UNIVERSIDADE DE VILA VELHA

Breno Hombre Pimentel
Henrique Fernandes Cipriano
Hugo Capucho

Métodos de ordenação

Vila Velha
2014

Breno Hombre Pimentel
Henrique Fernandes Cipriano
Hugo Capucho

Métodos de ordenação

Trabalho apresentado como requisito parcial para obtenção de aprovação na disciplina Estrutura de dados I, no curso de Ciência da computação, na Universidade de Vila Velha.

Professor: Marcelo Novaes

Vila Velha
2014

SUMÁRIO

1 INTRODUÇÃO

Esse documento tem como finalidade de instruir o usuário a utilizar o programa e como os métodos de ordenação MergeSort, QuickSort, InsertionSort, SelectionSort trabalham em uma lista para a deixarem ordenada.

2 DESENVOLVIMENTO

2.1Utilização do programa

O programa que mescla a utilização dos quatro tipos de ordenação, Multisort é utilizado da seguinte forma na plataforma Linux:

./Multisort <Métodos de ordenação> <Tipo de Saída> <Arquivo> <Quantidade de números>

Método de ordenação: -q, -m, -i, -s

-q : QuickSort, -m , MergeSort, -i : InsertionSort, -s : SelectionSort

Tipo de Saída: -t, -b

-t : Texto, -b : Binário (ideal para salvar em arquivo, adicionando no final do comando, >nomeArquivo)

2.2QuickSort

O algoritmo QuickSort é do tipo divisão e conquista. Um algoritmo deste tipo resolve vários problemas quebrando um determinado problema em mais (e menores) subproblemas.

O algoritmo baseia-se na ideia simples de partir uma lista encadeada em três sub listas, de tal maneira que todos os elementos da primeira sub lista sejam menores que o Pivo da lista original, da segunda sub lista os iguais e da terceira sub lista os maiores. Estabelecida a divisão o problema estará resolvido, pois aplicando recursivamente a mesma técnica para a primeira e terceira sub lista, a lista estará ordenada ao se obter uma sub lista de apenas 1 elemento.

A grande vantagem desse algoritmo é que ele pode ser muito eficiente. Para facilitar a utilização da Lista encadeada, fora fixado o Pivo como o primeiro elemento da lista. Pode-se mostrar que, no melhor caso, o esforço computacional do algoritmo é proporcional a n log(n), e dizemos que o algoritmo é O(n log(n)).

A maior dificuldade foi fazer a chamada recursiva para o partition funcionar.

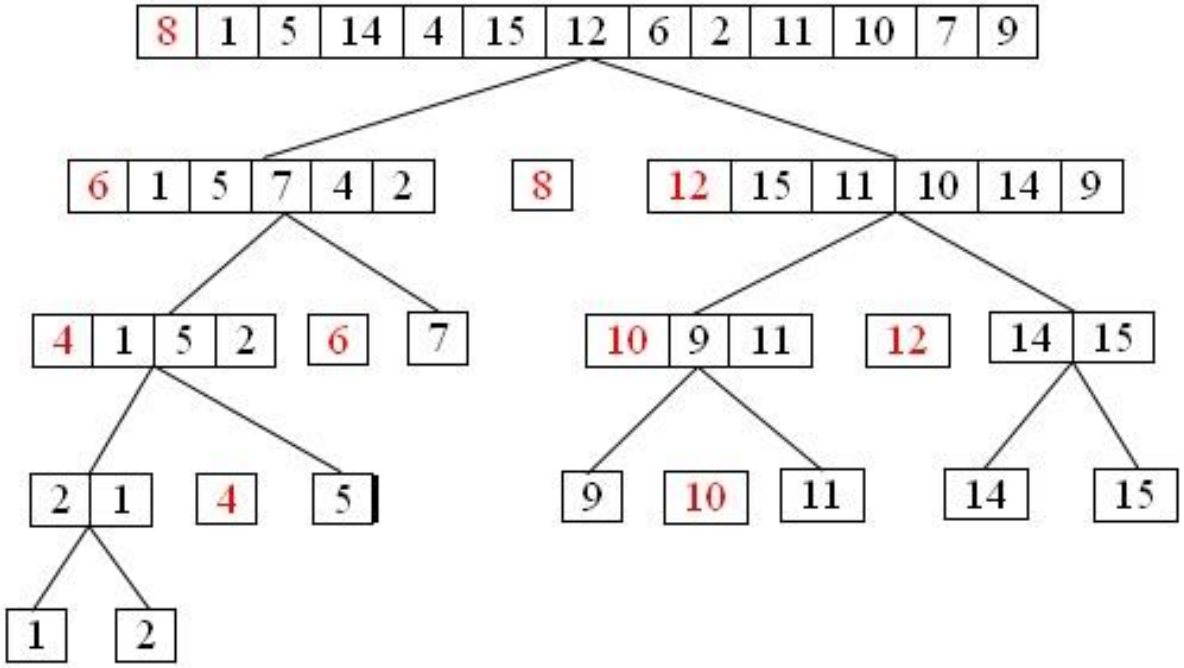
Funções especificas:

