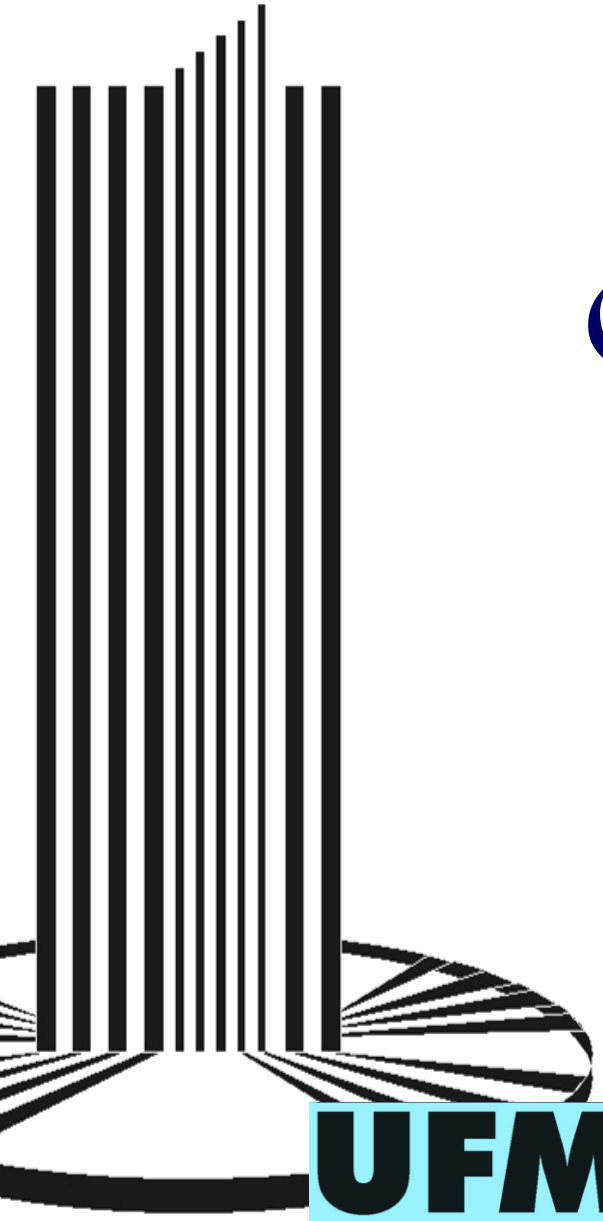


Ordenação Interna

Profª Yorah Bosse

yorah.bosse@gmail.com
yorah.bosse@ufms.br

Roteiro

- Ordenações Internas
  - Seleção Direta ou Linear (Selection Sort)
  - Seleção Direta Otimizada ou Linear Otimizada (*Delayed Selection Sort)*
  - Inserção (Insertion Sort)
  - Método Bolha (Bubble Sort)
  - Incrementos Decrescentes (Shell Sort)
  - Merge Sort
  - Método de Troca e Partições (Quick Sort)
  - Seleção em Árvore Binária (Heap Sort)

O.I.- Selection Sort

- troca os elementos de lugar toda vez que vet [i] > vet [x]
- após o término da primeira iteração, vet [0] terá o menor valor
- vet [1] terá o segundo menor valor após a segunda iteração, e assim sucessivamente

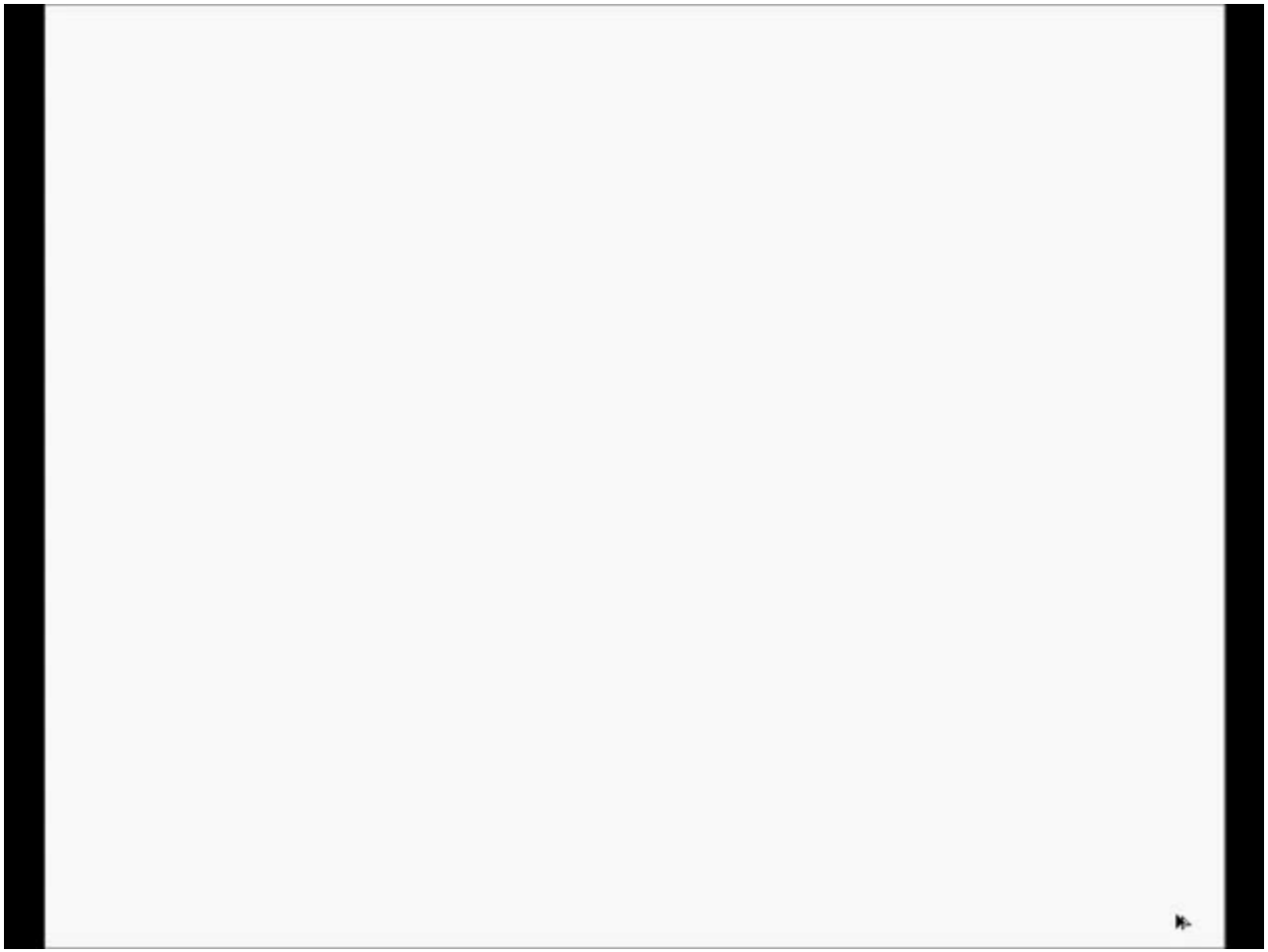
```
void direta(int *vet){
    int aux;
    for (int i_menor = 0; i_menor < max-1; i_menor++)
        for (int i_maior = i_menor+1; i_maior < max; i_maior++)
            if (vet[i_menor] > vet[i_maior]){
                aux = vet[i_menor];
                vet[i_menor] = vet[i_maior];
                vet[i_maior] = aux;
            }
}
```

O.I.- Selection Sort

Vantagens
- Fácil Implementação
- Pequeno número de movimentações
- Interessante para arquivos pequenos

Desvantagens
- O fato de o arquivo já estar ordenado não influencia em nada
- Ordem de complexidade quadrática
- Algoritmo não estável

O.I.- Selection Sort



Fonte: http://www.youtube.com/watch?v=BSXIolKg5F8

Roteiro

- Ordenações Internas
  - Seleção Direta ou Linear (Selection Sort)
  - Seleção Direta Otimizada ou Linear Otimizada (*Delayed Selection Sort)*
  - Inserção (Insertion Sort)
  - Método Bolha (Bubble Sort)
  - Incrementos Decrescentes (Shell Sort)
  - Merge Sort
  - Método de Troca e Partições (Quick Sort)
  - Seleção em Árvore Binária (Heap Sort)

O.I. - Delayed Selection Sort

- é uma otimização do método de Seleção Direta (ou Linear)
- a posição do menor elemento é guardada para que ele seja trocado apenas uma vez de lugar

```
void otimizada(int *vet){
    int aux, i_troca;
    for (int i_menor = 0; i_menor < max-1; i_menor++){
        i_troca = i_menor;
        for (int i_maior = i_menor+1; i_maior < max; i_maior++)
            if (vet[i_troca] > vet[i_maior])
                i_troca = i_maior;
            aux = vet[i_menor];
            vet[i_menor] = vet[i_troca];
            vet[i_troca] = aux;
    }
}
```

Roteiro

- Ordenações Internas
  - Seleção Direta ou Linear (Selection Sort)
  - Seleção Direta Otimizada ou Linear Otimizada (Delayed Selection Sort)
  - Inserção (Insertion Sort)
  - Método Bolha (Bubble Sort)
  - Incrementos Decrescentes (Shell Sort)
  - Merge Sort
  - Método de Troca e Partições (Quick Sort)
  - Seleção em Árvore Binária (Heap Sort)

