PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Graduação em Sistemas de Informação

Fernando Bretz

Igor Palhares

Paulo Henrique

TRABALHO DE ALGORITMO E ESTRUTURA DE DADOS – MÉTODOS DE ORDENAÇÃO

Prof. Eveline Veloso

Belo Horizonte

2020

Sumário

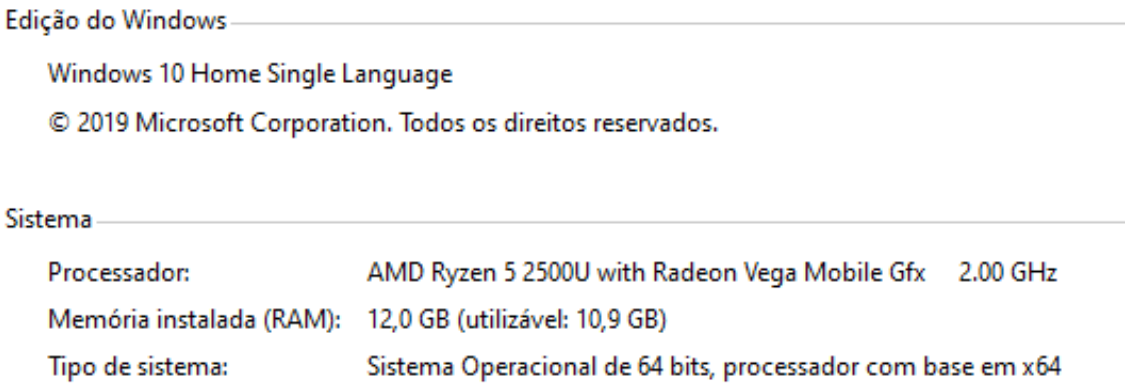
1. Introdução

Neste trabalhamos apresentaremos os métodos de ordenação: bolha, mergesort, heapsort, inserção, seleção e quicksort.

Vamos analisar a performance de todos os testes em relação a vetores de diversos tamanhos retirados de arquivos, após a análise da performance fizemos média de tempo de execução de todos os métodos e criamos gráficos para comparação entre eles, além de apresentarmos os códigos utilizados para testar os métodos.

Ferramentas utilizadas

Utilizamos a linguagem Java para execução dos métodos, a máquina utilizada para executar o programa e registar o tempo possui as seguintes configurações:



2. Desenvolvimento

Funcionamento do Método de bolha:

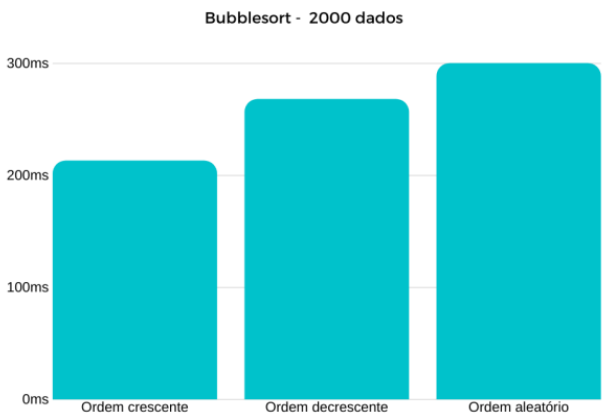
O Método de bolha, é um algoritmo de ordenação dos mais simples. A ideia é percorrer o vector diversas vezes, e a cada passagem fazer flutuar para o topo o maior elemento da sequência. Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

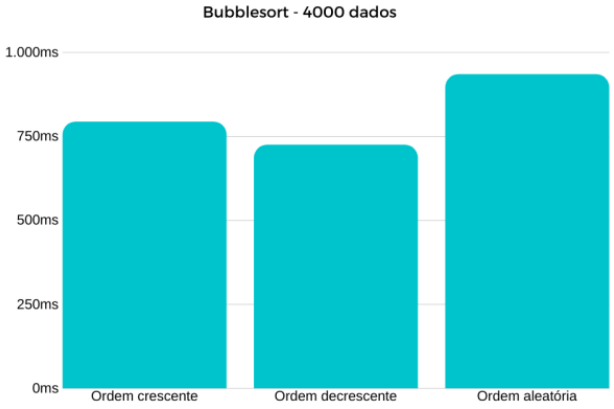
No melhor caso, o algoritmo executa n operações relevantes, onde n representa o número de elementos do vector. No pior caso, são feitas n^2 operações. A complexidade desse algoritmo é de ordem quadrática. Por isso, ele não é recomendado para programas que precisem de velocidade e operem com quantidade elevada de dados.

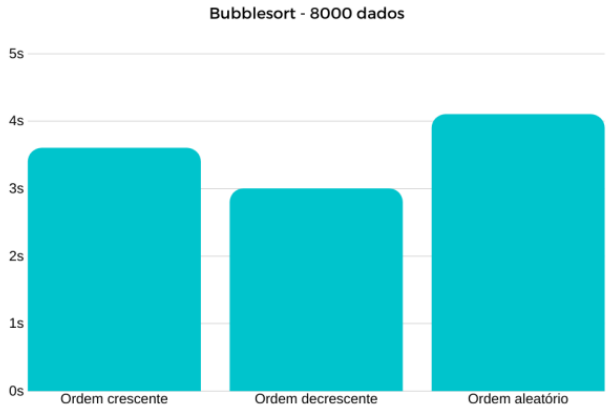
Código utilizado (crescente):

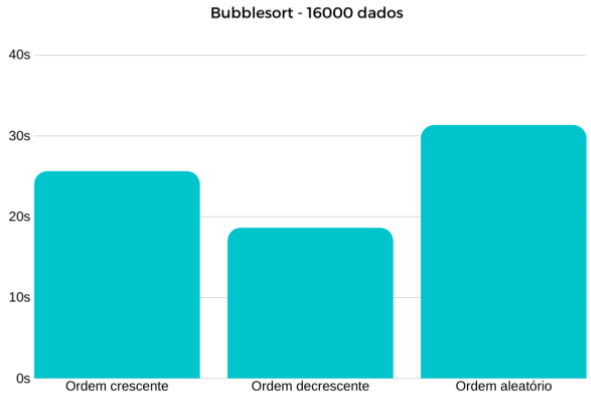
```
168     public static void sortBolhaCresc(Airbnb[] nome, int tam) {
169         Airbnb temp;
170 
171         for (int j = 1; j < tam - 1; j++) {
172             for(int i = j+1; i < tam;i ++) {
173                 if (Integer.parseInt(nome[j].getRoomID()) >= Integer.parseInt(nome[i].getRoomID())+1) {
174                     temp = nome[j];
175                     nome[j] = nome[i];
176                     nome[i] = temp;
177                 } // end if
178             } // end for
179         } // end for
180     } // end sortBolhaCresc
181 
```

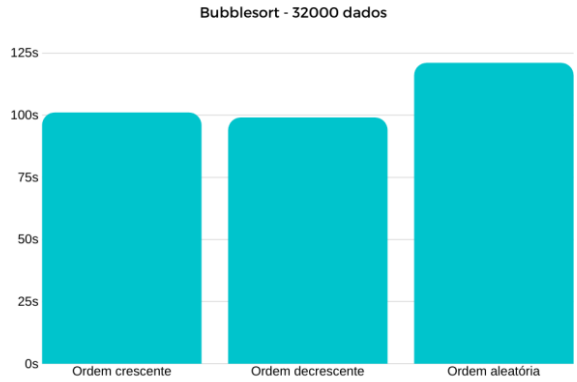
Analise de desempenho - Gráficos:

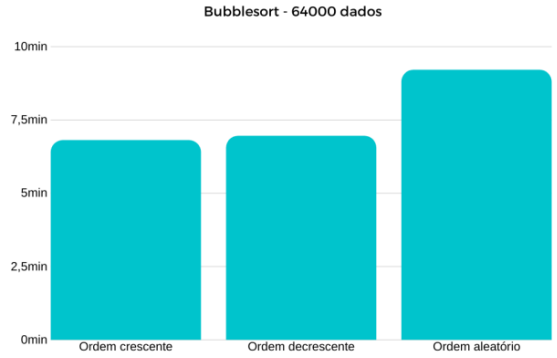


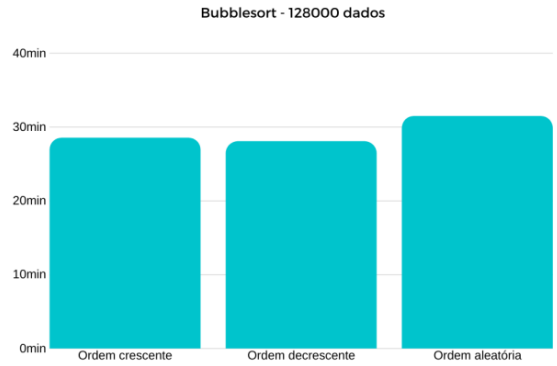

Bubblesort - 4000 dados
1.000ms
750ms
500ms
250ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatória


Bubblesort - 8000 dados
5s
4s
3s
2s
1s
0s
Ordem crescente
Ordem decrescente
Ordem aleatório


Bubblesort - 16000 dados
40s
30s
20s
10s
0s
Ordem crescente
Ordem decrescente
Ordem aleatório







Funcionamento do Método Heapsort:

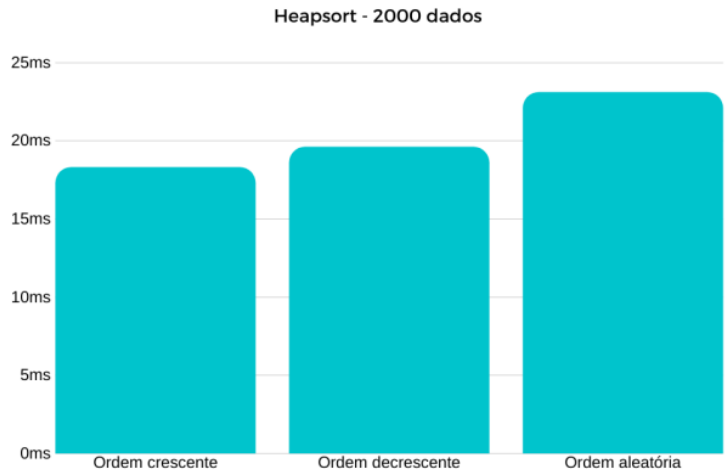
O heapsort utiliza uma estrutura de dados chamada heap, para ordenar os elementos à medida que os insere na estrutura. Assim, ao final das inserções, os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada, lembrando-se sempre de manter a propriedade de max-heap.

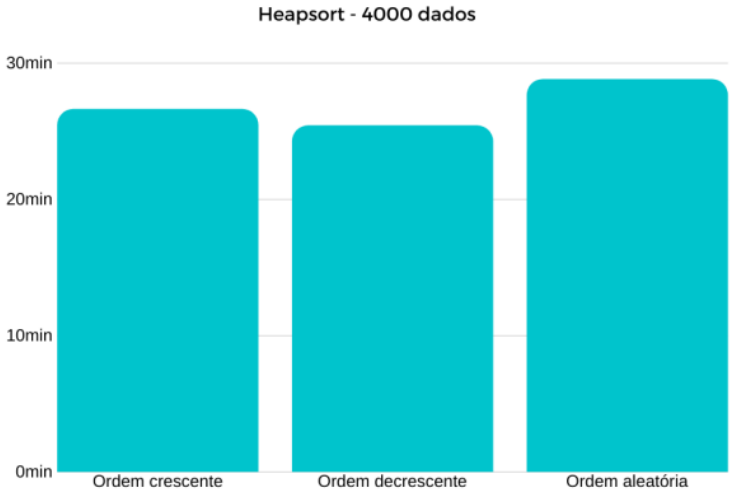
A heap pode ser representada como uma árvore ou como um vetor. Para uma ordenação decrescente, deve ser construída uma heap mínima (o menor elemento fica na raiz). Para uma ordenação crescente, deve ser construído uma heap máxima (o maior elemento fica na raiz).

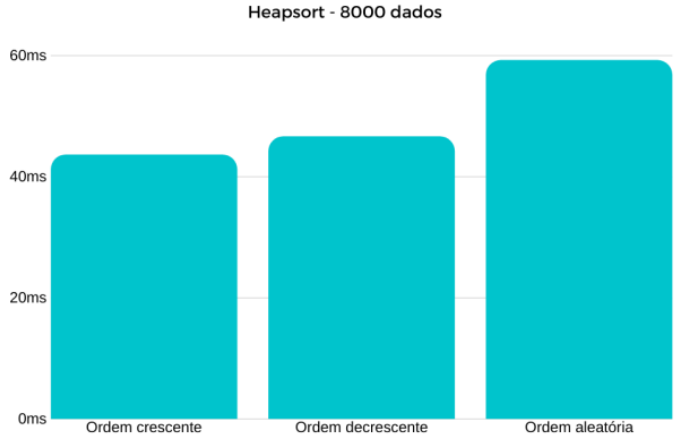
Código utilizado (crescente):

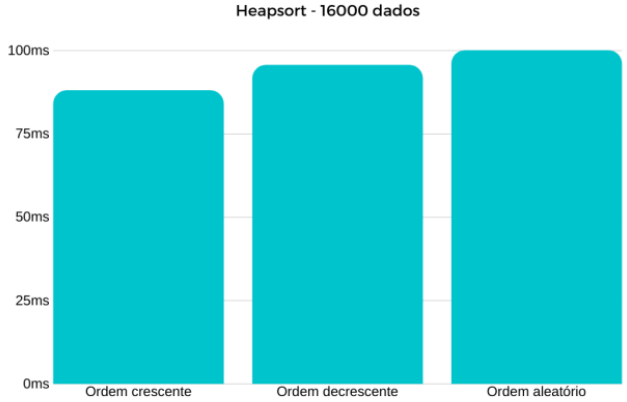
```
169     public static void heapCres (Airbnb arr[]) {
170         int n = arr.length;
171
172         // Build heap (rearrange array)
173         for (int i = n / 2 - 1; i >= 0; i--)
174                 heapify(arr, n, i);
175
176         // One by one extract an element from heap
177         for (int i = n - 1; i > 0; i--) {
178             // Move current root to end
179             Airbnb temp = arr[0];
180             arr[0] = arr[i];
181             arr[i] = temp;
182
183             // call max heapify on the reduced heap
184             heapify(arr, i, 0);
185         } // end for
186     } // end heapCres
```