void partition(lista *L, lista *A, lista *B, lista *C)

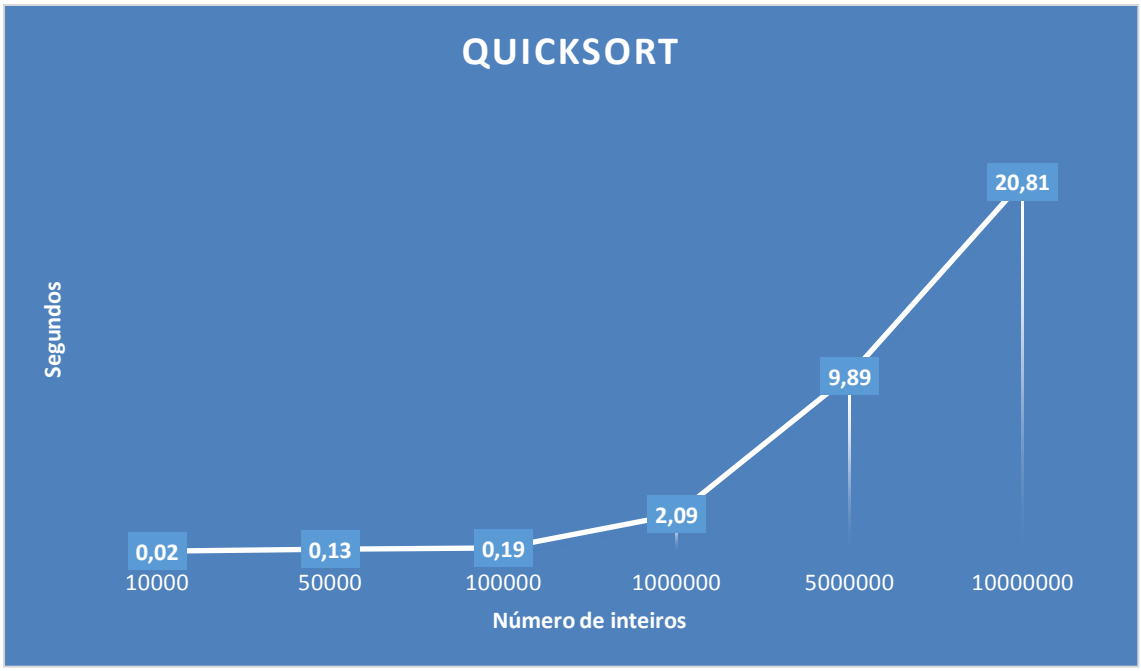
Faz a divisão da lista original em outras três sendo a segunda o pivô a primeira o nó que possui menor valor que o pivô e na terceira lista o nó que possui o maior valor que o pivô.

Recebe:4 lista uma contendo já preenchida e as outras em que ela será divida.

Retorna nada.



1	2	4	5	6	7	8	9	10	11	12	14	15



2.3MergeSort

Como o algoritmo *QuickSort,* o *MergeSort* é outro exemplo de algoritmo do tipo *divisão e conquista*, sendo é um algoritmo de ordenação por intercalação ou segmentação. A idéia básica é a facilidade de criar uma sequência ordenada a partir de duas outras também ordenadas. Para isso, o algoritmo divide a sequência original em pares de dados, ordena-as; depois as agrupa em sequências de quatro elementos, e assim por diante, até ter toda a sequência dividida em apenas duas partes.
A maior dificuldade foi em fazer a chamada recursiva para unir a duas lista com a lista original.