O.I.- Insertion Sort

- cada valor que pegamos, é inserido, de uma vez, em seu lugar correto

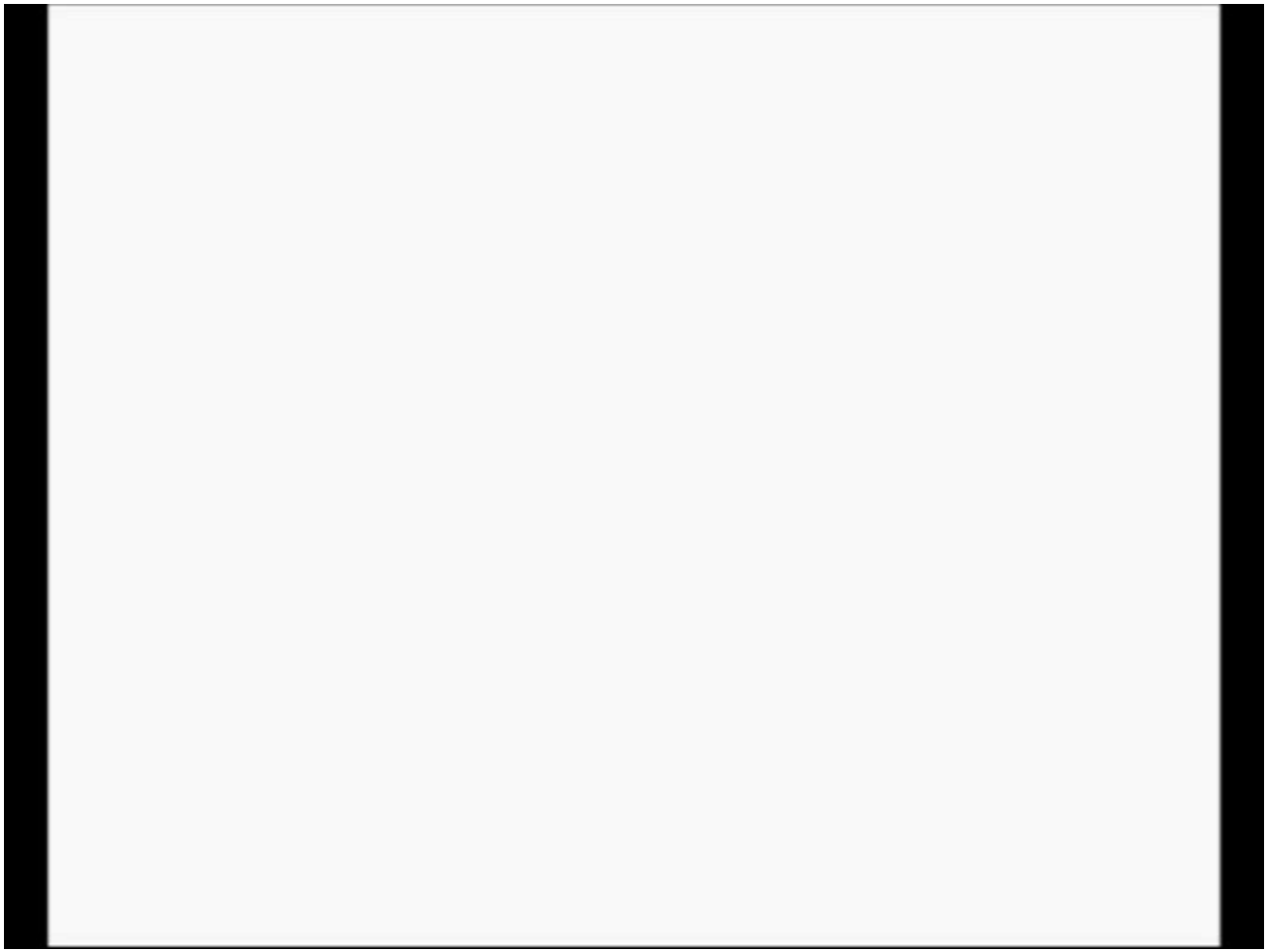
```
void insercao(int *vet){
        int aux, anterior;
        for (int atual = 1; atual < max; atual++){
                aux = vet[atual];
                anterior = atual - 1;
                while (anterior > -1 && vet[anterior] > aux){
                        vet[anterior+1] = vet[anterior];
                        anterior--;
                }
                vet[anterior+1] = aux;
        }
}
```

O.I.- Insertion Sort

Vantagens
- Fácil Implementação
- Algoritmo Estável
- O vetor já ordenado favorece a ordenação

Desvantagens
- Número grande de movimentações
- Ordem de complexidade quadrática
- Ineficiente quando o vetor está ordenado inversamente;

O.I.- Insertion Sort



Fonte: http://www.youtube.com/watch?v=-Z00it6Nkz8

Roteiro

- Ordenações Internas
  - Seleção Direta ou Linear (Selection Sort)
  - Seleção Direta Otimizada ou Linear Otimizada (Delayed Selection Sort)
  - Inserção (Insertion Sort)
  - Método Bolha (Bubble Sort)
  - Incrementos Decrescentes (Shell Sort)
  - Merge Sort
  - Método de Troca e Partições (Quick Sort)
  - Seleção em Árvore Binária (Heap Sort)

O.I. - Bubble Sort

- este método utiliza uma variável, aqui denominada de BOLHA, para guardar qual a última posição que foi trocada de lugar
- na iteração seguinte, este valor é passado para a variável que indica o tamanho do vetor, sendo que as posições já ordenadas não são mais verificadas.

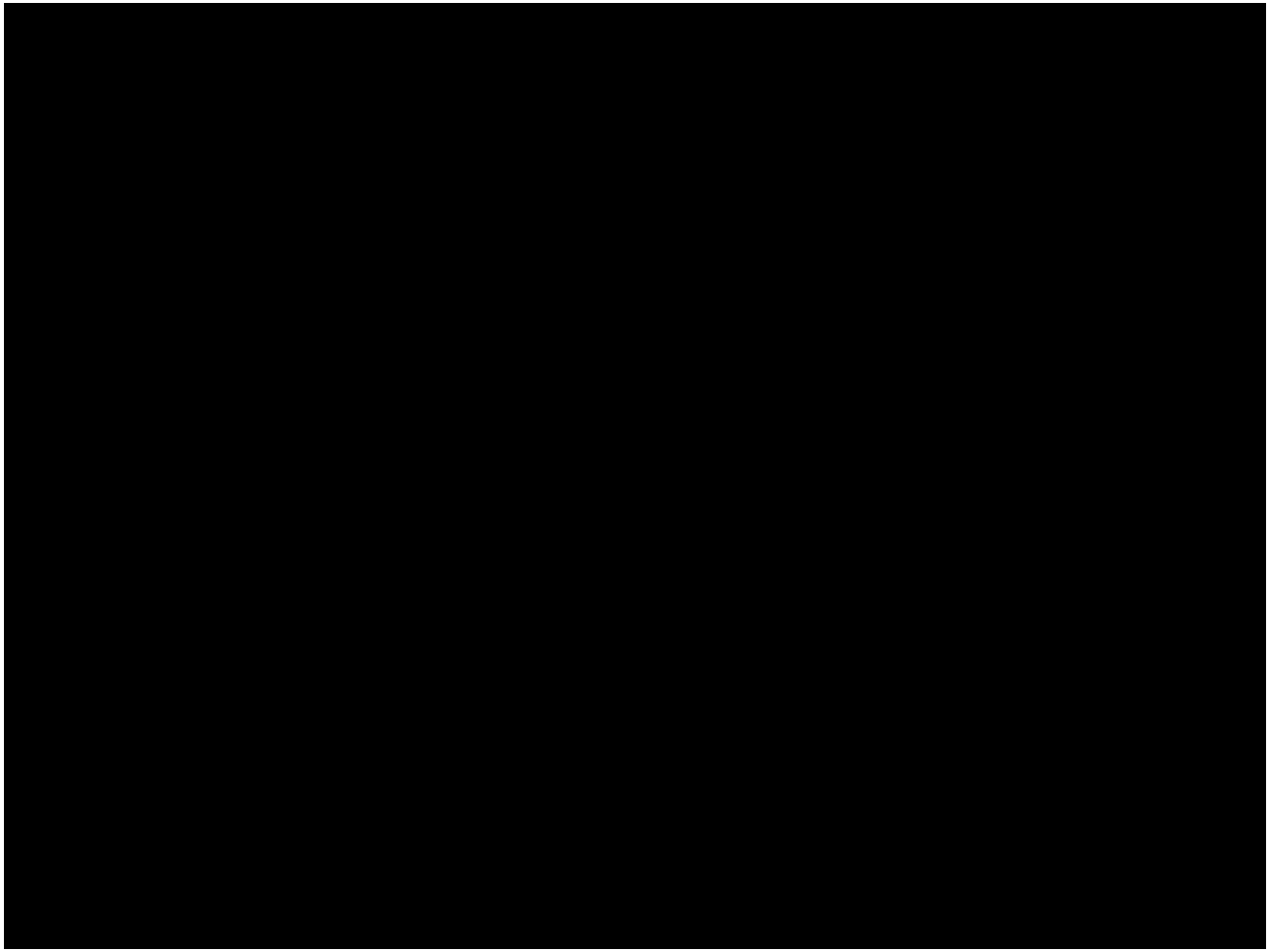
```
void bolha(int *vet){
        int lsup, bolha, aux;
        lsup = max-1;
        while (lsup > 0)  {
                bolha = -1;
                for (int i = 0; i < lsup; i++)
                if (vet[i] > vet[i+1])  {
                                aux = vet[i];
                                vet[i] = vet[i+1];
                                vet[i+1] = aux;
                                bolha = i;
                }
                lsup = bolha;
        }
}
```