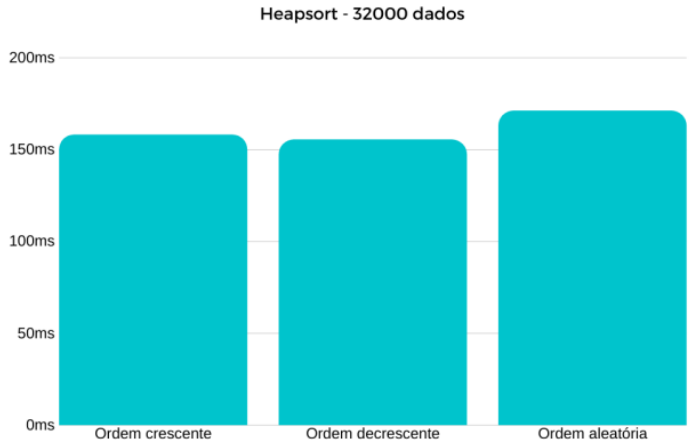
Analise de desempenho - Gráficos:

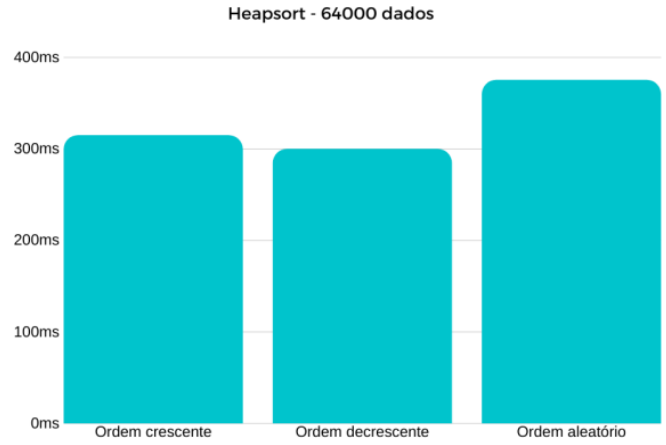


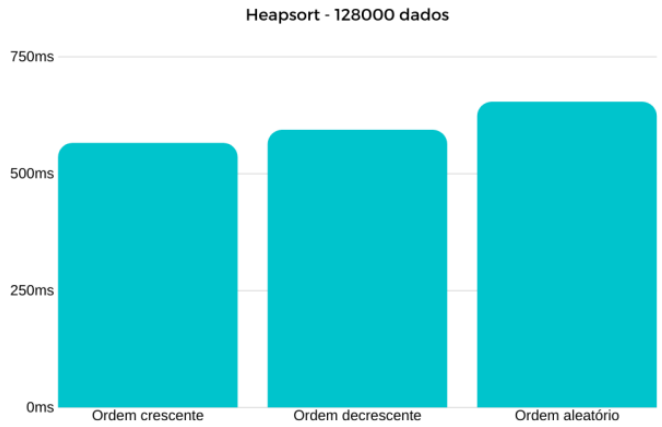

Heapsort - 4000 dados
30min
20min
10min
0min
Ordem crescente
Ordem decrescente
Ordem aleatória


Heapsort - 8000 dados
60ms
40ms
20ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatória


Heapsort - 16000 dados
100ms
75ms
50ms
25ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Heapsort - 32000 dados
200ms
150ms
100ms
50ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatória


Heapsort - 64000 dados
400ms
300ms
200ms
100ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Heapsort - 128000 dados
750ms
500ms
250ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório

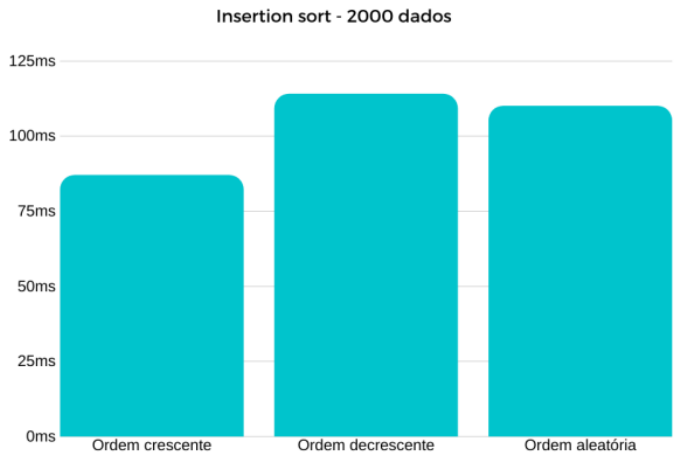
Funcionamento do Método de Inserção:

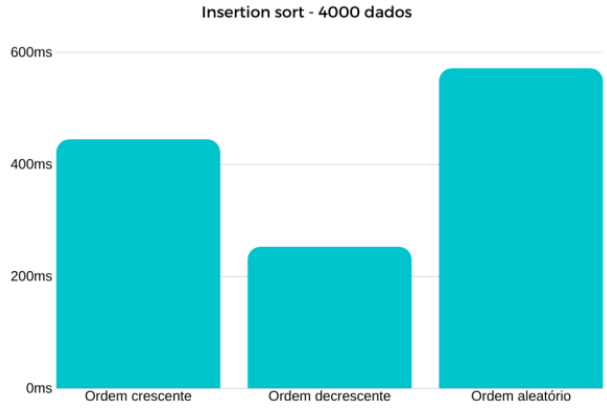
É um algoritmo de ordenação que, dado uma estrutura (array, lista) constrói uma matriz final com um elemento de cada vez, uma inserção por vez. Assim como algoritmos de ordenação quadrática, é bastante eficiente para problemas com pequenas entradas, sendo o mais eficiente entre os algoritmos desta ordem de classificação.

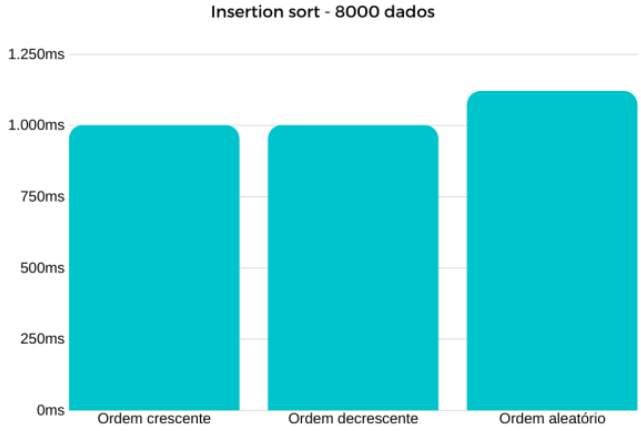
Código utilizado (crescente):

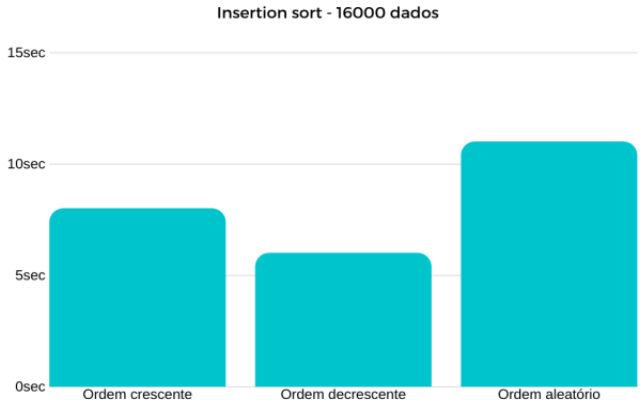
```
169     public static void InserCresc(Airbnb[] nome, int tam) {
170         Airbnb eleito;
171 
172         for (int i = 1; i < tam; i++) {
173             eleito = nome[i];
174             int j = i-1;
175             //laco q percorre elementos a esquerda do numero eleito
176             //ou ate encontrar a posicao para
177             //realocao do numero eleito
178             //rxing a ordenacao procurada
179 
180             while(j>=0 && Integer.parseInt(nome[j].roomID) >= Integer.parseInt(eleito.roomID)) {
181                 nome[j+1] = nome[j];
182                 j = j -1;
183             } // end while
184             nome[j+1] = eleito;
185         } // end for
186     } // InserCresc
```