Funções especificas:

void divide(lista *, lista *, lista *)

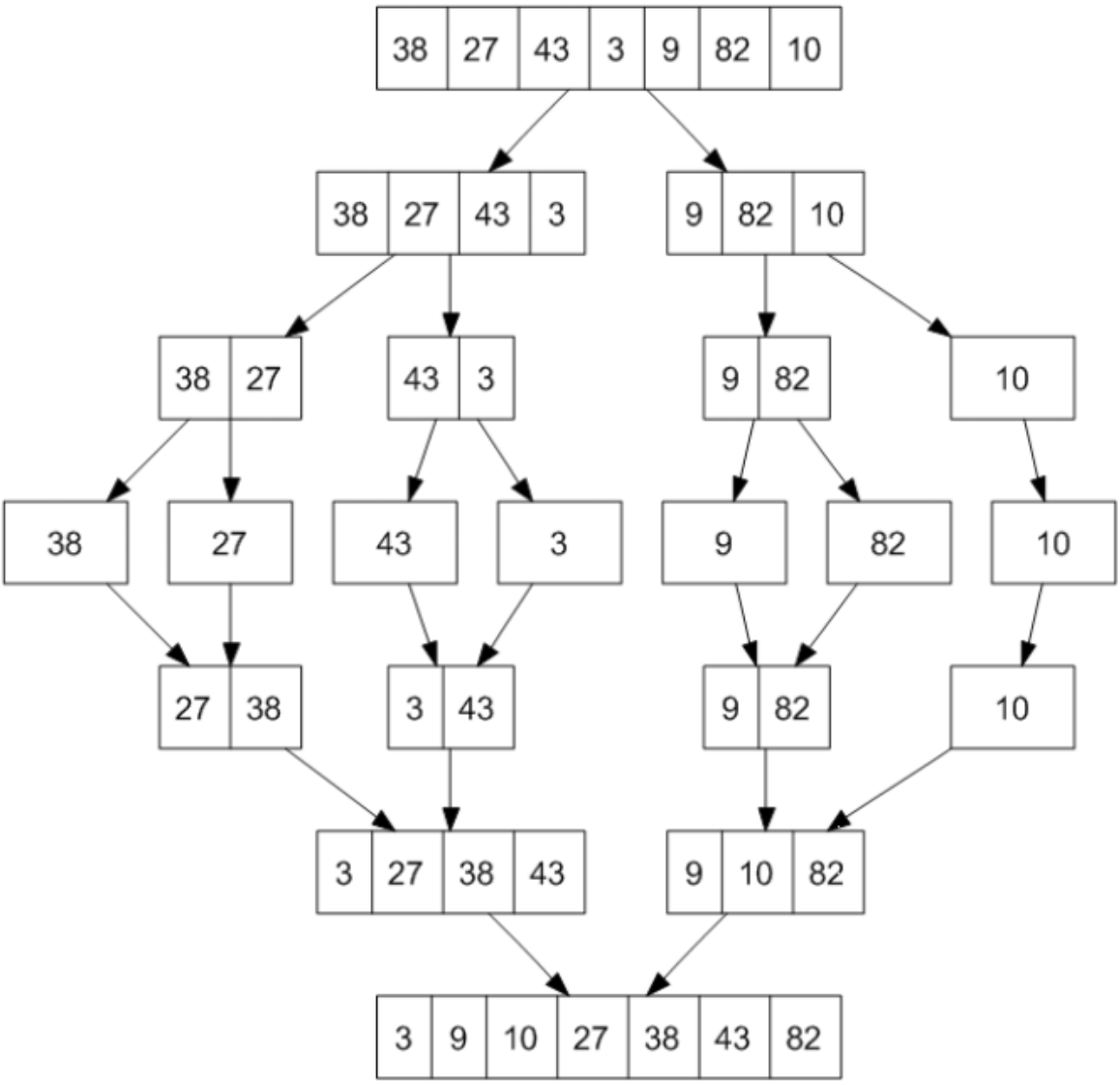
Divide uma lista em duas lista iguais.

