Capítulo 2 - A Sinfonia Inacabada da Ordenação

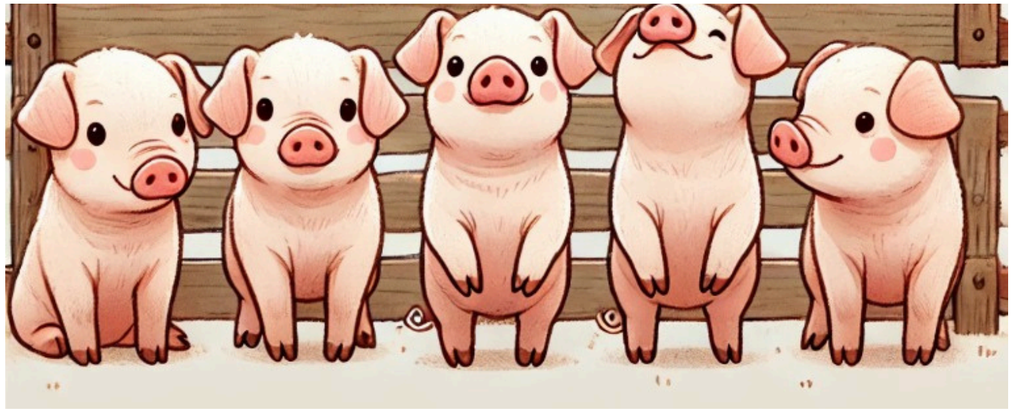
Objetivo:

1. Implementar os algoritmos MergeSort e QuickSort em java.
2. Apresentar os algoritmos de forma criativa e didática para a turma.
3. Realizar a análise de contagem de tempo e operações dos algoritmos BubbleSort e SelectionSort.
4. Gerar gráficos comparativos de desempenho para os algoritmos BubbleSort, InsertionSort e SelectionSort.
5. Ordenar a mensagem encontrada no dispositivo apreendido no museu.

Implementação

Apresentação dos algoritmos

Olá turma! temos 10 porquinhos, com diferentes alturas, esses porquinhos querem se organizar do menor ao maior. 5 desses porquinhos ficaram responsáveis por implementar essa ordenação de altura que eles queriam fazer, acabou que ficou organizado assim:



O primeiro porco responsável pela ordenação era um cara muito impaciente e não se importava com o tempo que isso ia demorar, ele então resolveu fazer uma fila e pediu para que, em ordem, cada porco se comparasse com o que está do seu lado, se ocorresse de um porco maior estar antes do menor na fila, eles trocariam de lugar. Resumidamente, ficaram um século tentando se organizar até finalmente funcionar e o porquinho foi brutalmente agredido pela ideia sugerida, muito demorada.

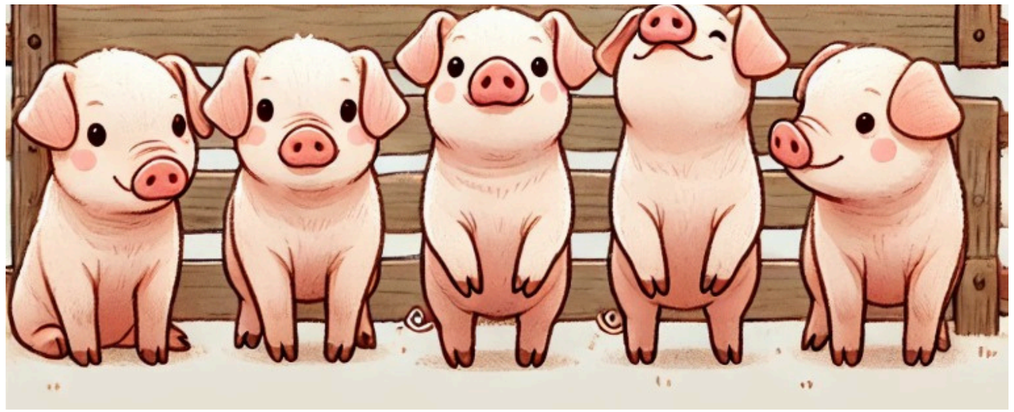
A) BubbleSort

B) SelectionSort

C)InserionSort

D) MergeSort

E)QuickSort



O segundo porco era um cara mais perfeccionista, ele pensou em organizar uma fila, procurar o menor dos porcos e pôr ele no início da fila, fazendo isso repetidas vezes até que estivessem todos em ordem. Funcionou, porém, foi, assim como o primeiro, brutalmente xingado por sugerir uma ideia que necessitava de ficar transitando pela fila um milhão de vezes até encontrar o menor e colocar no início da fila. Resumidamente, pouco prática, muito demorada mas, melhor que a sugestão do primeiro porquinho.