O.I. - Bubble Sort

Vantagens
- Fácil Implementação;
- Algoritmo Estável;

Desvantagens
- O fato de o arquivo já estar ordenado não ajuda em nada [7];
- Ordem de complexidade quadrática;

O.I. - Bubble Sort



Fonte: http://www.youtube.com/watch?v=llX2SpDkQDc

Roteiro

- Ordenações Internas
  - Seleção Direta ou Linear (Selection Sort)
  - Seleção Direta Otimizada ou Linear Otimizada (Delayed Selection Sort)
  - Inserção (Insertion Sort)
  - Método Bolha (Bubble Sort)
  - Incrementos Decrescentes (Shell Sort)
  - Merge Sort
  - Método de Troca e Partições (Quick Sort)
  - Seleção em Árvore Binária (Heap Sort)

O.I. – Shell Sort

- este método ordena pares de elementos separados por uma distância D
- D é reduzido a cada passo até ser menor que 1
- segundo Feldman (apud Lopes, 1999), o valor indicado para D é um número primo
- no algoritmo abaixo, foi utilizada a metade do tamanho total do vetor
- este método é superior ao de inserção, seleção direta, seleção direta otimizada e bolha

O.I. – Shell Sort

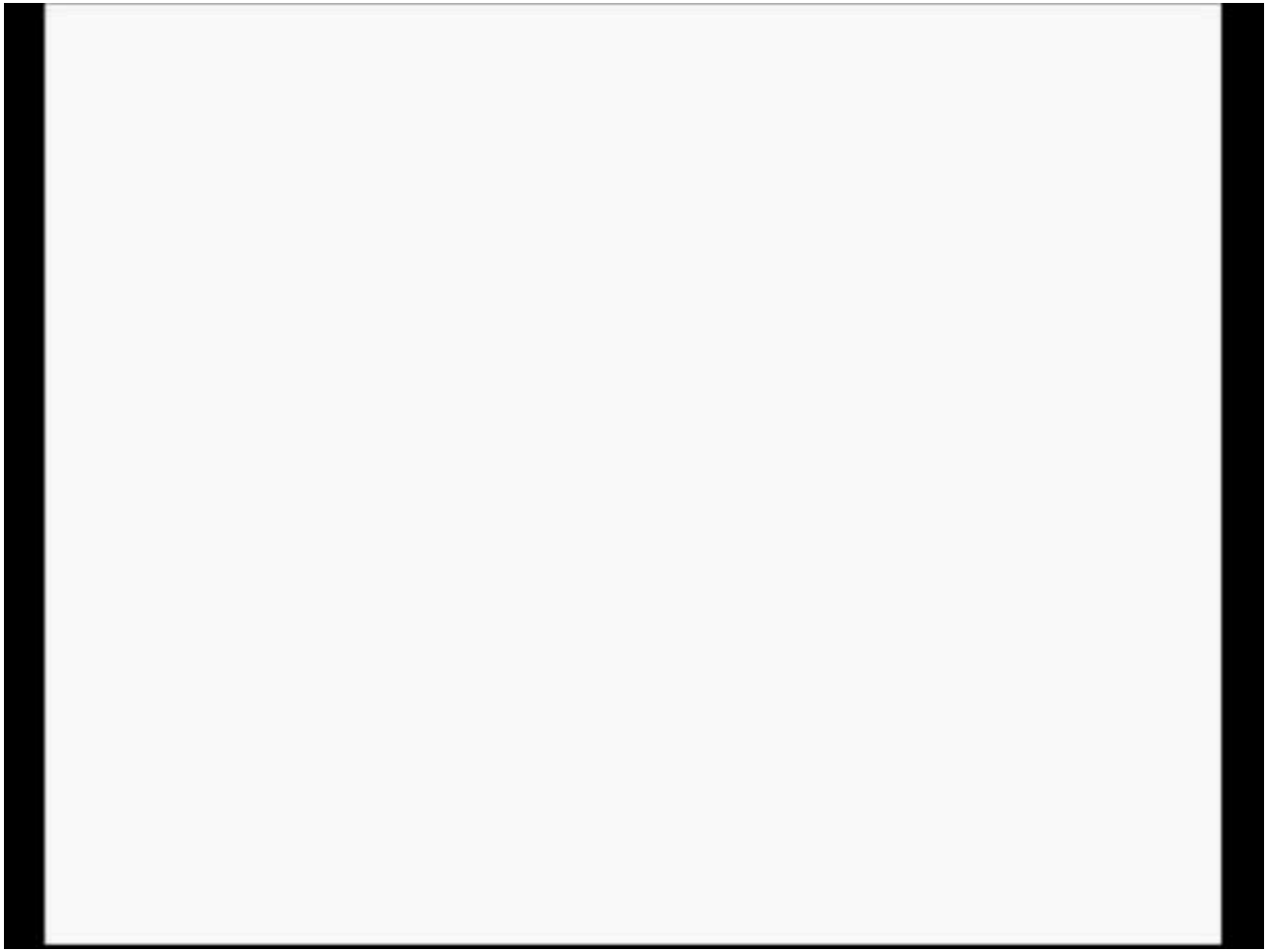
Vantagens
- Código Simples
- Interessante para arquivos de tamanho moderado

Desvantagens
- Algoritmo não estável
- Tempo de execução sensível à ordem inicial do arquivo [11]

O.I. – Shell Sort

```
void shell(int *vet){
        int lsup = max,
        d = lsup / 2,
        i,
        x,
        aux;
        while (d > 0) {
                for (i = d; i < lsup; i++) {
                        x = i - d;
                        while (x >= 0 && vet[x] > vet[x+d]) {
                                aux = vet[x];
                                vet[x] = vet[x+d];
                                vet[x+d] = aux;
                                x = x - d;
                        }
                }
                d = d / 2;
        }
}
```

O.I. – Shell Sort



Fonte: http://www.youtube.com/watch?v=BSdReI-6FOA

Roteiro

- Ordenações Internas
  - Seleção Direta ou Linear (Selection Sort)
  - Seleção Direta Otimizada ou Linear Otimizada (Delayed Selection Sort)
  - Inserção (Insertion Sort)
  - Método Bolha (Bubble Sort)
  - Incrementos Decrescentes (Shell Sort)
  - Merge Sort
  - Método de Troca e Partições (Quick Sort)
  - Seleção em Árvore Binária (Heap Sort)

O.I. - Merge Sort

- segundo Lopes, o termo Merge é utilizado para indicar a união de dois ou mais arquivos ordenados
- neste caso, “arquivos” pode ser lido como um conjunto de dados
- Mergesort realiza a ordenação de um conjunto de dados V a partir do Merge (união) da ordenação das duas metades de V
- cada metade é ordenada da mesma forma, utilizando-se recursividade
- A desvantagem é que é utilizada duas vezes a área ocupada pelo vetor (o VET e o TEMP)
- na primeira chamada da função, esq receberá 0 e dir receberá o valor do índice da última posição do vetor