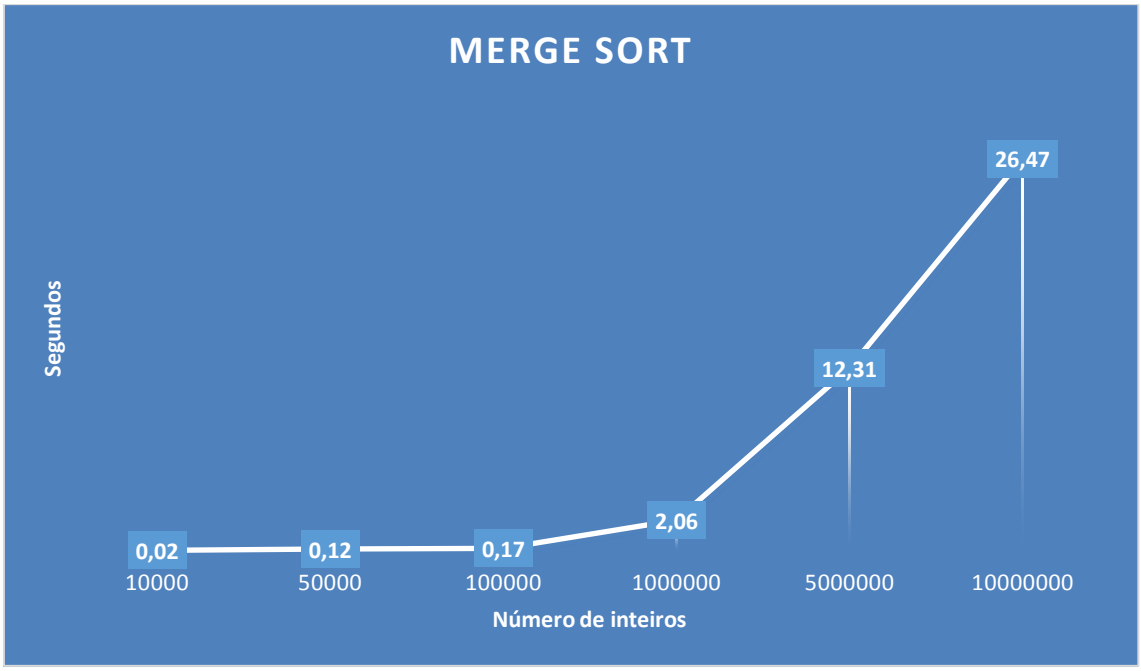
Recebe: Três lista, a primeira que ira se dividida na outras duas que são passadas

com parâmetro.

Retorna:Nada

void merge(lista *, lista *, lista *)
Faz a concatenação de duas lista para uma terceira
Recebe:Três lista e ira concatenar as duas ultimas na primeira lista.


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82


MERGE SORT
Segundos
0,02
0,12
0,17
2,06
12,31
26,47
10000
50000
100000
1000000
5000000
10000000
Número de inteiros

2.4 InsertionSort

A ordenação por inserção é um algoritmo simples e indicado para listas pequenas de valores a serem ordenados. Inicialmente, ela ordena os dois primeiros membros da lista, em seguida o algoritmo insere o terceiro membro na sua posição ordenada com relação aos dois primeiros membros. Na sequência, é inserido o quarto elemento na lista dos três primeiros elementos e o processo continua até que toda a lista esteja ordenada.

Inicialmente, ela ordena os dois primeiros membros da lista, em seguida o algoritmo insere o terceiro membro na sua posição ordenada com relação aos dois primeiros membros. Na sequência, é inserido o quarto elemento na lista dos três primeiros elementos e o processo continua até que toda a lista esteja ordenada. O número de comparações é n^2 e no pior caso e no melhor caso é 2(n-1).

A dificuldade de implementação foi fazer o menor nó ir ate a o começo da lista.