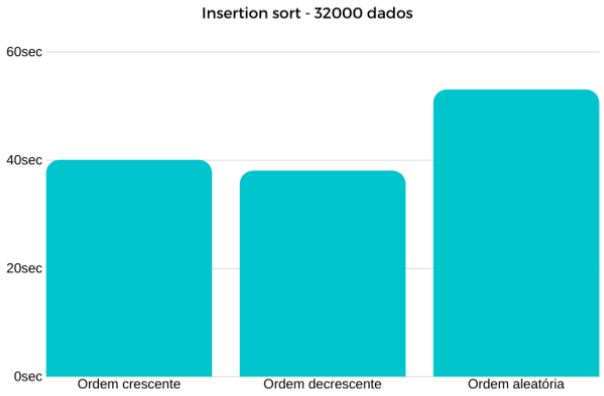
Analise de desempenho - Gráficos:

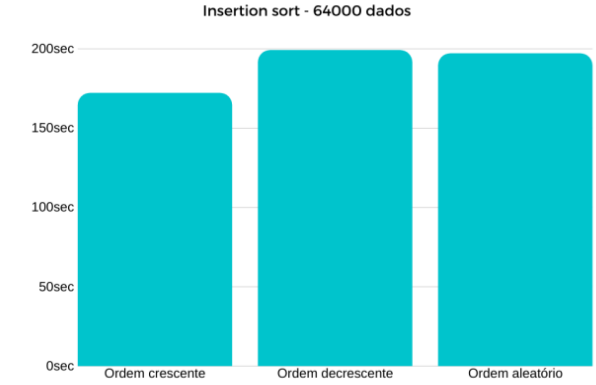


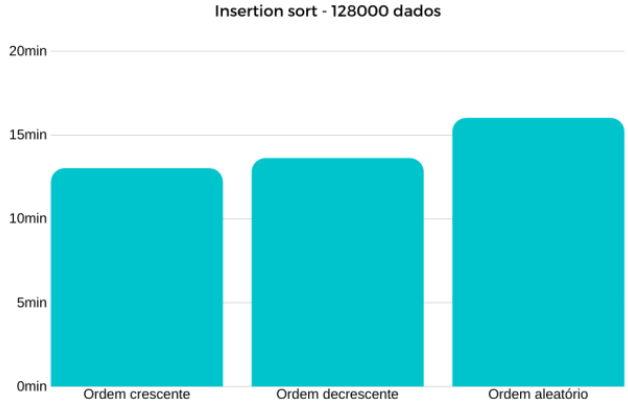

Insertion sort - 4000 dados
600ms
400ms
200ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Insertion sort - 8000 dados
1.250ms
1.000ms
750ms
500ms
250ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Insertion sort - 16000 dados
15sec
10sec
5sec
0sec
Ordem crescente
Ordem decrescente
Ordem aleatório







Funcionamento do Método de Mergesort:

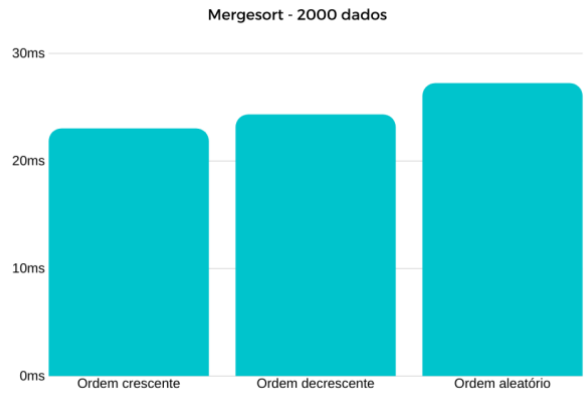
Sua ideia básica consiste em Dividir (o problema em vários subproblemas e resolver esses subproblemas através da recursividade) e conquistar (após todos os

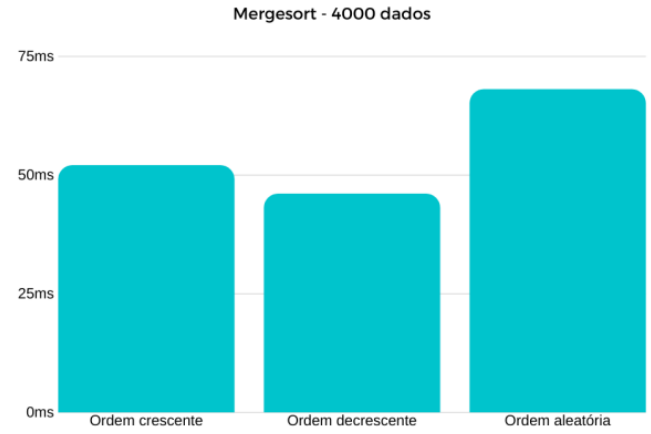
subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas). Como o algoritmo Mergesort usa a recursividade, há um alto consumo de memória e tempo de execução, tornando esta técnica não muito eficiente em alguns problemas.

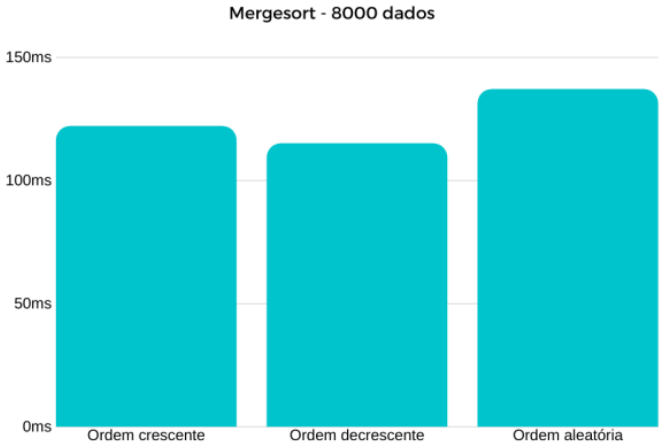
Código utilizado (crescente):

