Faculdade Metropolitana de Guaramirim - Grupo UNIASSELVI
Curso de análise e desenvolvimento de sistemas
Trabalho acadêmico

MÉTODOS DE ORDENAÇÃO

Fábio D. C. Depin
Doglas Vegini
Jeferson Buzzi
Cloves Perreira De Jesus

Guaramirim, 12 de novembro de 2012.

INTRODUÇÃO

Em vários momentos do dia a dia, o homem depara-se com a necessidade de consultar dados ordenados.
- Como exemplo, pode-se citar uma lista telefônica. Imagine como seria consultar o telefone de uma pessoa se os nomes não estivessem classificados em ordem alfabética.

- Por isso uma das atividades mais utilizada na computação é a ordenação.
As ordens mais utilizadas são as numéricas e as lexicográficas.

OBJETIVO

O presente trabalho tem o objetivo de realizar a análise de desempenho de cinco métodos de ordenação, lembrando que além destes ainda existem muitos outros.

Os métodos a serem avaliados são:
- Bubble Sort
- Shell Sort
- Quick Sort
- Heap Sort
- Insertion Sort

DESENVOLVIMENTO

As seguintes ferramentas foram utilizadas para implementação dos algoritmos de ordenação e testes.

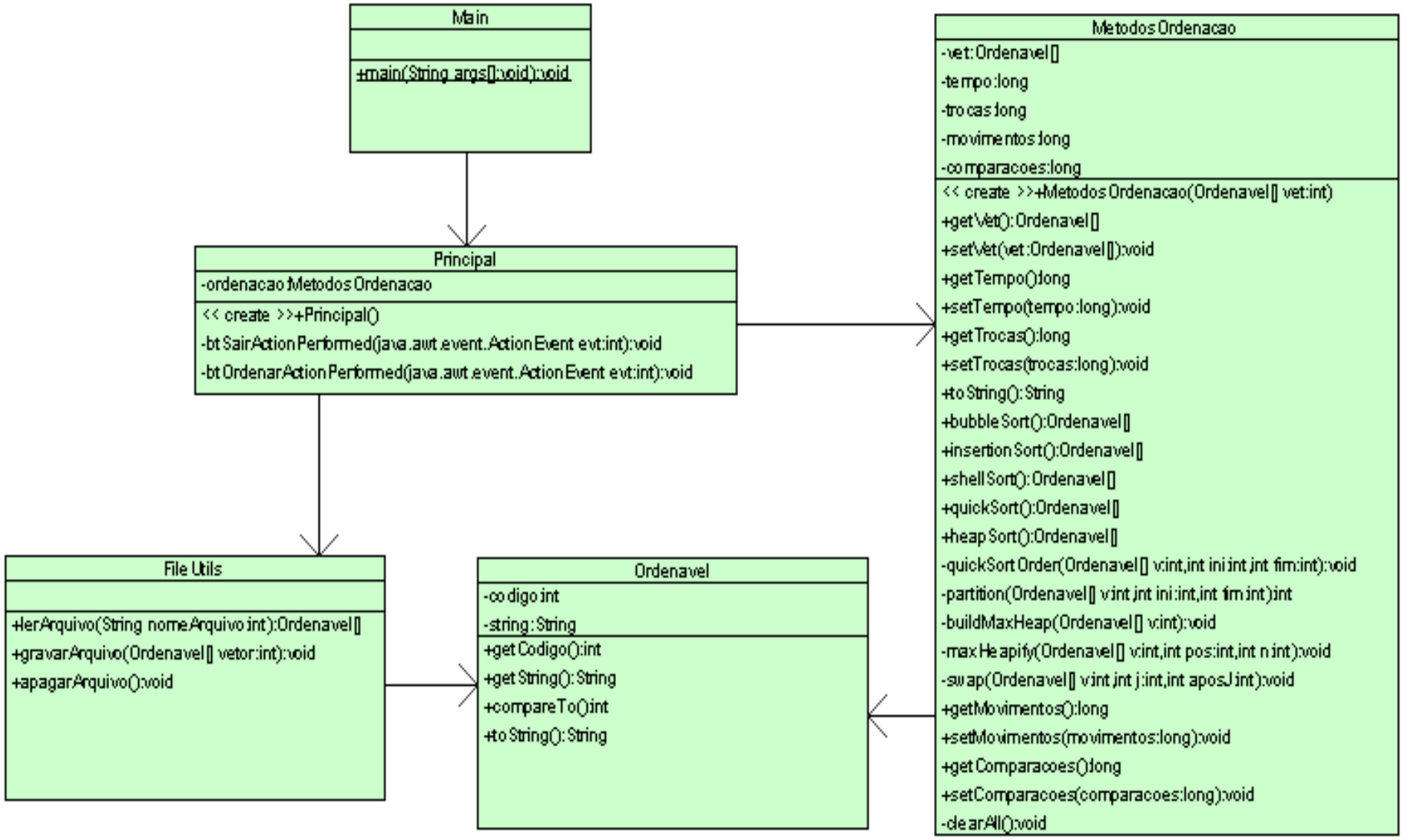
TESTES

Para realização dos testes e análises de desempenho, cada algoritmo terá que ordenar um vetor de objetos ordenáveis por um campo de código com número inteiro.
Os vetores terão 10.000 posições e serão de três tipos (ordenados em ordem crescente e decrescente e aleatório).
- Como medidas para a comparação entre os métodos foi escolhido durante cada teste:
1. Número de comparações entre chaves do vetor;
2. Número de movimentações;
3. Contagem de tempo gasto durante a execução do algoritmo;

IMPLEMENTAÇÃO DOS MÉTODOS

Os métodos foram implementados em uma única classe MetodosOrdenacao que é responsável pela ordenação dos vetores e pela medição do tempo, número de comparações e número de movimentações.