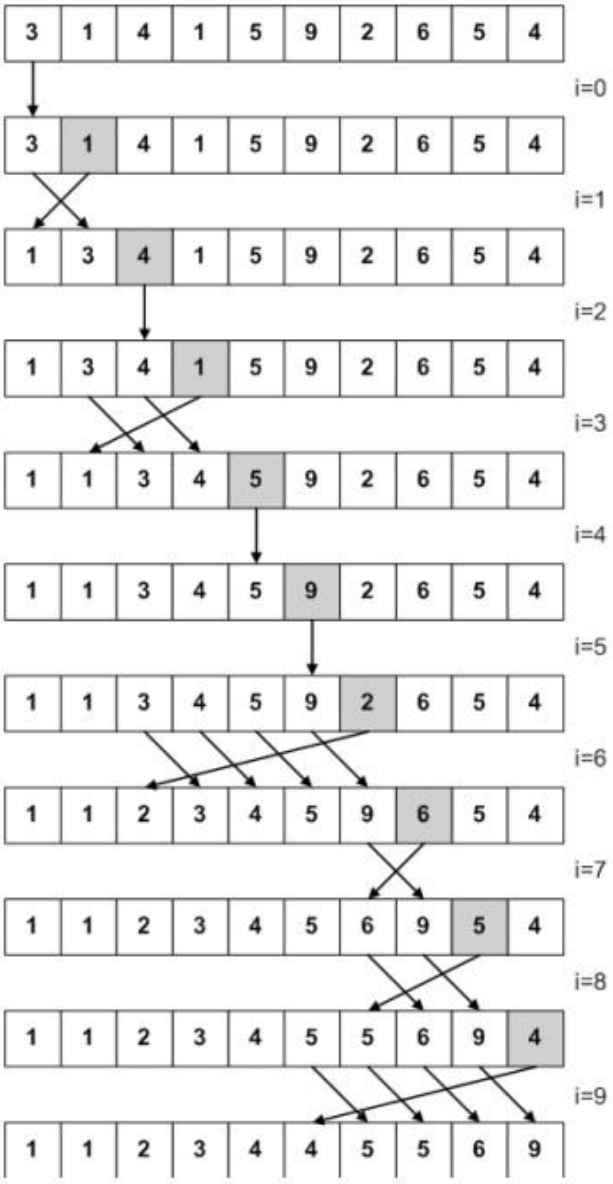
Funções especificas:

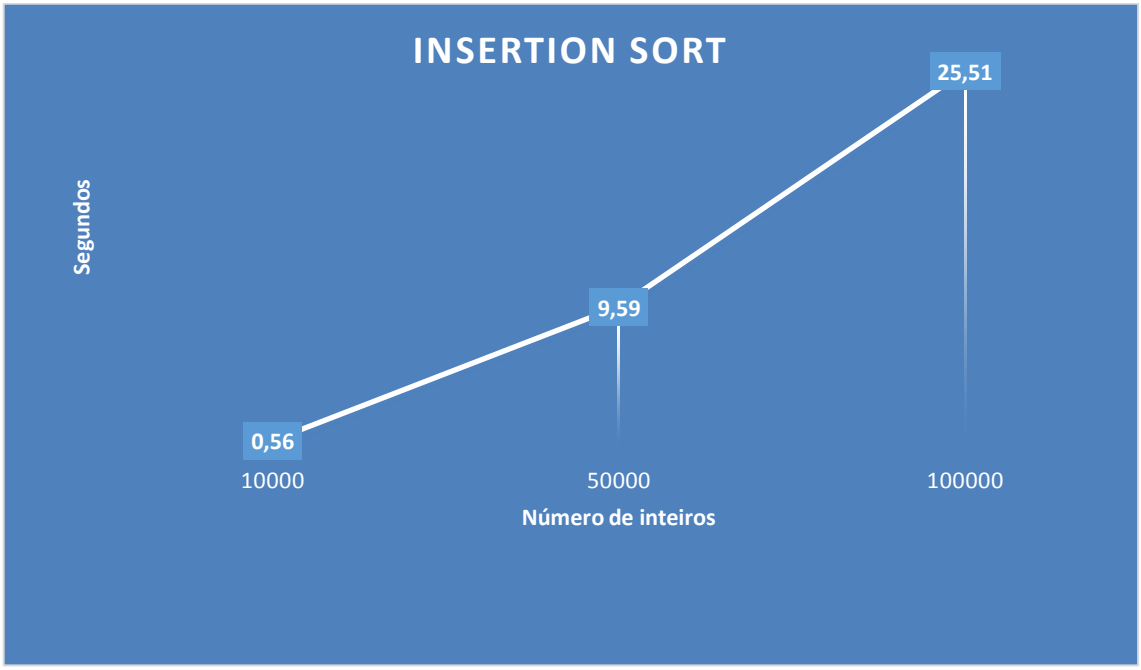
node* swap(lista , node , node)