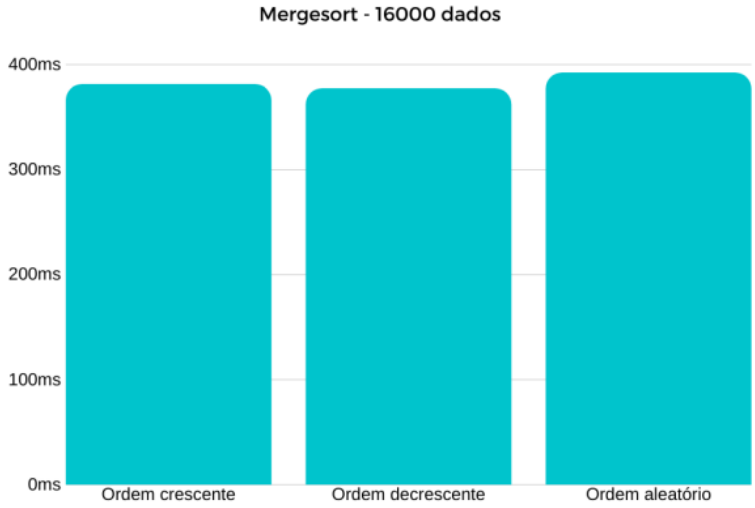
```
177  // M�todo IntercalaCres: � chamado pelo m�todo MergeCres que ordena em ordem Crescente
178
179      public static void IntercalaCres (Airbnb X[], int inicio, int fim, int meio) {
180          int posLivre, inicioVet1, inicioVet2, i;
181          Airbnb aux[] = new Airbnb[X.length];
182          inicioVet1 = inicio;
183          inicioVet2 = meio + 1;
184          posLivre = inicio;
185
186          while (inicioVet1 <= meio && inicioVet2 <= fim) {
187              if (Integer.parseInt(X[inicioVet1].getRoomID()) <= Integer.parseInt(X[inicioVet2].getRoomID())) {
188                  aux[posLivre] = X[inicioVet1];
189                  inicioVet1 = inicioVet1+1;
190              } else {
191                  aux[posLivre] = X[inicioVet2];
192                  inicioVet2 = inicioVet2+1;
193                  } // end else
194              posLivre = posLivre+1;
195          } // end while
196
197          // se ainda existirem numeros no primeiro vetor
198          // q n foram intercalados:
199          for (i = inicioVet1; i <= meio; i++) {
200              aux[posLivre] = X[i];
201              posLivre = posLivre+1;
202          } // end for
203
204          // se ainda existirem numeros no SEGUNDO vetor
205          // q n foram intercalados:
206          for (i = inicioVet2; i <= fim; i++) {
207              aux[posLivre] = X[i];
208              posLivre = posLivre+1;
209          } // end for
210
211          // retorna valores do vetor aux para vetor X
212          for (i = inicio; i <= fim; i++) {
213              X[i] = aux[i];
214          } // end for
215      } // end IntercalaCres
216
```

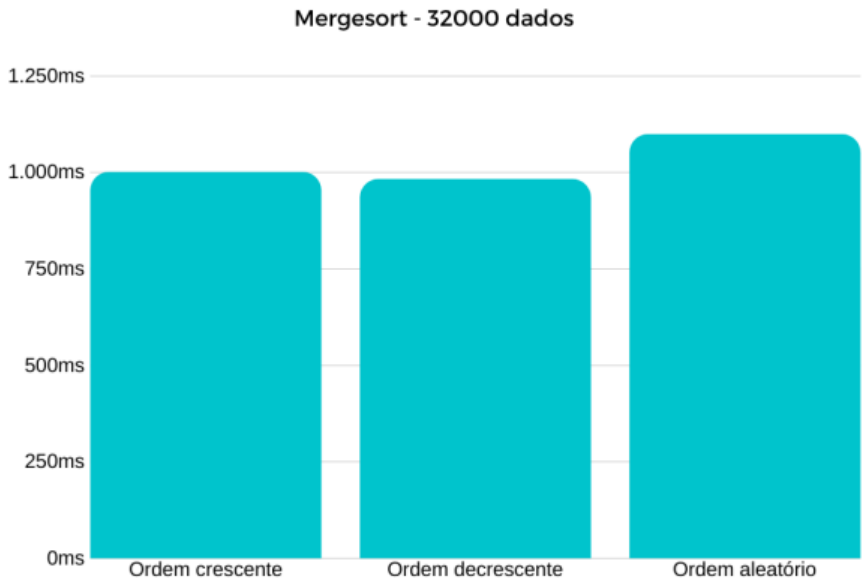
Analise de desempenho - Gráficos:

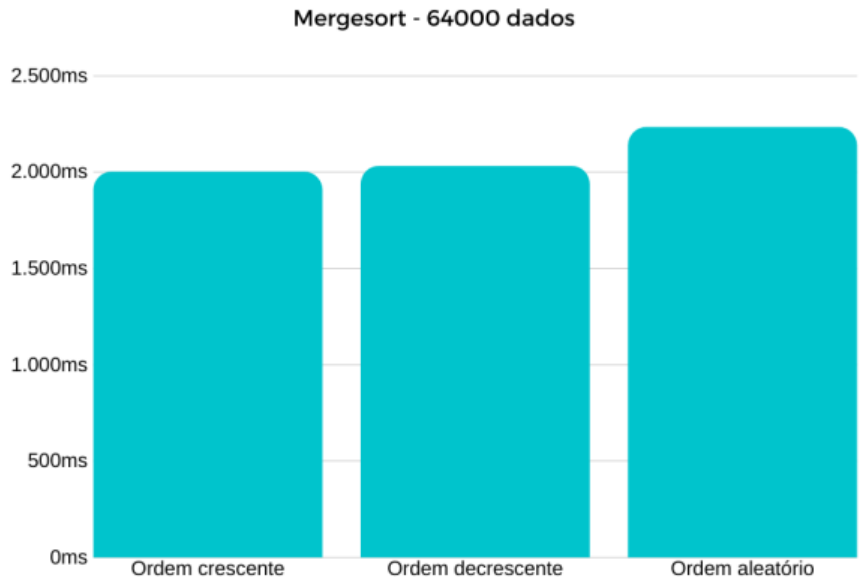


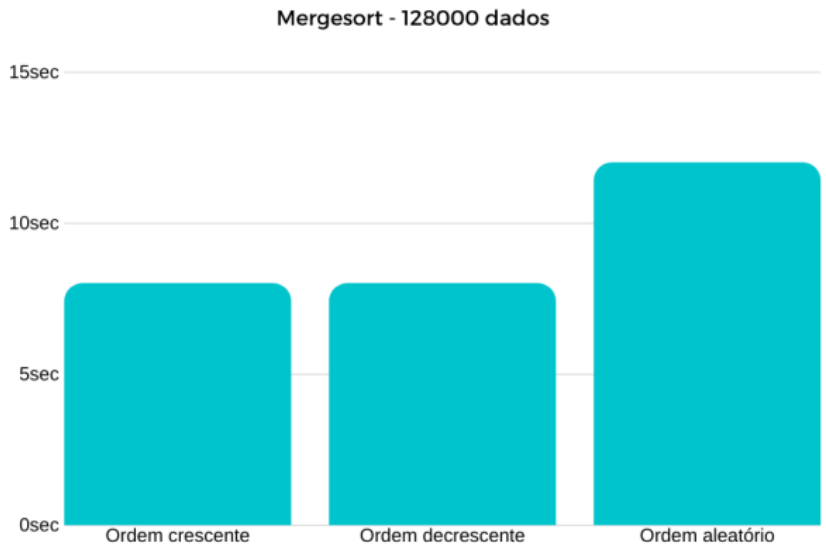

Mergesort - 4000 dados
75ms
50ms
25ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatória


Mergesort - 8000 dados
150ms
100ms
50ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatória


Mergesort - 16000 dados
400ms
300ms
200ms
100ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Mergesort - 32000 dados
1.250ms
1.000ms
750ms
500ms
250ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Mergesort - 64000 dados
2.500ms
2.000ms
1.500ms
1.000ms
500ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório



