UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
DEPARTAMENTO DE ESTATÍSTICA E INFORMÁTICA
COORDENAÇÃO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALGORITMOS E ESTRITURA DE DADOS

EXERCÍCIO PROPOSTO 1

ALGORITMOS DE ORDENAÇÃO - SORTING

ALUNO
FÁBIO ALVES DE FREITAS

RECIFE
15/06/2017

ÍNDICE

1. Introdução

Na computação em determinados problemas é necessário a utilização dos Arrays, porém quando seus elementos não estão ordenados da maneira que facilite nosso trabalho é necessária a utilização dos algoritmos de ordenação, que os ordenam de maneira a simplificar nosso trabalho.

2. Discussão do exercício

Neste exercício proposto serão testados de forma empírica alguns algoritmos de ordenação para testar sua eficiente em diferentes situações. Provar que mesmo um algoritmo bem estruturado e de complexidade baixa (n*Log n) sempre será melhor em número de iterações que algoritmos de complexidade alta (n^2) a partir de um determinado valor.

3. Metodologia de Trabalho

Atividade	Descrição
1	Realizaram-se testes com diferentes Arrays de tamanho relativamente grande.
2	Estes Arrays foram gerados com tamanhos predefinidos e elementos gerados pseudo randomicamente.
3	Todos os algoritmos são testados com o mesmo array inicial para te ser um resultado mais preciso.
4	O esboço dos gráficos são da seguinte forma: O eixo da abscissa representa os array de diferentes tamanhos e o eixo da ordenada representa o número de iterações necessárias para sua ordenação com o respectivo algoritmos.

