rio-grandense
Campus Gravataí