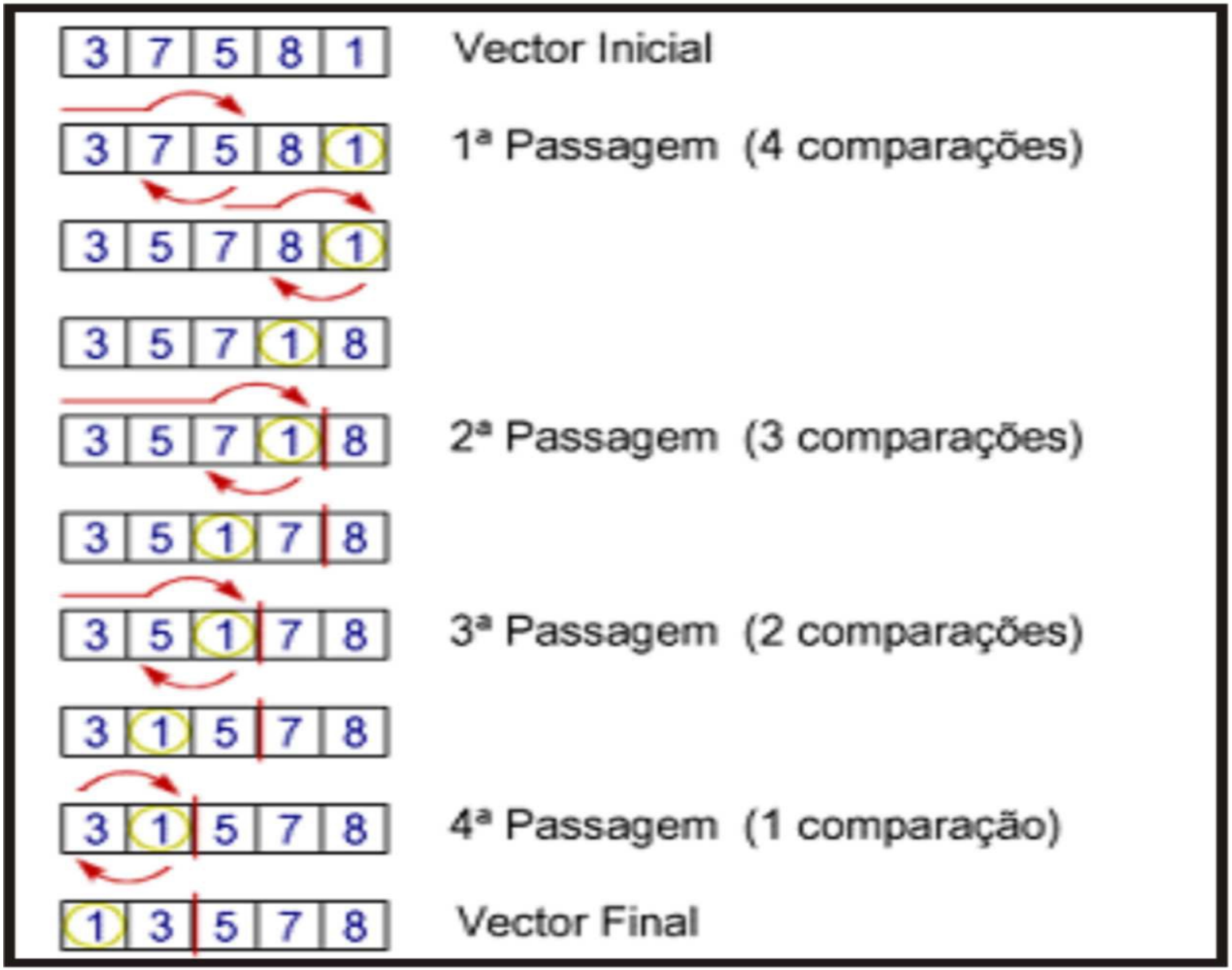
DIAGRAMA DE CLASSE



BUBLESORT

É um dos algoritmos de ordenação mais simples.
O algoritmo consiste de percorrer os N elementos de um vetor, para cada vez percorrida, todos os elementos são comparados com o seu próximo, para verificar se estão na ordem desejada.