O.I. - Merge Sort

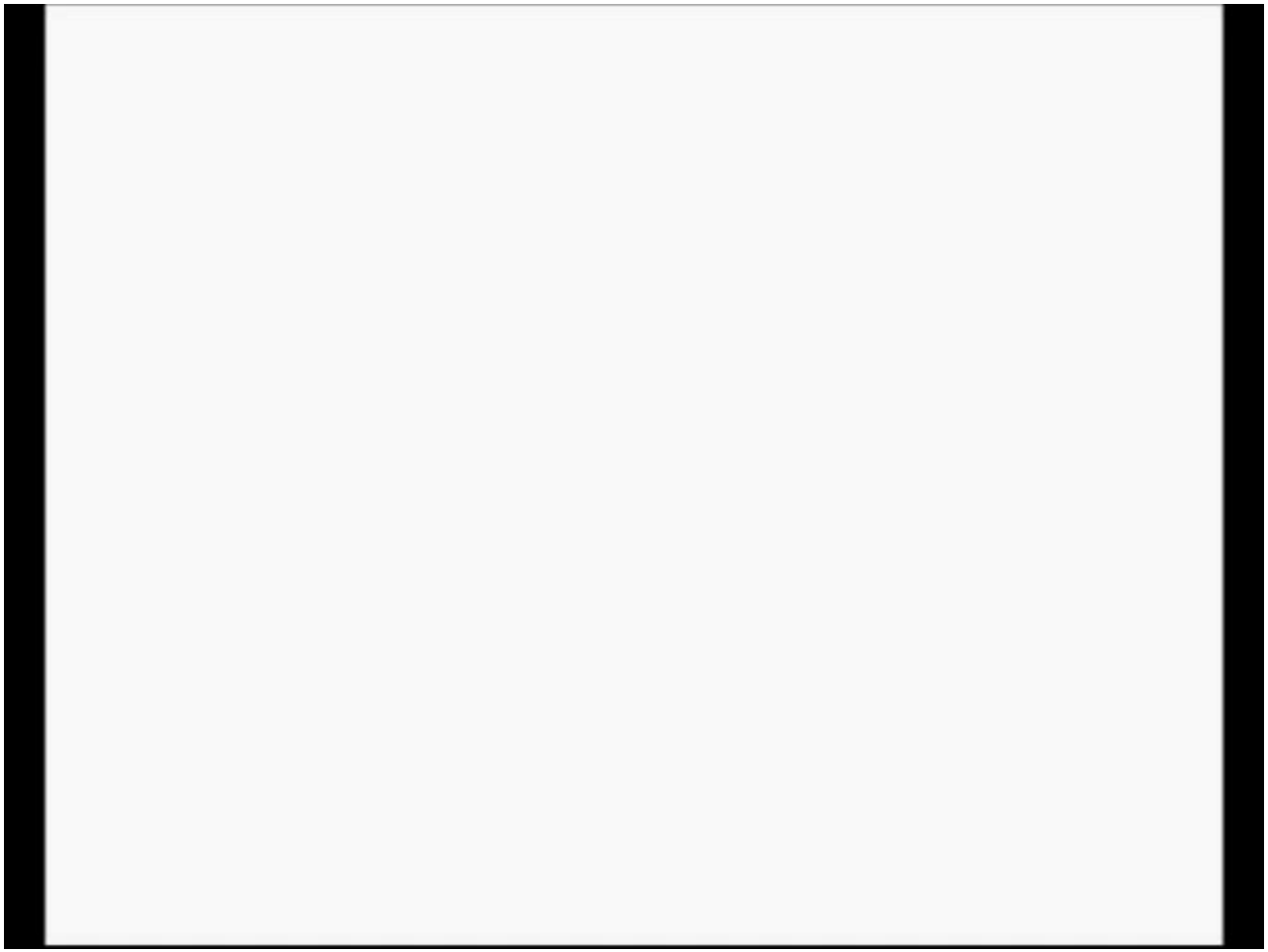
Vantagens
- Passível de ser transformado em estável
- Fácil Implementação
- Complexidade $O(n \log n)$

Desvantagens
- Utiliza memória auxiliar
- Mais lento que o ***HeapSort***

O.I. – Merge Sort

```
void ordenacao_mergesort (int *vet, int esq, int dir){
      int temp [max];
      int i, j, k, metade;
      if (dir - esq > 0){
        metade = (dir + esq) / 2;
        ordenacao_mergesort (vet,esq,metade);
        ordenacao_mergesort (vet,metade+1,dir);
        for (i = esq; i <= metade; i++)
          temp[i] = vet[i];
        for (i = metade+1; i <= dir; i++)
          temp[dir+metade+1-i] = vet[i];
        i = esq;
        j = dir;
        for (k = esq; k <= dir; k++)
          if (temp[i] < temp[j]){
            vet[k] = temp[i];
            i++;
          }
                else{
                  vet[k] = temp[j];
                  j--;
                }
        }
}
```

O.I. - Merge Sort



Fonte: http://www.youtube.com/watch?v=cDNqk4tdvqQ

Roteiro

- Ordenações Internas
  - Seleção Direta ou Linear (Selection Sort)
  - Seleção Direta Otimizada ou Linear Otimizada (Delayed Selection Sort)
  - Inserção (Insertion Sort)
  - Método Bolha (Bubble Sort)
  - Incrementos Decrescentes (Shell Sort)
  - Merge Sort
  - Método de Troca e Partições (Quick Sort)
  - Seleção em Árvore Binária (Heap Sort)

O.I. - Quick Sort

- este método de ordenação consiste em dividir a área de ordenação em duas partições, de modo que os elementos da partição da esquerda sejam menores ou iguais aos elementos da direita. O método é então aplicado de forma recursiva sobre cada partição
- comparado com o Mergesort, tem a vantagem de não requerer uma área complementar.

O.I. - Quick Sort

Vantagens
- Extremamente Eficiente
- Necessita apenas de um pequena pilha como memória extra
- Complexidade $n \log n$

Desvantagens
- Tem um pior caso de $O(n^2)$
- Implementação difícil
- Não é estável

O.I. – Quick Sort

```
void ordenacao_quicksort (int *vet, int prim, int ult){
    int i, j, x, y;
    i = prim;
    j = ult;
    x = vet[(prim+ult)/2];
    do{
        while (vet[i] < x)
            i++;
        while (x < vet[j])
            j--;
        if (i <= j){
            y = vet[i];
            vet[i] = vet[j];
            vet[j] = y;
            i++;
            j--;
        }
    }while (i<=j);
```

```
  if (prim < j)
        ordenacao_quicksort(vet,prim,j);
     if (i < ult)
        ordenacao_quicksort(vet,i,ult);
}
```

Fonte: http://www.youtube.com/watch?v=gu7x-jgzuOU

- Ordenações Internas
  - Seleção Direta ou Linear (Selection Sort)
  - Seleção Direta Otimizada ou Linear Otimizada (Delayed Selection Sort)
  - Inserção (Insertion Sort)
  - Método Bolha (Bubble Sort)
  - Incrementos Decrescentes (Shell Sort)
  - Merge Sort
  - Método de Troca e Partições (Quick Sort)
  - Seleção em Árvore Binária (Heap Sort)

O.I. - Heap Sort

- utiliza a árvore binária para ordenar o vetor
- o Heapsort gasta um tempo de execução proporcional a n log n, no pior caso.
- a ordenação é realizada em duas fases:
  1. é montada a árvore binária – HEAP, contendo todos os valores do vetor, sendo que, o valor de qualquer nó seja maior do que os valores dos seus sucessores, ou seja, o pai sempre terá o valor maior do que os dos seus filhos
  2. o “heap” é usado para a seleção dos elementos na ordem desejada

Vantagens
- Tempo $n \log n$

Desvantagens
- O anel interno do algoritmo é bastante complexo se comparado ao ***Quicksort***.
- Não é estável

O.I. - Heap Sort

```
void monta_heap (int *vet, int e, int d)
{
   int i, j, x;
   i = e;
   j = (i * 2) + 1;
   x = vet[i];
   while (j <= d)   {
      if (j < d && vet[j] < vet[j+1])
        j++;
      if (x < vet[j])   {
        vet[i] = vet[j];
        i = j;
        j = (i * 2) + 1;
      }
      else
        j = d + 1;
   }
   vet[i] = x;
}
```

```
void heap(int *v){
            int esq = max / 2,
                dir = max-1,
                aux;
            while (esq > 0)   {
                esq--;
                monta_heap (v,esq,dir);
            }
            while (dir > 0)
            {
                aux = v[0];
                v[0] = v[dir];
                v[dir] = aux;
                dir--;
                monta_heap (v,esq,dir);
            }
}
```

O.I. - Heap Sort

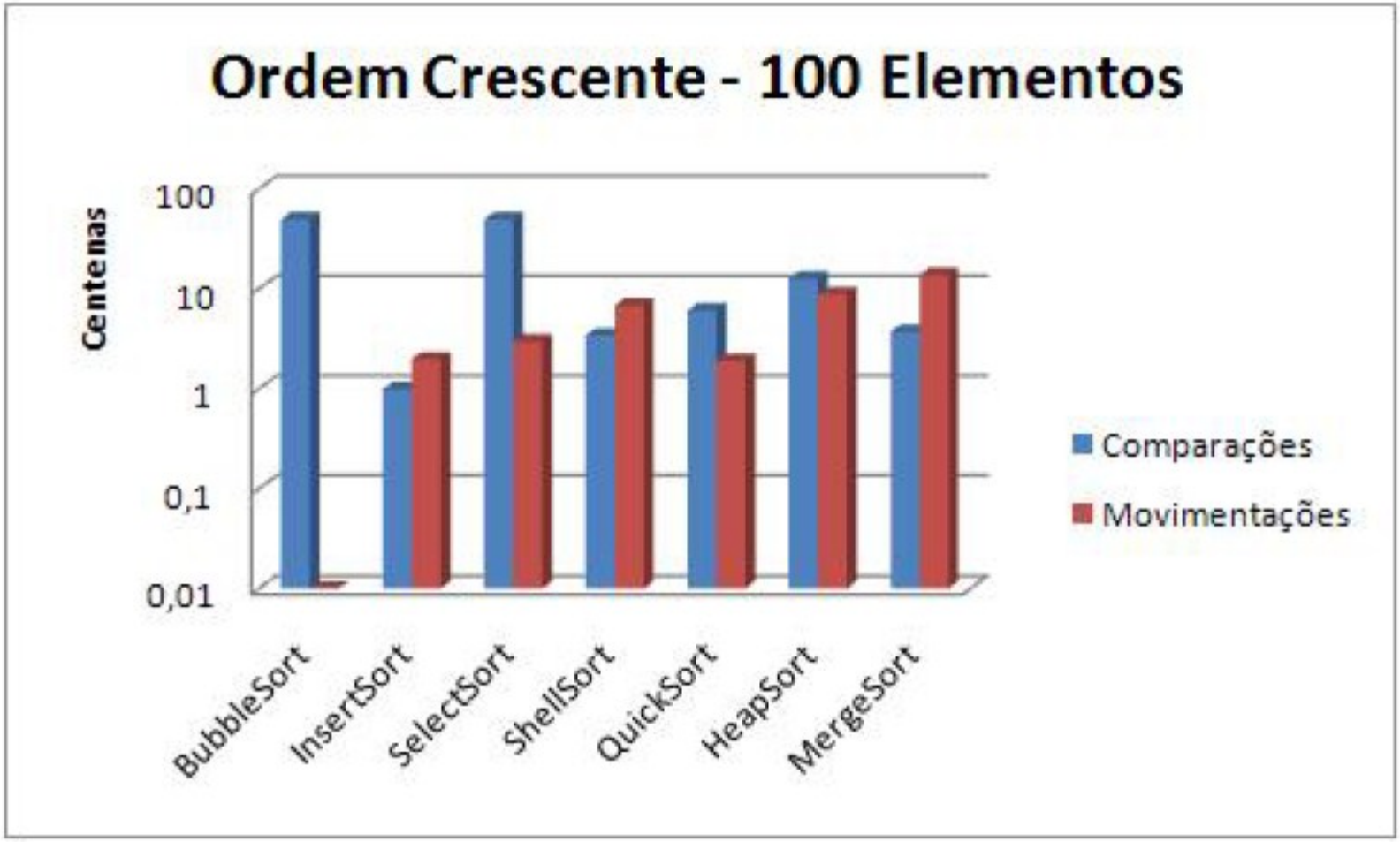
Fonte: http://www.youtube.com/watch?v=jT_yalDN4qI