Altera a posição na lista de dois nós que estão conectado.

Recebe: A lista é dois nós que irão mudar de posição.

Retorna: O segundo nó que trocou de posição.


3 1 4 1 5 9 2 6 5 4
i=0
3 1 4 1 5 9 2 6 5 4
i=1
1 3 4 1 5 9 2 6 5 4
i=2
1 3 4 1 5 9 2 6 5 4
i=3
1 1 3 4 5 9 2 6 5 4
i=4
1 1 3 4 5 9 2 6 5 4
i=5
1 1 3 4 5 9 2 6 5 4
i=6
1 1 2 3 4 5 9 6 5 4
i=7
1 1 2 3 4 5 6 9 5 4
i=8
1 1 2 3 4 5 5 6 9 4
i=9
1 1 2 3 4 4 5 5 6 9


INSERTION SORT
Segundos
25,51
9,59
0,56
10000
50000
100000
Número de inteiros

2.5SelectionSort

O princípio do SelectionSort é selecionar o elemento correto para cada posição. Assim na primeira etapa seleciona-se o menor (ou maior) elemento da lista e o colocar na primeira posição e assim sucessivamente ate chega ao penúltimo nó não sendo necessário ordenar o ultimo, pois já se encontra na posição correta. A complexidade desse algoritmo é n^2. O processo pode ser visto nas imagens abaixo.

A maior dificuldade encontrada para implementar o algoritmo foi fazer a função trocar os nós sem perde a sequencia da lista.

Funções especificas:

node* troca(lista, node* ,node*)

Faz a troca de posições de dois nós que estão na lista.

Recebe: A lista e os dois nós que irão se trocados

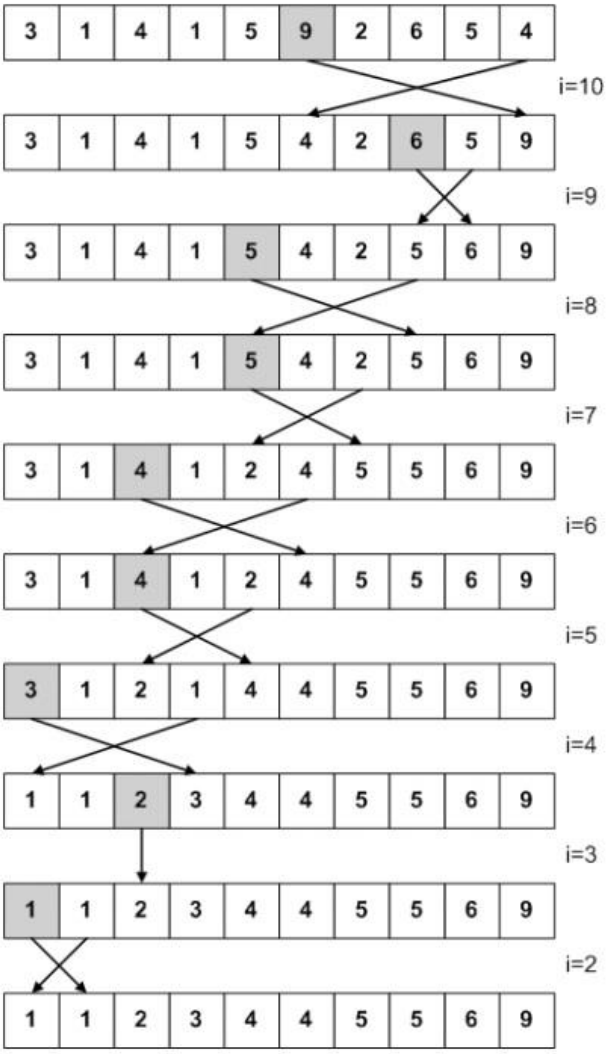
Retorna: O segundo nó.