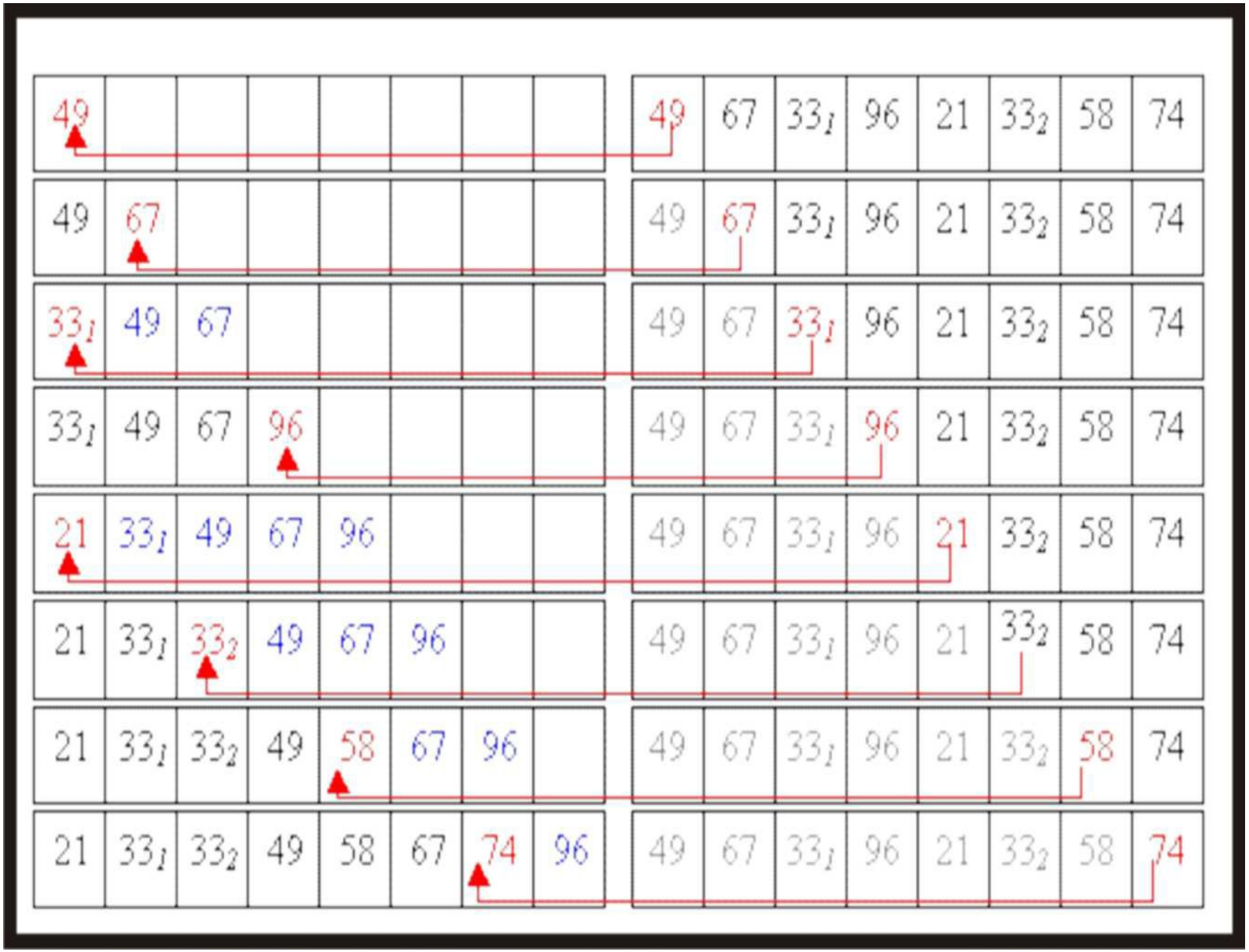
BUBLESORT


3 7 5 8 1
Vector Inicial
3 7 5 8 1
1ª Passagem (4 comparações)
3 5 7 8 1
3 5 7 1 8
3 5 7 1 8
2ª Passagem (3 comparações)
3 5 1 7 8
3 5 1 7 8
3ª Passagem (2 comparações)
3 1 5 7 8
3 1 5 7 8
4ª Passagem (1 comparação)
1 3 5 7 8
Vector Final

INSERTIONSORT

É eficiente quando aplicado à um vetor com poucos elementos.
- Em cada passo, a partir de i=2 o *i*-ésimo item da sequência é apanhado e transferido para a sequência destino, sendo inserido no seu lugar apropriado. O algoritmo assemelha-se com a maneira que um jogador de cartas ordena as cartas na mão em um jogo.

INSERTIONSORT



SHELLSORT

É derivado do método InsertionSort.

Funcionamento:

1. Ordenação é efetuada dividindo o conjunto em subconjuntos.
2. Os subconjuntos são constituídos por elementos separados n elementos (incrementos).
3. Os subconjuntos são ordenados utilizando o método de inserção.
4. Repetição dos pontos 1, 2 e 3, com diminuição do incremento, até o conjunto estar ordenado.