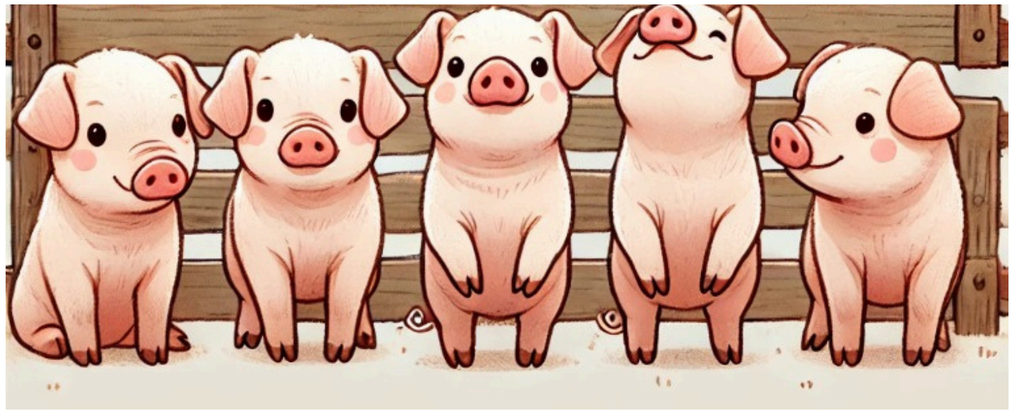
A) BubbleSort

B) SelectionSort

C)InserionSort

D) MergeSort

E)QuickSort



O terceiro Porquinho já tinha um pouco mais de organização, montou a fila um por um, cada porquinho entrava no seu lugarzinho da fila, empurrando os outros caso necessário. Acabou que demorou um pouquinho menos do que as soluções propostas pelos últimos 2 porquinhos, ainda foi necessário cada um se comparar com os outros que já estavam na fila mas, no geral, foi bem rápido e prático, o porco não foi agredido fisicamente pela sua solução proposta.

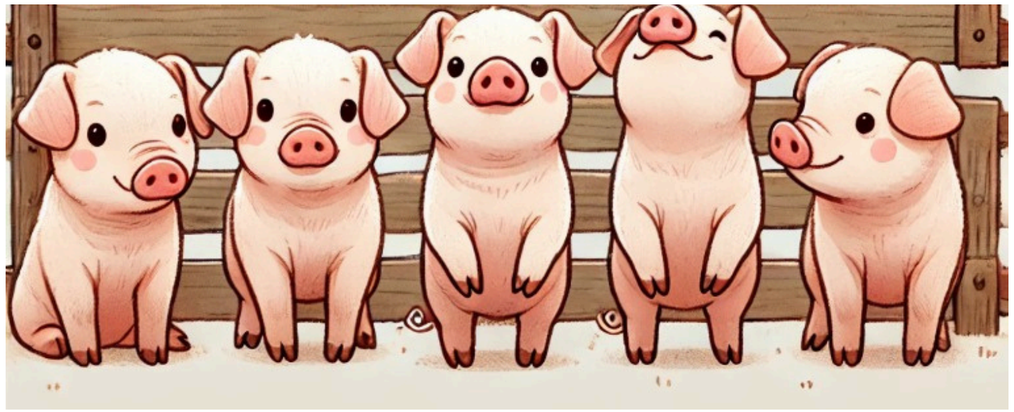
A) BubbleSort

B) SelectionSort

C)InserionSort

D) MergeSort

E)QuickSort



O 4° Porquinho, era um cara estrategista e disciplinado, dividiu os porquinhos em dois grupos, cada grupo se ordenou de forma independente, depois se juntaram de maneira ordenada. O 4° porquinho recebeu palmas dos demais pela sua proposta, mesmo se fosse com o dobro da quantidade de porcos para ordenar, funcionaria bem do mesmo jeito.