1. Vetor ordenado em ordem crescente

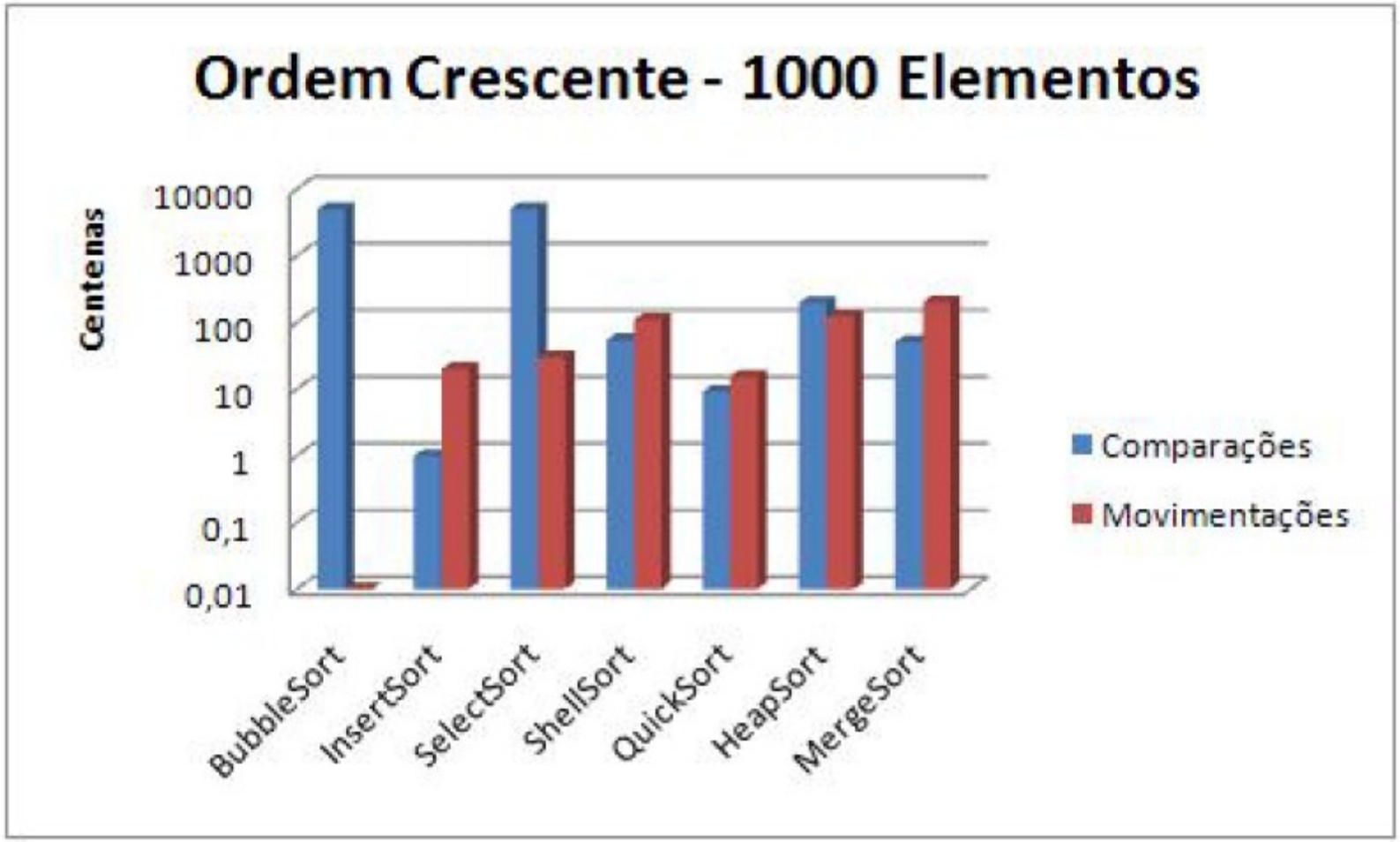
Vetor Ordenado em Ordem Crescente								
	100		1000		10000		100000	
	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.
1	4950	4950	499500	499500	49995000	49995000	4999950000	4999950000
2	99	198	99	1998	9999	19998	99999	199998
3	4950	297	499500	2997	49995000	29997	4999950000	299997
4	342	684	5457	10914	75243	150486	967146	1934292
5	606	189	909	1533	125439	17712	1600016	196605
6	1265	884	19562	12192	264433	156928	3312482	1900842
7	372	1376	5052	19968	71712	272640	877968	3385984