SHELLSORT

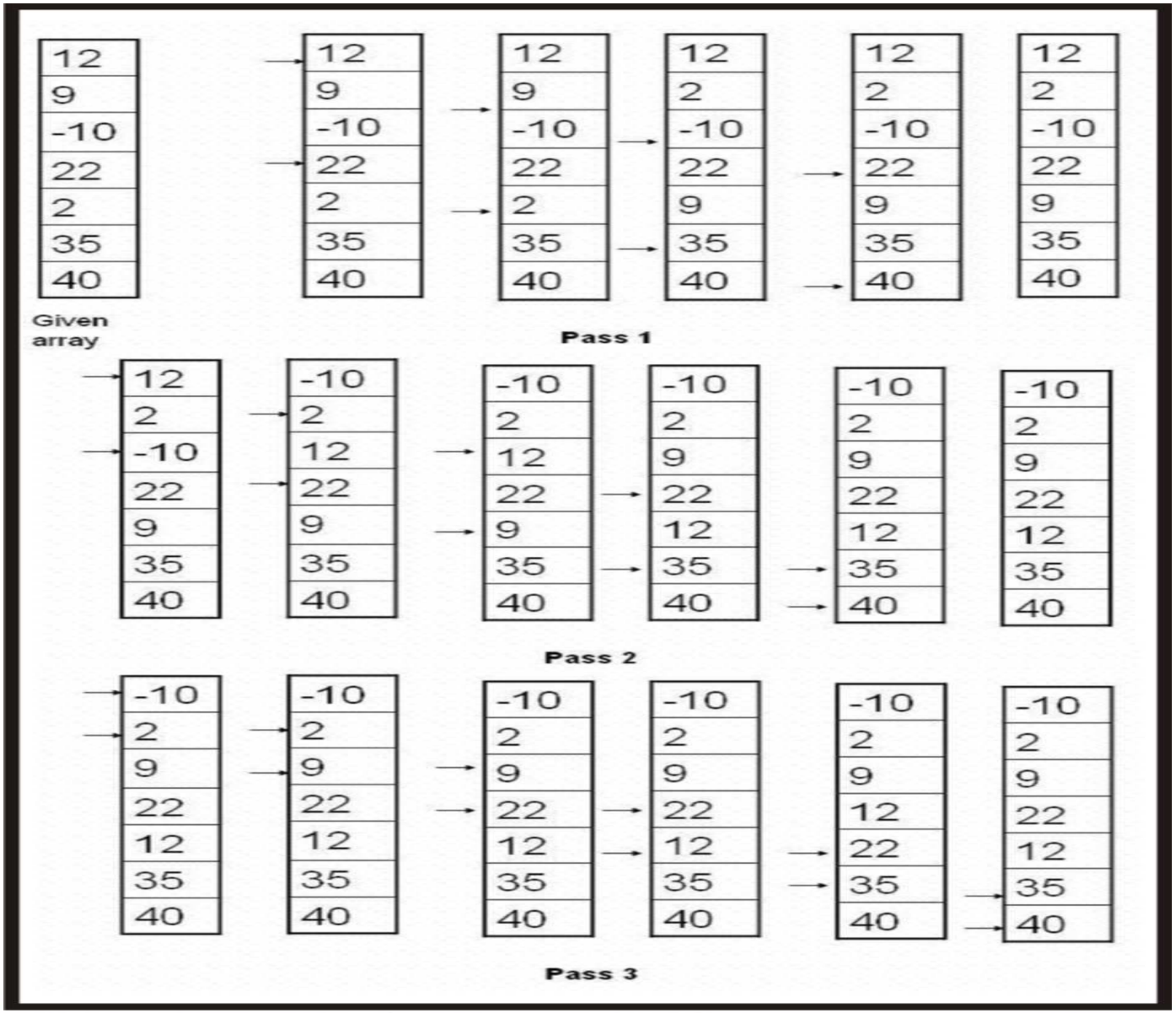
A diferença com relação ao InsertionSort:

É o número de segmentos do vetor.

Na inserção direta é considerado um único segmento do vetor onde os elementos são inseridos ordenadamente. No método do Shell são considerados diversos segmentos. Inserção direta movimenta elementos adjacentes.

Já o Shellsort permite a troca de elementos distantes.