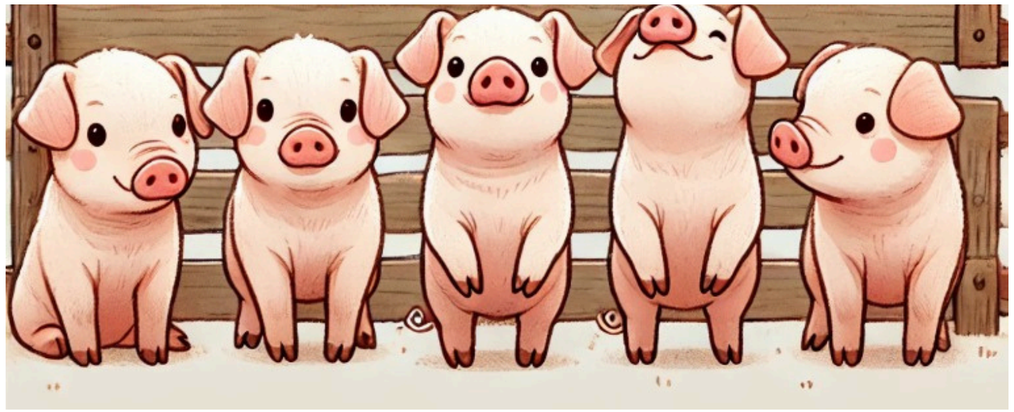
A) BubbleSort

B) SelectionSort

C)InserionSort

D) MergeSort

E)QuickSort



o 5° e último porco escolhido era um cara metido a líder, gostava de mandar nos outros, ele chegou no grupo e disse assim: "Quem for mais baixo que eu vai pra aquele lado, quem for mais alto, vai para o outro lado". Assim, cada subgrupo realizou o mesmo procedimento até todos estarem ordenados. O lado ruim, é que o porco metido a líder era o mais baixo de todos, sendo assim, acabou virando a mesma estratégia do porco 1 disfarçada, o 5° porquinho foi assassinado naquele dia.

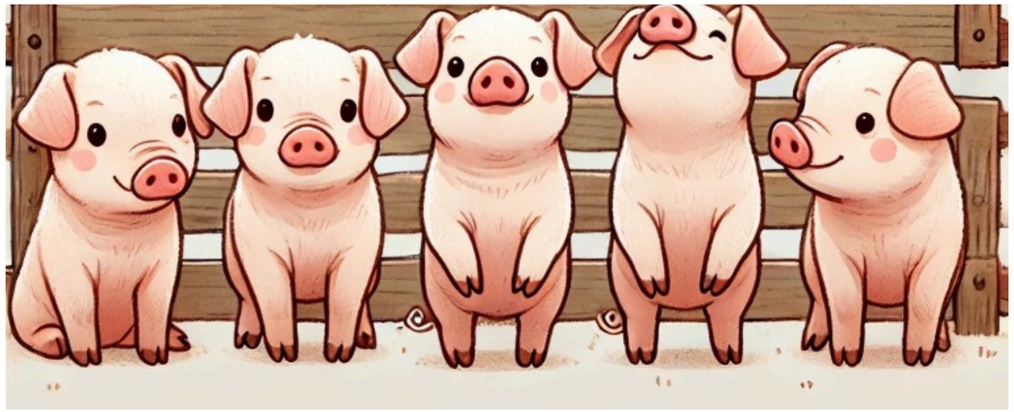
A) BubbleSort

B) SelectionSort

C)InserionSort

D) MergeSort

E)QuickSort



Análise de contagem do tempo

BubbleSort

```
array1:
Array ordenado:
658 2836 9787 26532
Qtd de operacoes: 12
Duração: 0,000002700 segundos

array2:
Array ordenado:
80 68653 265808 567657 932858 4737673 9069327
Qtd de operacoes: 42
Duração: 0,000002700 segundos

array3:
Array ordenado:
2653 279866 66532676 583653268 693268657665
Qtd de operacoes: 31
Duração: 0,000001600 segundos

80 658 2653 2836 9787 26532 68653 265808 279866
 567657 932858 4737673 9069327 66532676 5836532
68 693268657665
Qtd de operacoes: 219
Duração: 0,000005200 segundos
```

Por que o tempo é tão curto?

O BubbleSort consegue executar de forma bem mais rápida quando aplicado a arrays pequenos.

SelectionSort

```
array1:
658 2836 9787 26532
Qtd de operacoes: 15
Duração: 0,000003300 segundos

array2:
80 68653 265808 567657 932858 4737673 9069327
Qtd de operacoes: 39
Duração: 0,000002900 segundos

array3:
2653 279866 66532676 583653268 693268657665
Qtd de operacoes: 22
Duração: 0,000001200 segundos

array4:
80 658 2653 2836 9787 26532 68653 265808 279866 567657 932858 4737673 9069
327 66532676 583653268 693268657665
Qtd de operacoes: 165
Duração: 0,000003400 segundos
```