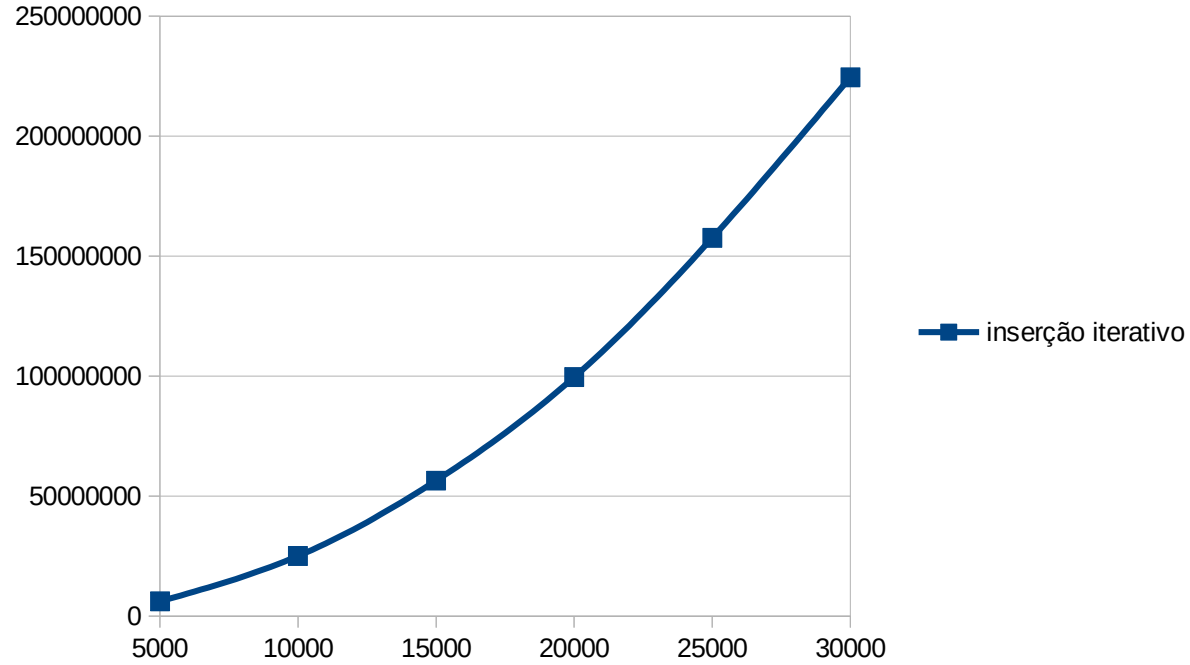
4. Conclusão

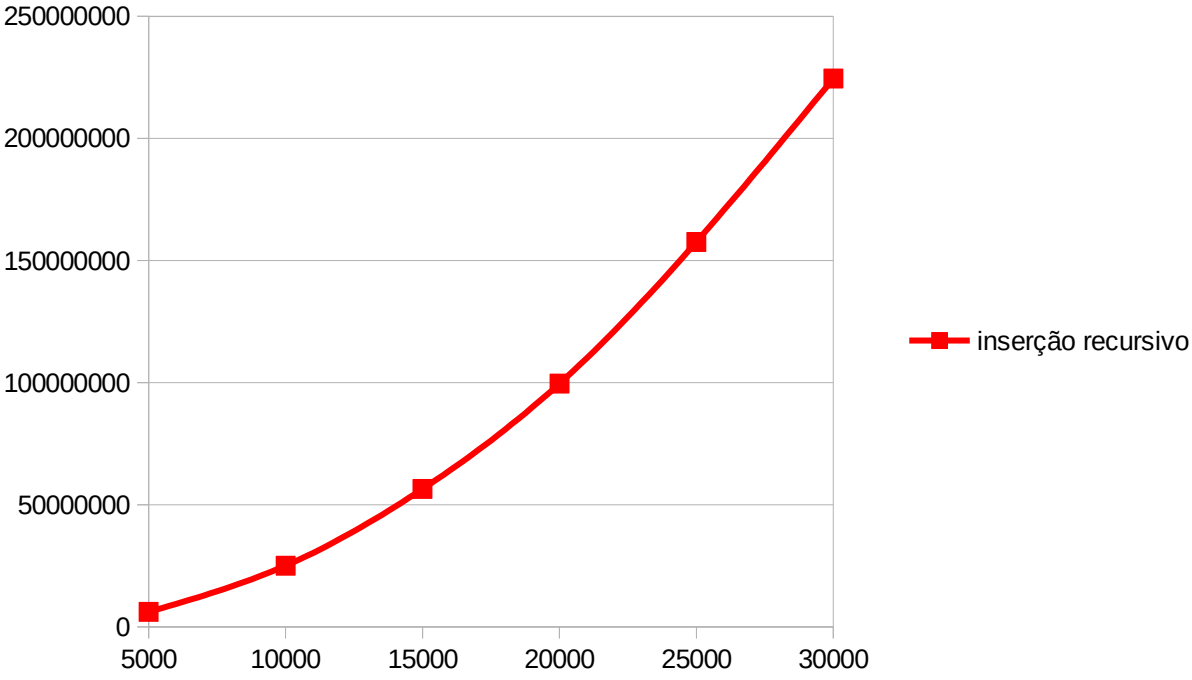
Algoritmos de complexidade n^2 foram eficientes apenas em arrays com tamanho menor que 50. Isto ocorreu por que a escrita destes algoritmos não demanda tantos comandos e por isso venceram.

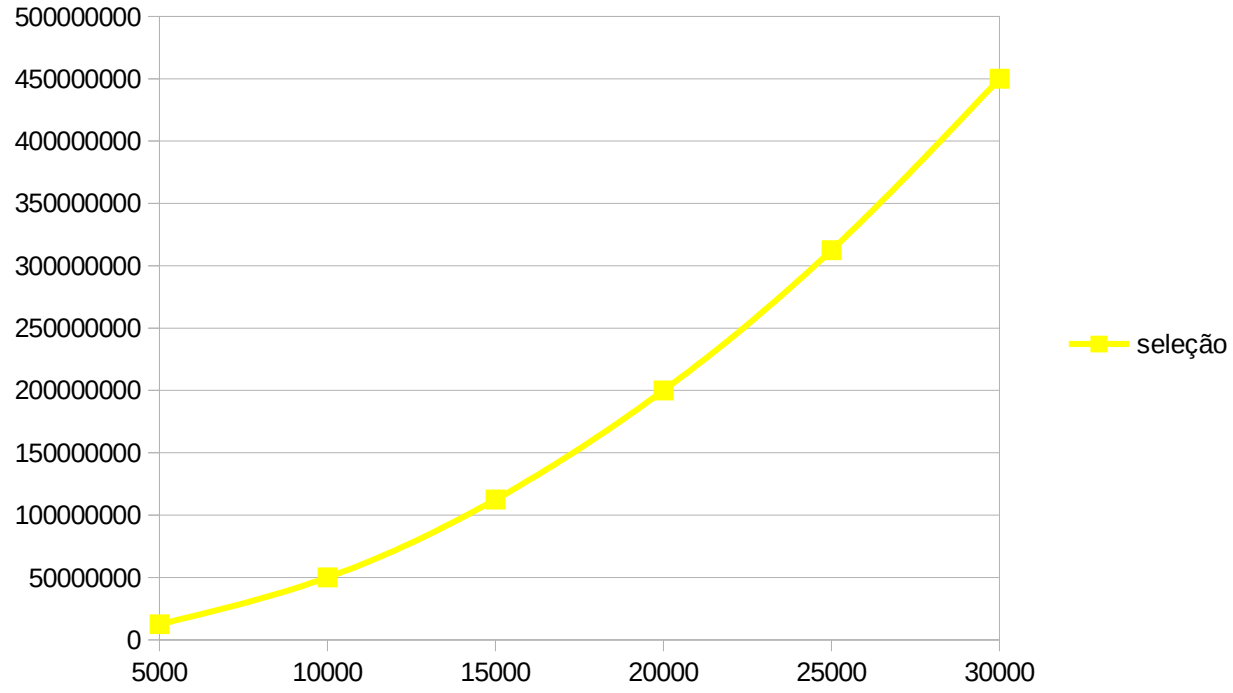
Quando o Array começou a passar de 50 os algoritmos de complexidade n^2 perdiam pelo fato de que mesmo o código deles ser menor a partir destes valores o número de iterações crescia exponencialmente enquanto que os algoritmos de complexidade n*Log n tinham o número de iterações crescendo muito lentamente.