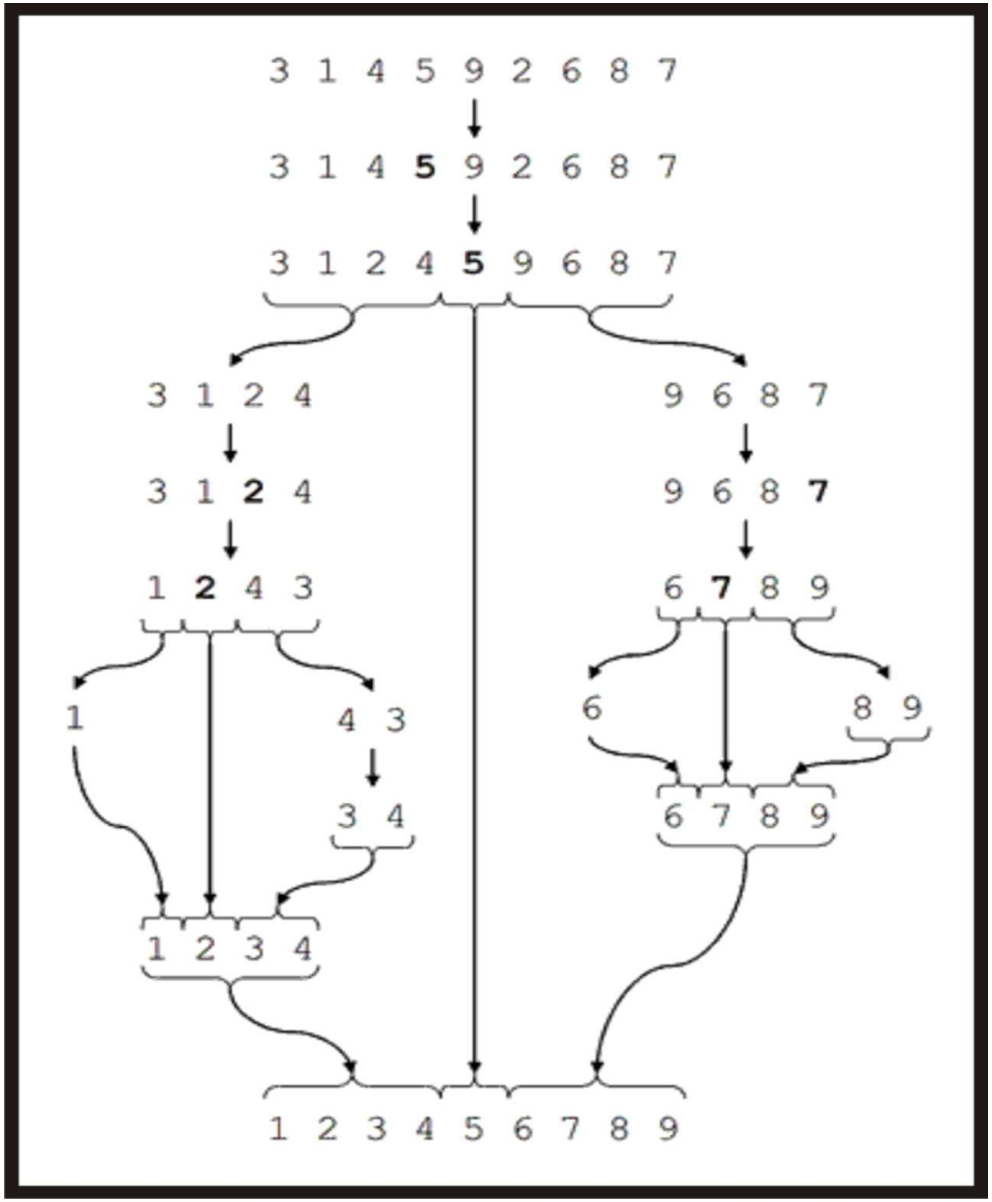
SHELLSORT



QUICKSORT

Adota a estratégia “Dividir para conquistar”.
O funcionamento resume-se a dividir o problema de ordenar o vetor em dois outros menores.