Tabela 1.1: Quantidade de comparações e movimentos dos testes no vetor

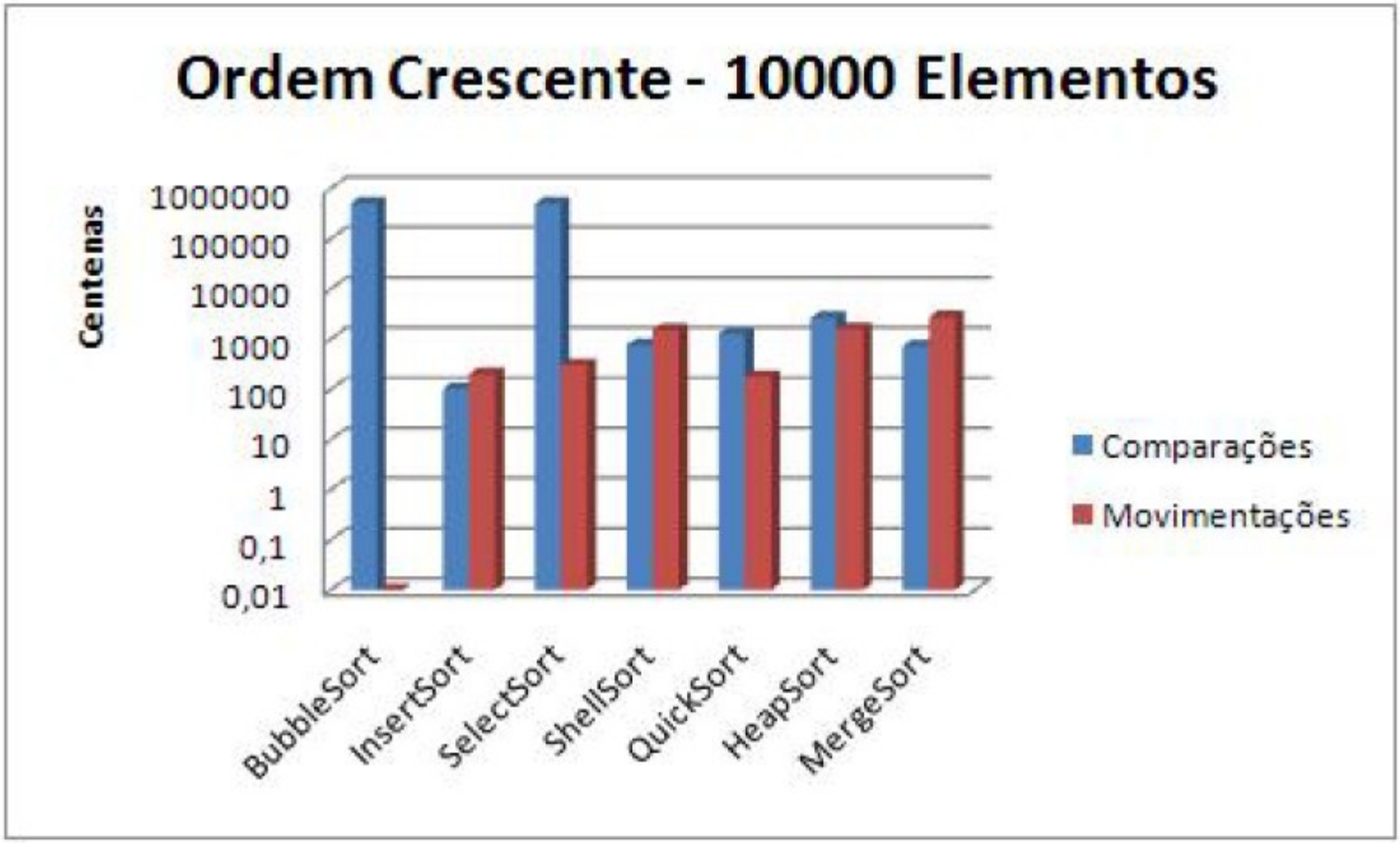
1. Vetor ordenado em ordem crescente



1. Vetor ordenado em ordem crescente



1. Vetor ordenado em ordem crescente



1. Vetor ordenado em ordem crescente

Vetor Ordenado em Ordem Crescente				
	100	1000	10000	100000
1	0,000050	0,004822	0,483164	48,373736
2	0,000003	0,00002	0,000193	0,001936
3	0,000050	0,004834	0,561589	58,300695
4	0,000008	0,000108	0,001453	0,018899
5	0,000020	0,000191	0,002403	0,028667
6	0,000033	0,000412	0,008299	0,083919
7	0,000150	0,001487	0,015855	0,205009

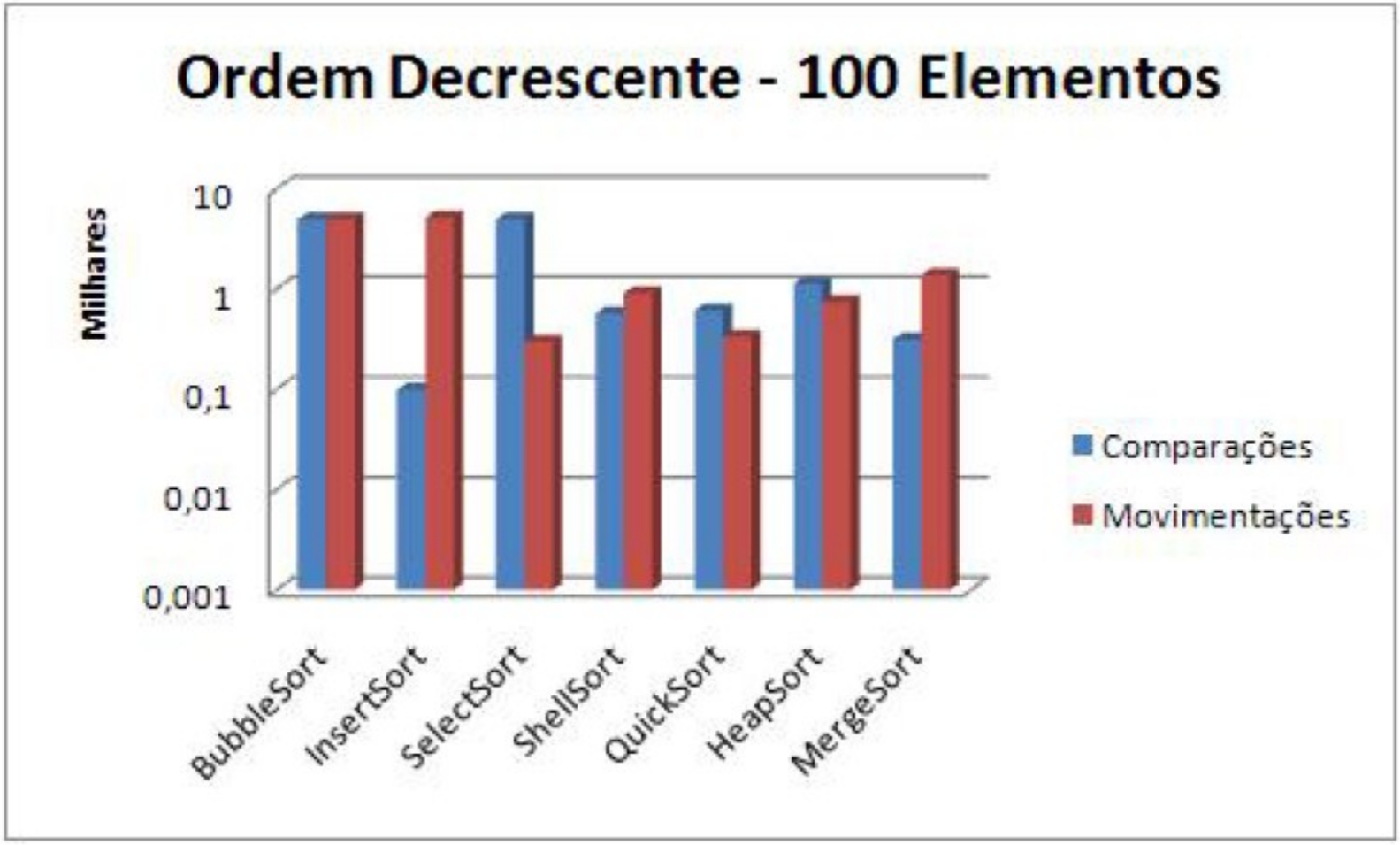
Tabela 1.2: Tempo gasto pelos testes no vetor

2. Vetor ordenado em ordem decrescente

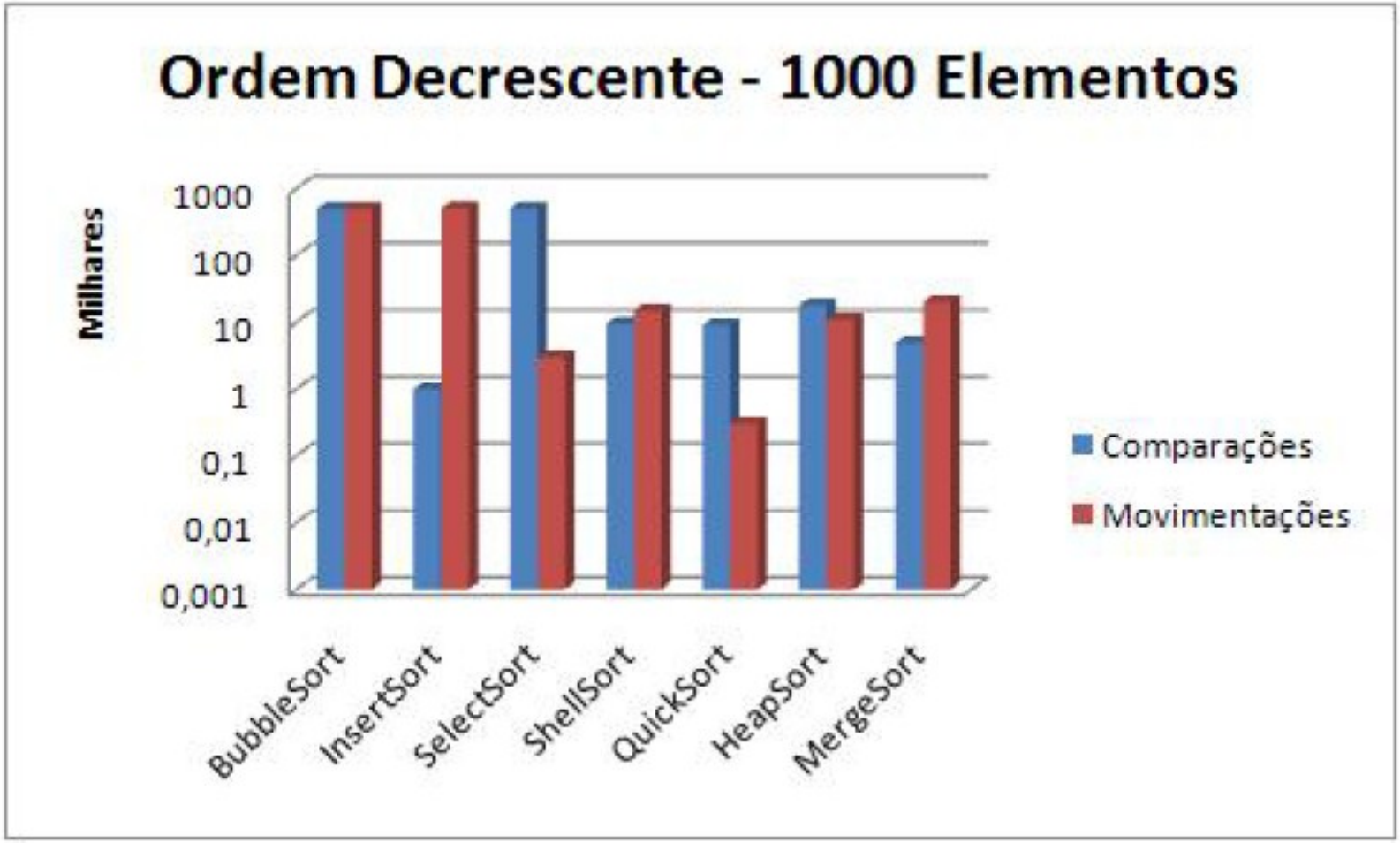
Vetor Ordenado em Ordem Decrescente								
	100		1000		10000		100000	
	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.
1	4950	4950	499500	499500	4999500	4999500	4999950000	4999950000
2	99	5148	999	501498	9999	50014998	9999	500014999
3	4950	297	499500	2997	49995000	29997	4999950000	299997
4	572	914	9377	14834	128947	204190	1586800	2553946
5	610	336	9016	303	125452	32712	1600030	346602
6	1125	747	17952	10811	246705	141975	3131748	1747201
7	316	1376	4932	19968	64608	272640	815024	3385984