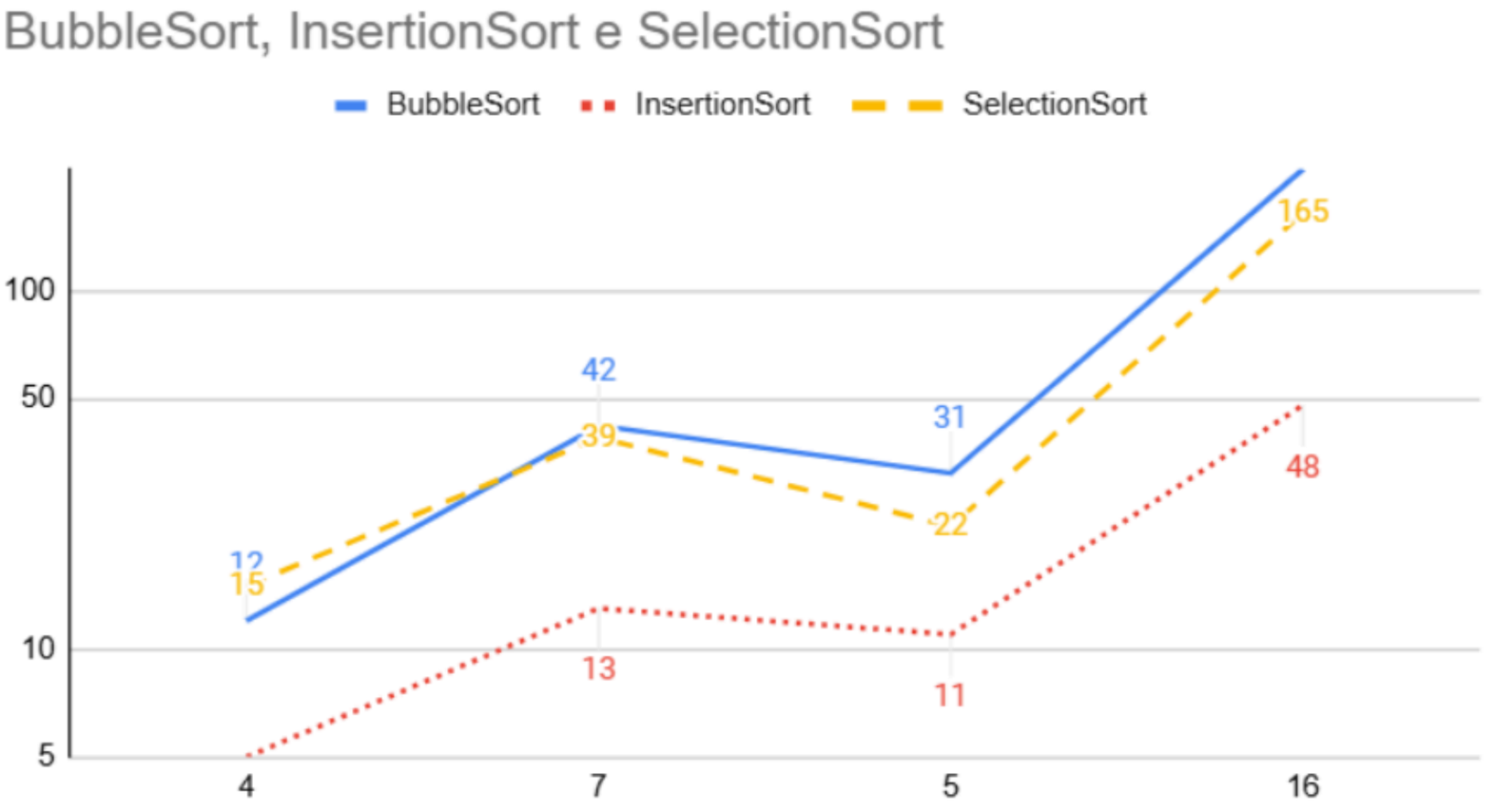
o algoritmo funciona bem para arrays pequenos, mas não é eficiente para grandes quantidades de dados, pois o tempo de execução cresce rapidamente conforme o tamanho do array aumenta.

Análise da contagem de Operações

Critério	Bubble Sort	Selection Sort
Complexidade de tempo (melhor caso)	O(n) (quando já está ordenado)	$O(n^2)$
Complexidade de tempo (pior caso)	$O(n^2)$ (quando a lista está invertida)	$O(n^2)$
Complexidade de tempo (caso médio)	$O(n^2)$	$O(n^2)$
Trocas	$O(n^2)$	O(n)
Comparações	$O(n^2)$	$O(n^2)$

Algoritmo	Melhor Caso	Caso Médio	Pior Caso	Trocas
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	Muitas
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Poucas

Gráficos comparativos de desempenho para Bubble, Insertion e SelectionSort



Tamanho do array (n)	BubbleSort	InsertionSort	SelectionSort
4	12	5	15
7	42	13	39
5	31	11	22
16	219	48	165

Mensagem Final encontrada

80 = P
65 = A
82 = R
65 = A
32 = ""
83 = S
69 = E
78 = N
72 = H
65 = A
32 = ""
68 = D
65 = A
32 = ""

65 = A
80 = P
82 = R
79 = O
86 = V
65 = A
67 = C
65 = A
79 = O
32 = ""

85 = U
84 = T
73 = I
76 = L
73 = I
90 = Z
69 = E
32 = ""

76 = L
65 = A
32 = “ “
67 = C
65 = A
83 = S
65 = A
32 = “ “
68 = D
69 = E
32 = “ “
68 = D
65 = A
76 = L
65 = A