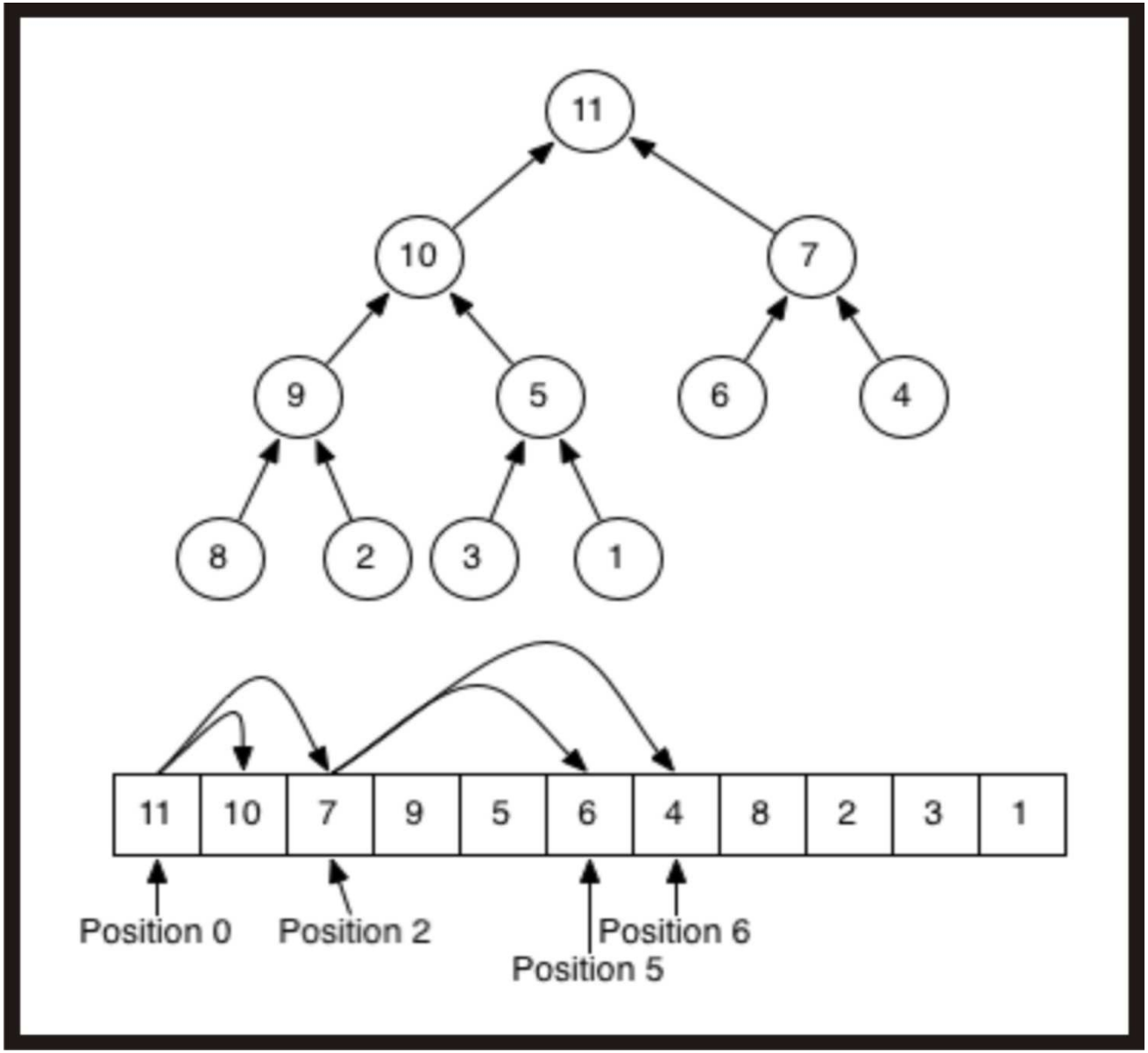
QUICKSORT


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

HEAPSORT

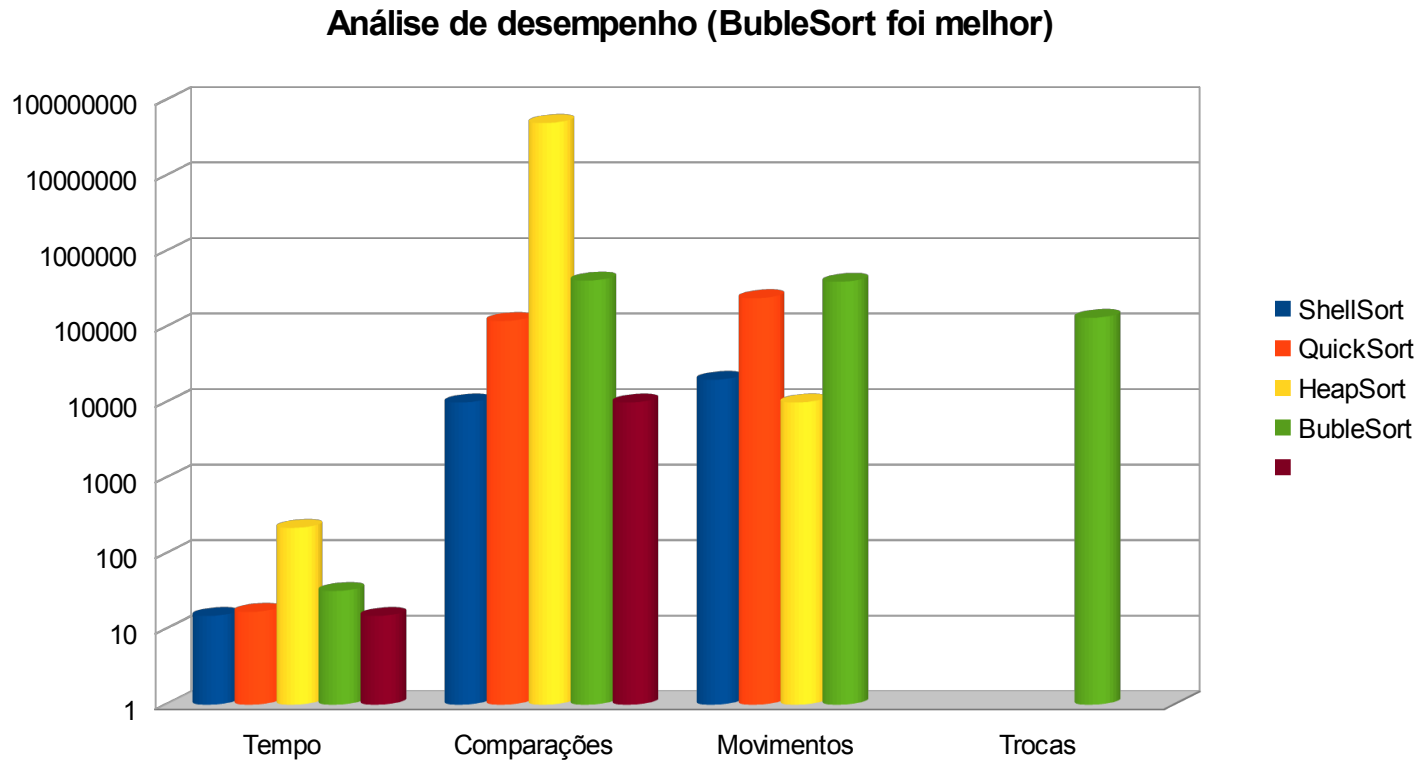
Faz parte da família de ordenação por seleção.
Utiliza uma estrutura de dados chamada heap, para ordenar os elementos a medida que os insere na estrutura.
Assim, ao final das inserções, os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada, lembrando-se sempre de manter a propriedade de max-heap.