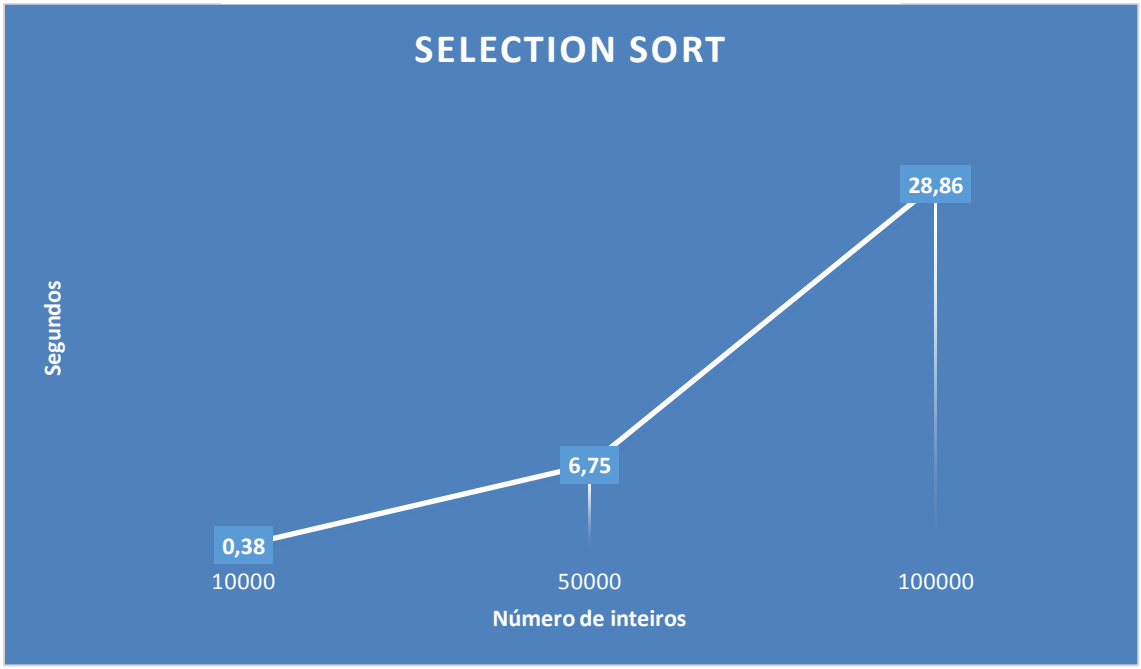
node *achaMenor(lista *,node*);

Percorre a lista a fim de achar o nó que possui o menor valor.

Recebe: Uma lista.

Retorna: O nó que possui o menor valor.


3 1 4 1 5 9 2 6 5 4
i=10
3 1 4 1 5 4 2 6 5 9
i=9
3 1 4 1 5 4 2 5 6 9
i=8
3 1 4 1 5 4 2 5 6 9
i=7
3 1 4 1 2 4 5 5 6 9
i=6
3 1 4 1 2 4 5 5 6 9
i=5
3 1 2 1 4 4 5 5 6 9
i=4
1 1 2 3 4 4 5 5 6 9
i=3
1 1 2 3 4 4 5 5 6 9
i=2
1 1 2 3 4 4 5 5 6 9


SELECTION SORT
Segundos
28,86
6,75
0,38
10000
50000
100000
Número de inteiros

3 CONSIDERAÇÕES FINAIS

Podemos observar desde os métodos de ordenação simples não estáveis, até os sofisticados e estáveis. Conhece-los, é de suma importância para distinguir a melhor adequação à um projeto implementado. Os algoritmos foram implementados através da Lista Encadeada, embora à abordagem Lista x Vetor não deva ser esquecida.

REFERENCIAS

Estrutura de Dados com Algoritmos e C- Marcos Laureano