Tabela 1.1: Quantidade de comparações e movimentos dos testes no vetor

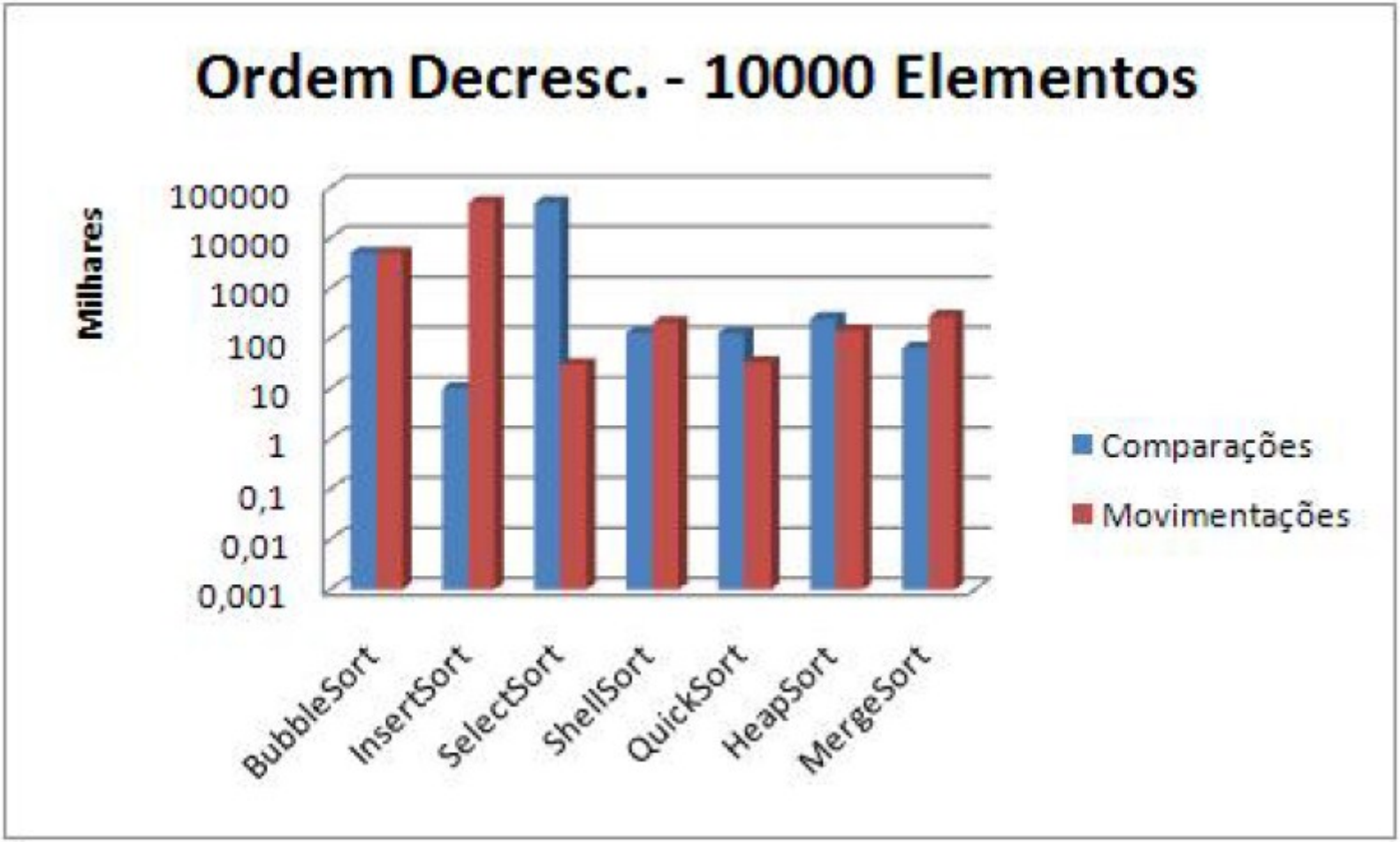
2. Vetor ordenado em ordem decrescente



2. Vetor ordenado em ordem decrescente



2. Vetor ordenado em ordem decrescente



2. Vetor ordenado em ordem decrescente

Vetor Ordenado em Ordem Decrescente				
	100	**1000**	**10000**	**100000**
1	0,000077	0,007503	0,752961	79,834753
2	0,000056	0,004897	0,667281	64,038128
3	0,000050	0,004834	0,561589	58,300695
4	0,000011	0,000164	0,002120	0,037005
5	0,000022	0,000190	0,002531	0,029430
6	0,000032	0,000381	0,007504	0,084207
7	0,000141	0,001495	0,016313	0,219894

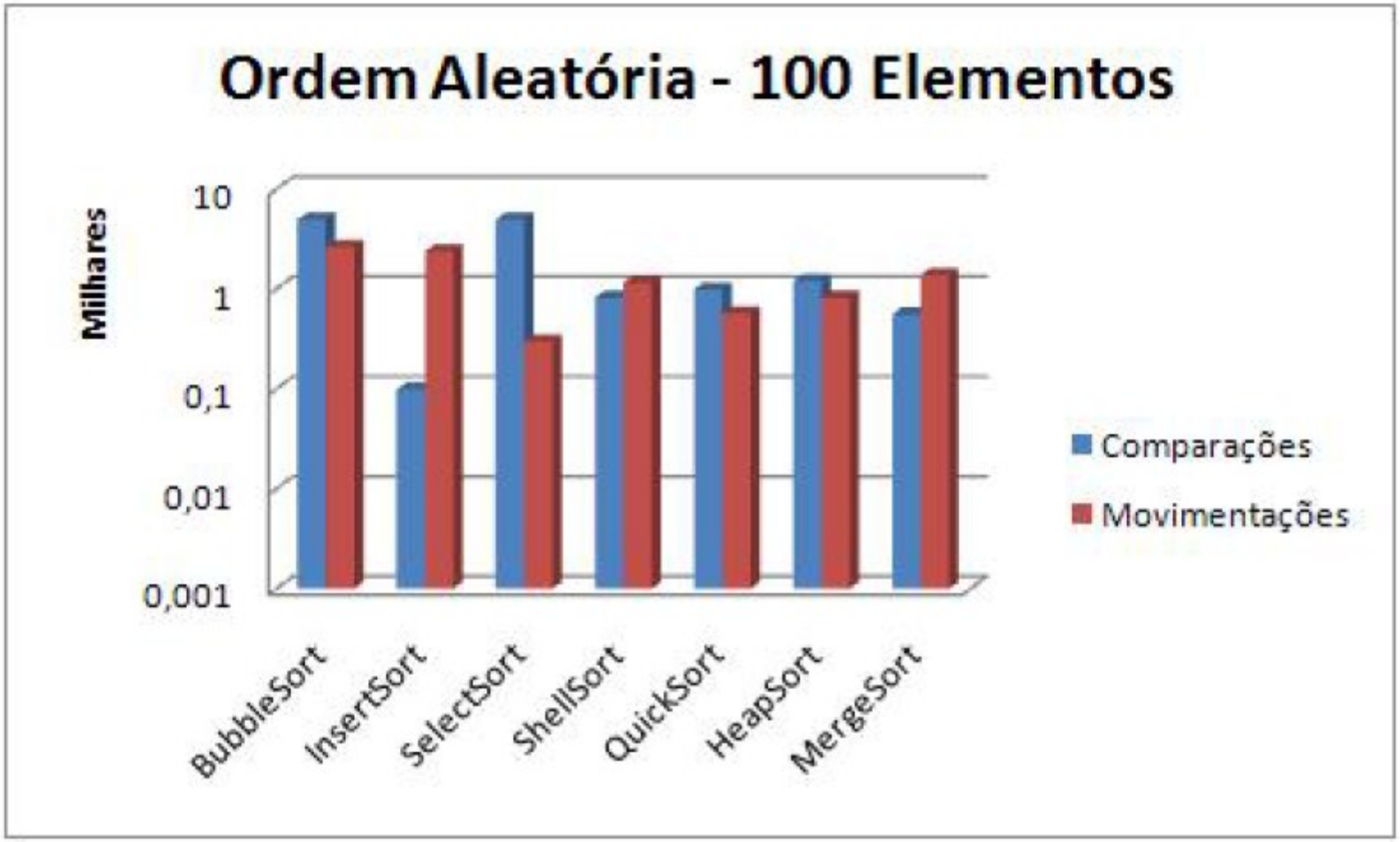
Tabela 1.1: Tempo gasto pelos testes no vetor