HEAPSORT


11
10
7
9
5
6
4
8
2
3
1
11 10 7 9 5 6 4 8 2 3 1
Position 0
Position 2
Position 5
Position 6

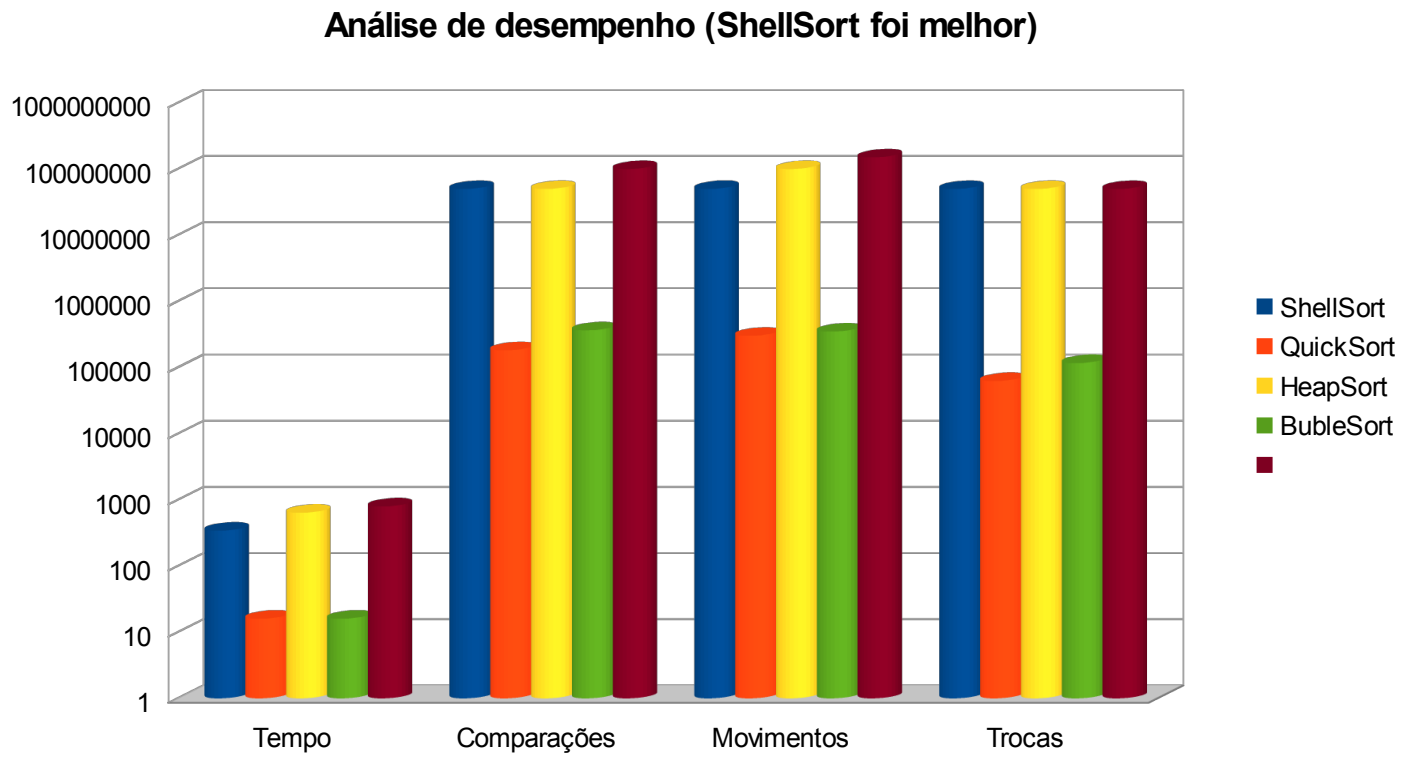
Análise de desempenho *(vetor ordenado ascendente)*

	InsertSort	ShellSort	QuickSort	HeapSort	BubleSort
Tempo	15	17	219	32	15
Comparações	10.000	120.006	49.995.001	410.869	10.000
Movimentos	19.998	240.010	9.999	395.868	0
Trocas	0	0	0	131.956	0