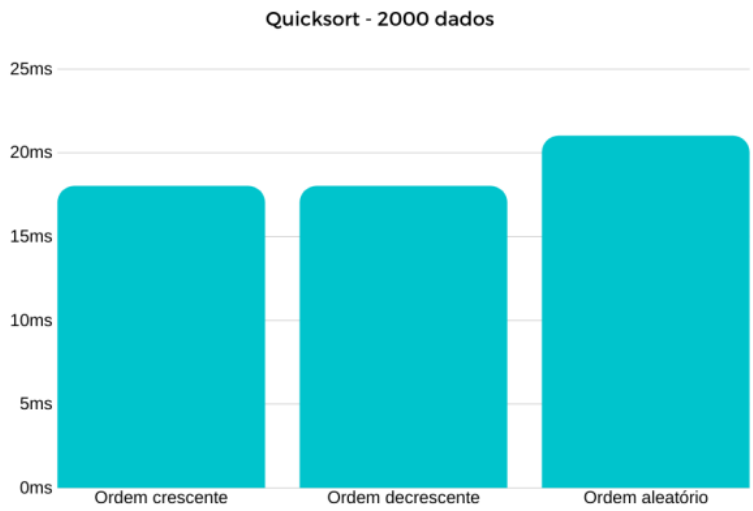
Funcionamento do Método de Quicksort:

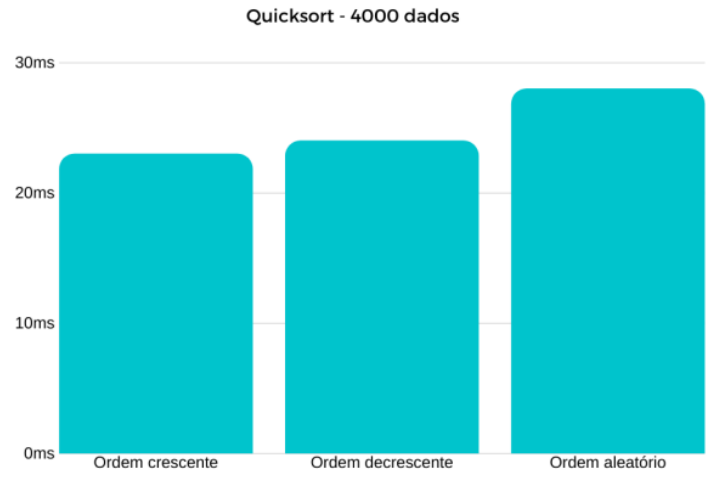
O quicksort adota a estratégia de divisão e conquista. A estratégia consiste em rearranjar as chaves de modo que as chaves "menores" precedam as chaves "maiores". Em seguida o quicksort ordena as duas sublistas de chaves menores e maiores recursivamente até que a lista completa se encontre ordenada.

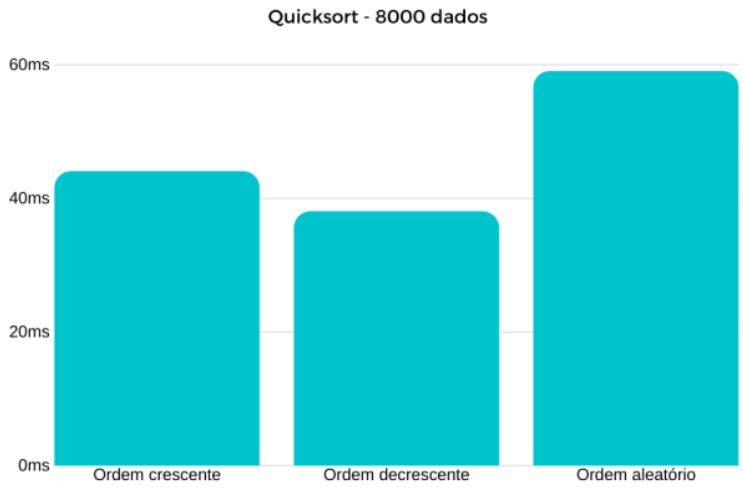
Código utilizado (crescente):

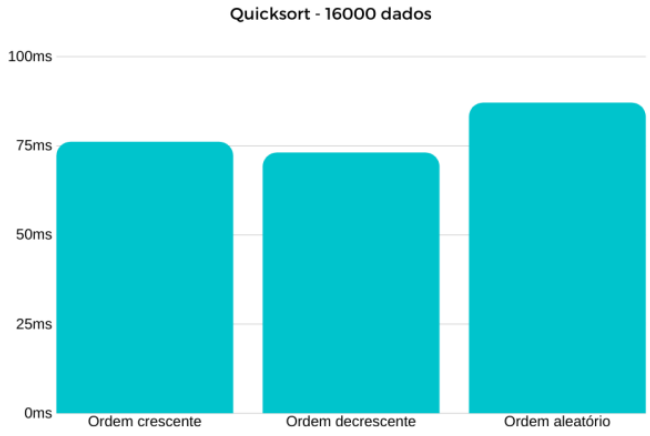
```java
    public static void quickSort (Airbnb X[],int p, int r) {
        int q;

        if (p < r) {
            q = particao(X,p,r);
            quickSort(X,p,q);
            quickSort(X,q+1,r);
        } // end if
    } // end quickSort

// M�todo particao: � chamado pelo m�todo quickSort que ordena em ordem Crescente

    public static int particao (Airbnb X[], int p , int r) {
        int i,j;
        Airbnb pivo;
        pivo= X[(p+r)/2];
        i = p-1;
        j = r+1;

        while (i < j) {
            do {
                j = j - 1;
            } // end do

            while(Integer.parseInt(X[j].getRoomID()) > Integer.parseInt(pivo.getRoomID()));

            do {
                i=i+1;
            } // end do

            while (Integer.parseInt(X[i].getRoomID()) < Integer.parseInt(pivo.getRoomID()));
                if (i < j)
                    troca(X,i,j);
        } // end while

        return j;
    } // end particao

// M�todo troca: � chamado pelo m�todo particao que ordena em ordem Crescente

    public static void troca(Airbnb X[], int i ,int j) {
        Airbnb aux;
        aux = X[i];
        X[i] = X[j];
        X[j] = aux;
    } // end troca
```

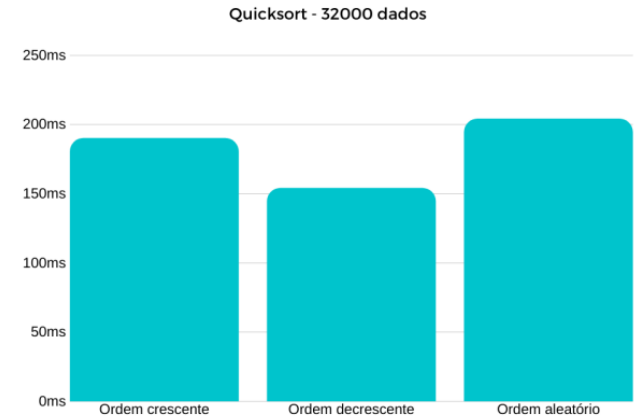
Analise de desempenho - Gráficos:

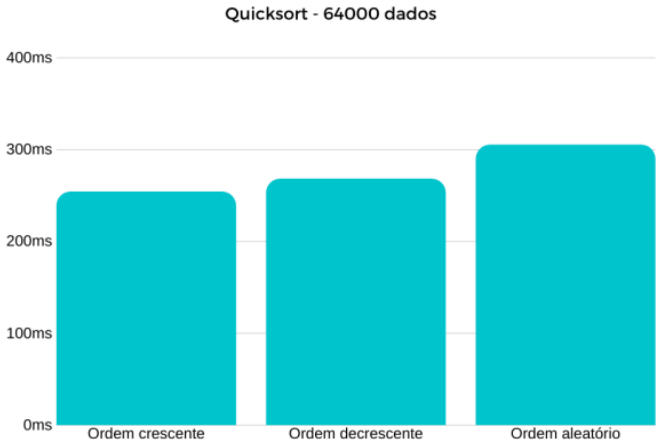


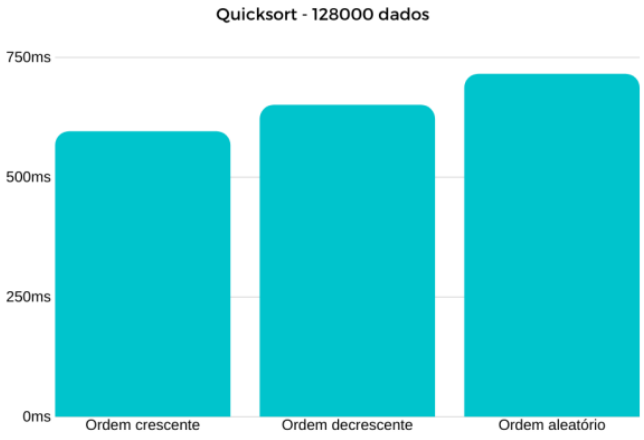

Quicksort - 4000 dados
30ms
20ms
10ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Quicksort - 8000 dados
60ms
40ms
20ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Quicksort - 16000 dados
100ms
75ms
50ms
25ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Quicksort - 32000 dados
250ms
200ms
150ms
100ms
50ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório


Quicksort - 64000 dados
400ms
300ms
200ms
100ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório

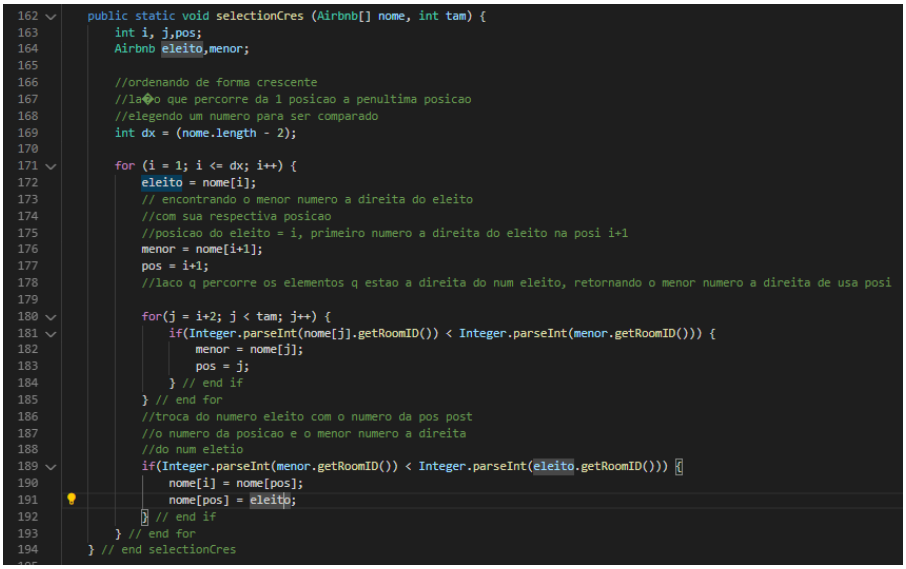

Quicksort - 128000 dados
750ms
500ms
250ms
0ms
Ordem crescente
Ordem decrescente
Ordem aleatório

Funcionamento do Método de Seleção:

A ideia é sempre procurar o menor elemento do vetor e inseri-lo no início do vetor. Procuramos o menor valor do vetor e colocamos ele em *vetor[1]*. Procuramos o menor valor do vetor excluindo o já colocado e colocamos ele em *vetor[2]*. E assim vamos indo até termos todo o vetor ordenado.

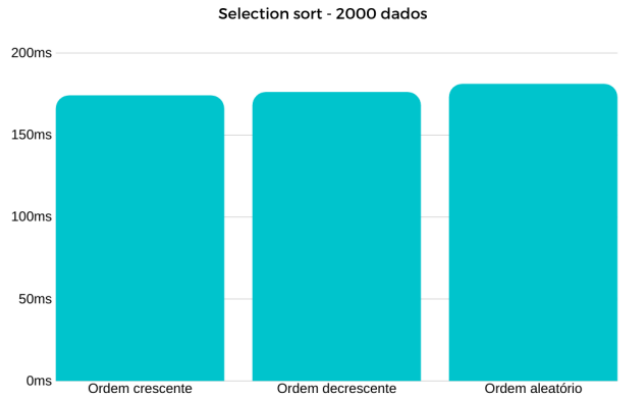
Partindo sempre a partir do último elemento reordenado (a partir do *i*), o programa procura o menor elemento no vetor e o substitui pelo elemento *i* atual.

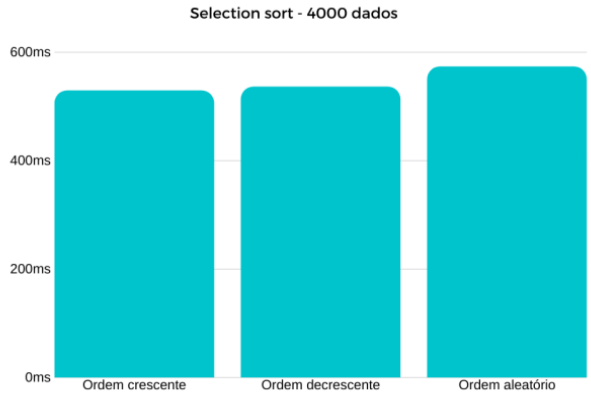
Código utilizado (crescente):