3. Vetor aleatório

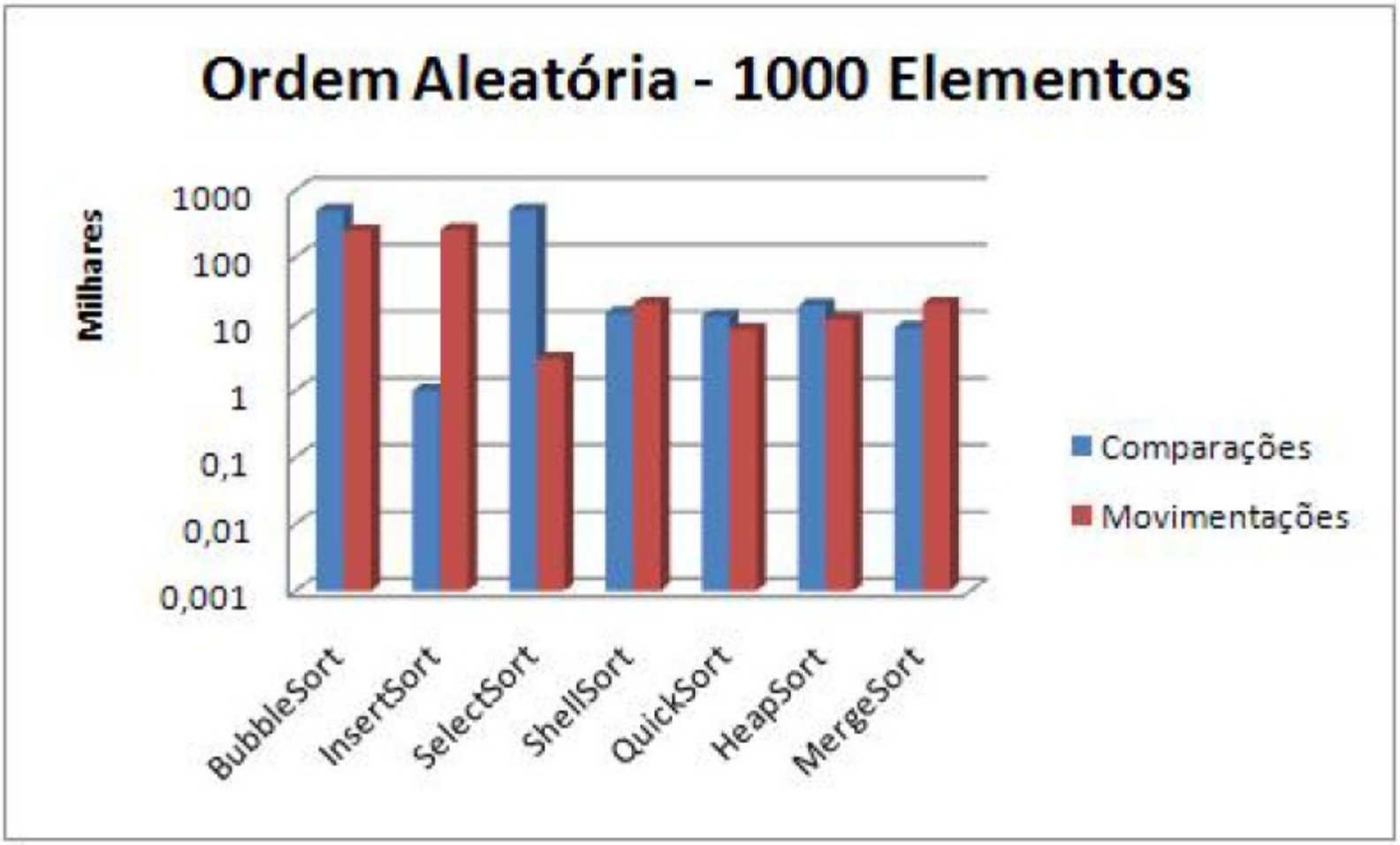
Vetor Aleatório								
	100		1000		10000		100000	
	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.	Comp.	Mov.
1	4950	2628	499500	242827	49995000	25160491	4999950000	2499136980
2	99	2387	999	253364	9999	25012754	99999	2500402956
3	4950	297	499500	2997	49995000	29997	4999950000	299997
4	818	1160	14364	19821	236735	311978	3711435	4670697
5	997	570	12852	8136	181203	103575	2114943	1310586
6	1205	817	18837	11556	255288	149150	3220006	1825075
7	558	1376	8744	19968	123685	272640	1566749	3385984

Tabela 1.1: Quantidade de comparações e movimentos dos testes no vetor

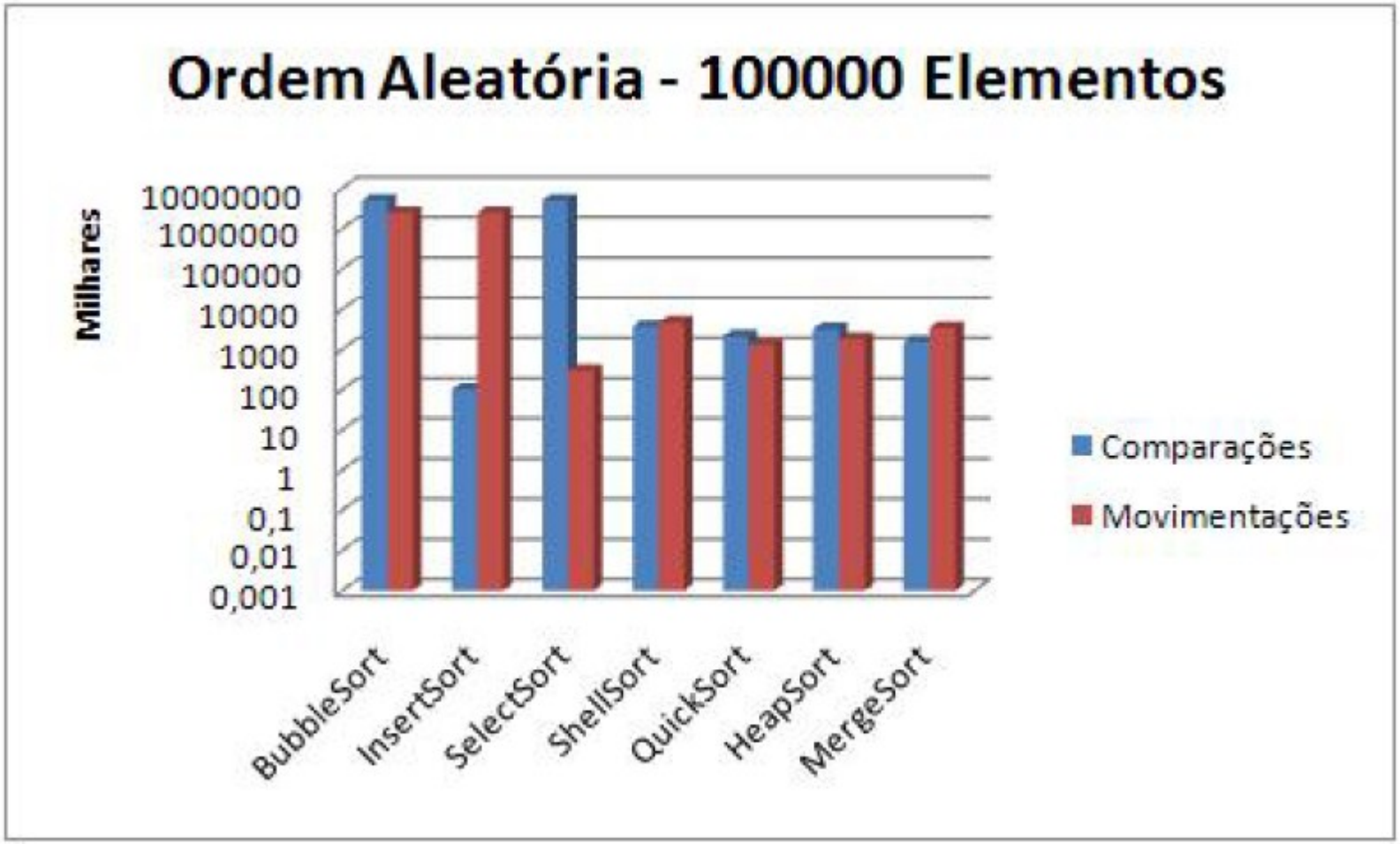
3. Vetor aleatório



3. Vetor aleatório



3. Vetor aleatório



3. Vetor aleatório

Vetor Aleatório				
	100	1000	10000	100000
1	0,000076	0,008084	0,858684	114,840058
2	0,000027	0,002531	0,323982	31,500937
3	0,000050	0,004834	0,561589	58,300695
4	0,000016	0,000290	0,006158	0,104286
5	0,000031	0,000403	0,004987	0,084427
6	0,000034	0,000420	0,009227	0,087964
7	0,00015	0,001598	0,019384	0,231564

Tabela 1.1: Tempo gasto pelos testes no vetor

Referências

- Vantagens e desvantagens dos algoritmos apresentados e dados dos slides 36 ao 50 foram retirados do seguinte trabalho:
  - NAZARÉ JÚNIOR, Antonio Carlos e GOMES, David Menotti. Algoritmos e Estruturas de Dados: Métodos de Ordenação Interna. Ouro Preto : UFOP, 2008.