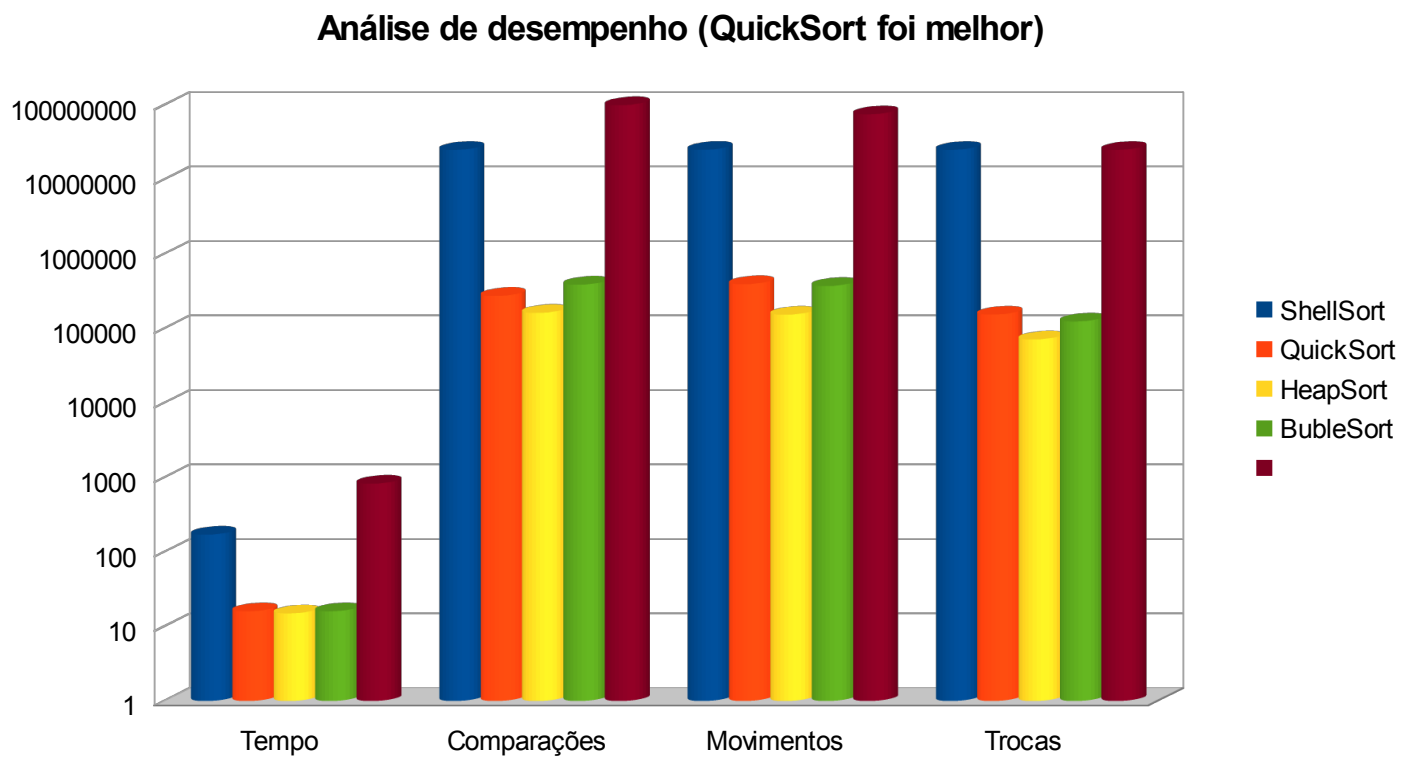
Análise de desempenho *(vetor ordenado descendente)*

	InsertSort	ShellSort	QuickSort	HeapSort	BubleSort
Tempo	343	16	639	16	796
Comparações	50.005.000	182.566	50.005.000	365.089	99.990.001
Movimentos	50.014.998	302.570	100.000.000	350.088	149.985.000
Trocas	49.995.000	62.560	49.995.000	116.696	49.995.000



Análise de desempenho *(vetor aleatório)*

	InsertSort	ShellSort	QuickSort	HeapSort	BubleSort
Tempo	171	16	15	16	827
Comparações	25.038.709	274.123	162.097	387.946	99.590.041
Movimentos	25.048.707	394.127	153.115	372.945	75.086.127
Trocas	25.028.709	154.117	71.558	124.315	25.028.709



CONCLUSÃO

Com este trabalho, pudemos entender melhor o funcionamento e a implementação de cinco tipos de algoritmos de ordenação.

Os objetivos iniciais do projeto foram alcançados com sucesso, pois foi possível analisar cada método de ordenação proposto *(em linguagem Java)* e fazer a análise de desempenho de cada um, identificando assim qual algoritmo é melhor indicado para cada caso.

REFERÊNCIAS

http://pt.wikipedia.org/wiki/Bubble_sort
http://pt.wikipedia.org/wiki/Quick_sort
http://pt.wikipedia.org/wiki/Insertion_sort
http://pt.wikipedia.org/wiki/Insertion_sort
http://pt.wikipedia.org/wiki/Shell_sort

Agradecemos a todos.

Espaço para dúvidas.