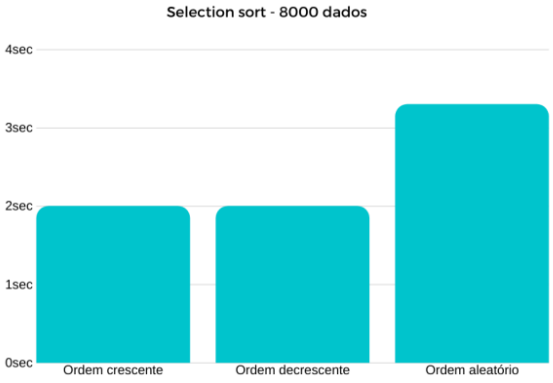


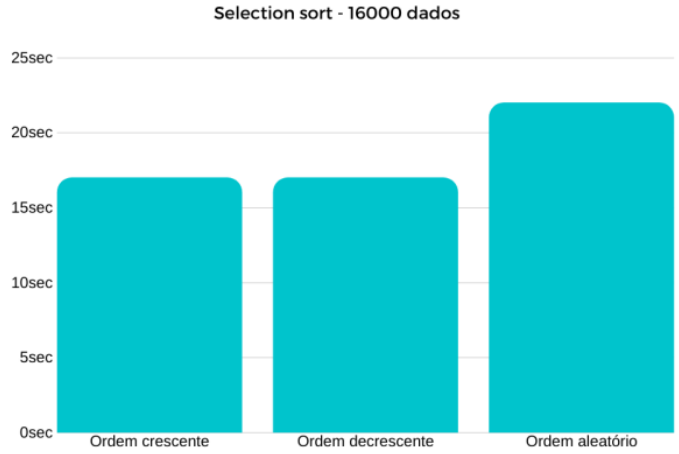
```
public static void selectionCres (Airbnb[] nome, int tam) {
    int i, j,pos;
    Airbnb eleito,menor;

    //ordenando de forma crescente
    //la�o que percorre da 1 posicao a penultima posicao
    //elegendo um numero para ser comparado
    int dx = (nome.length - 2);

    for (i = 1; i <= dx; i++) {
        eleito = nome[i];
        // encontrando o menor numero a direita do eleito
        //com sua respectiva posicao
        //posicao do eleito = i, primeiro numero a direita do eleito na posi i+1
        menor = nome[i+1];
        pos = i+1;
        //laco q percorre os elementos q estao a direita do num eleito, retornando o menor numero a direita de usa posi

        for(j = i+2; j < tam; j++) {
            if(Integer.parseInt(nome[j].getRoomID()) < Integer.parseInt(menor.getRoomID())) {
                menor = nome[j];
                pos = j;
            } // end if
        } // end for
        //troca do numero eleito com o numero da pos post
        //o numero da posicao e o menor numero a direita
        //do num eletio
        if(Integer.parseInt(menor.getRoomID()) < Integer.parseInt(eleito.getRoomID())) {
            nome[i] = nome[pos];
            nome[pos] = eleito;
        } // end if
    } // end for
} // end selectionCres
```

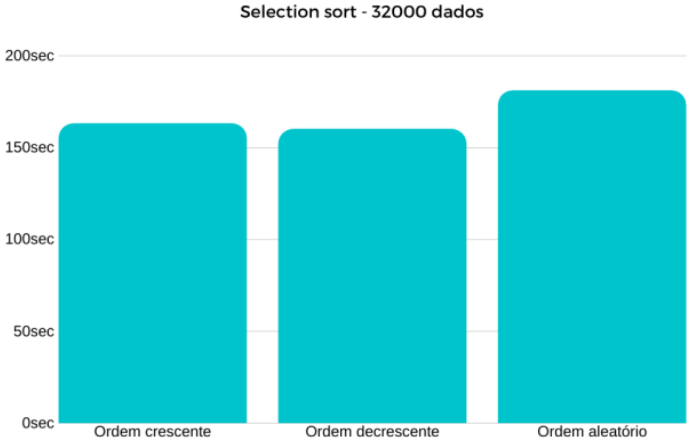
Analise de desempenho - Gráficos:

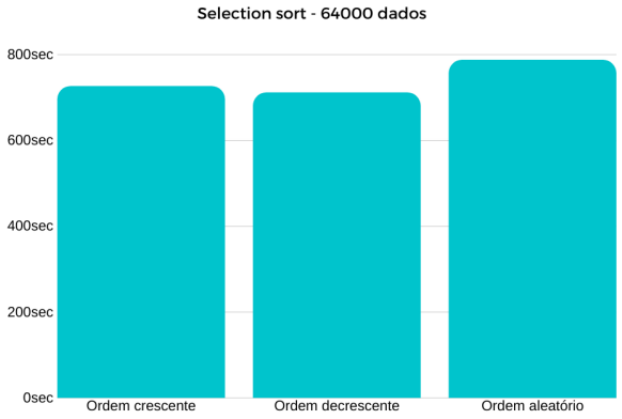


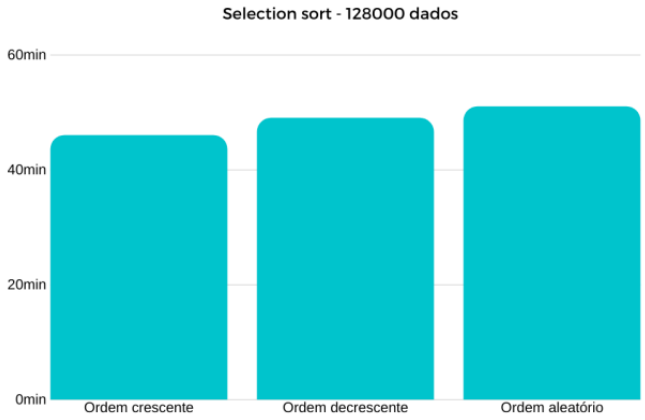





Selection sort - 16000 dados
25sec
20sec
15sec
10sec
5sec
0sec
Ordem crescente
Ordem decrescente
Ordem aleatório


Selection sort - 32000 dados
200sec
150sec
100sec
50sec
0sec
Ordem crescente
Ordem decrescente
Ordem aleatório


Selection sort - 64000 dados
800sec
600sec
400sec
200sec
0sec
Ordem crescente
Ordem decrescente
Ordem aleatório


Selection sort - 128000 dados
60min
40min
20min
0min
Ordem crescente
Ordem decrescente
Ordem aleatório

3. CONCLUSÃO

Considerando as medidas e análises realizadas neste trabalho, percebemos que para a criação dos programas, foram utilizados recursos práticos e teóricos desenvolvidos em salas de aula e em laboratório de algoritmo e estrutura de dados como como complexidade do algoritmo e os tipos de métodos de organização de dados.

Neste trabalho, foram criados programas utilizando os métodos de organização (inserção, seleção, bolha, mergesort, heapsort e quicksort), e foram usados para ordenar de forma crescente, decrescente e aleatória arquivos com 2.000, 4.000, 8.000, 16.000, 32.000, 64.000 e 128.000 dados.

Após a ordenação dos arquivos, os programas mostram o tempo de execução de cada método de organização e foram feitos gráficos para facilitar a comparação entre eles.

Analisando os resultados concluímos que o melhor método para organizar os arquivos neste caso foi o método de ordenação heapsort, pois o método atinge o objetivo de ordenar todos os dados do arquivo de forma crescente, decrescente e aleatória em menos tempo.

4. REFERÊNCIAS

Wikipédia**.** **Heapsort**. Disponivel em: <https://pt.wikipedia.org/wiki/Heapsort/> Acesso em: 21 abr. 2020.

Wikipédia**.** **Bubble Sort**. Disponivel em: <https://pt.wikipedia.org/wiki/Bubble_sortHYPERLINK "https://www.canalti.com.br/sistemas-operacionais/como-funciona-um-processador-cpu-uc-ula-registradores/"/> Acesso em: 21 abr. 2020.

Wikipédia**. Insertion sort**. Disponivel em: <https://pt.wikipedia.org/wiki/Insertion_sort/> Acesso em: 21 abr. 2020.

Wikipédia**.** **Merge Sort**. Disponivel em: <https://pt.wikipedia.org/wiki/Merge_sortHYPERLINK "https://www.canalti.com.br/sistemas-operacionais/como-funciona-um-processador-cpu-uc-ula-registradores/"/> Acesso em: 21 abr. 2020.

THIAGO MADEIRA**.** **Ordenação por seleção**. Disponivel em: <https://tiagomadeira.com/2006/01/ordenacao-por-selecao//> Acesso em: 21 abr. 2020.