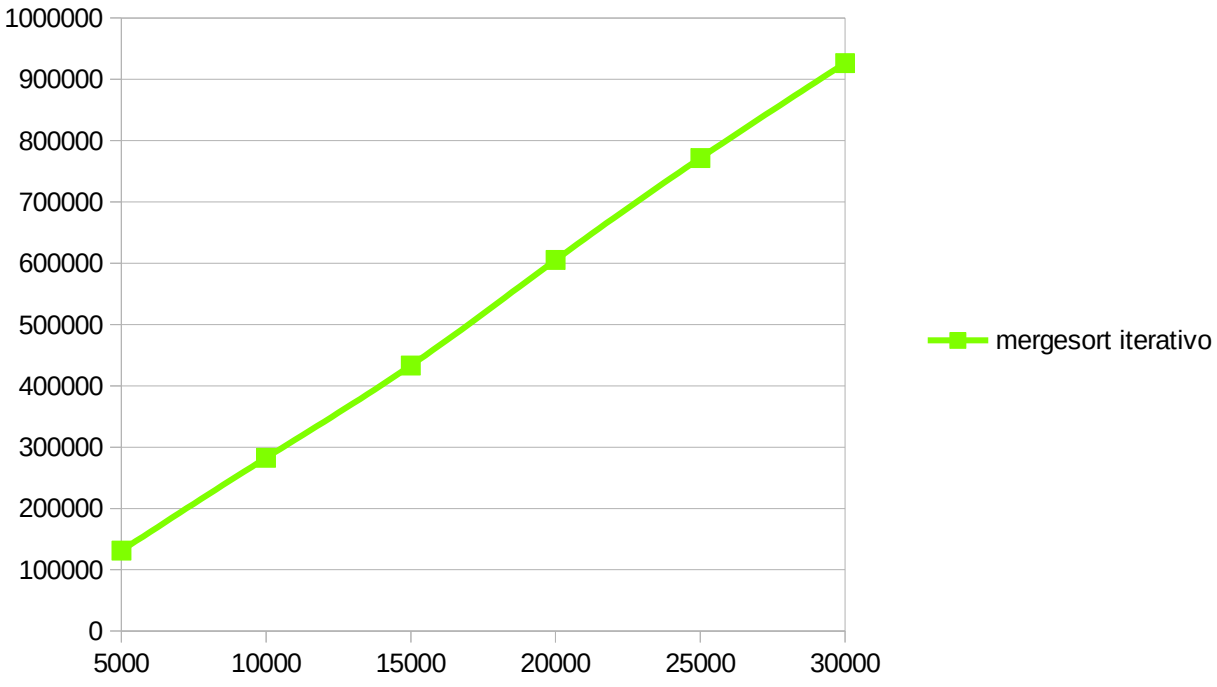
A partir de arrays de tamanho na escala dos 5000 acima os algoritmos de complexidade n^2 sempre perdiam, a ponto de o código travar a execução por não suportar tantas iterações. Desta maneira é mostrada intuitivamente que os algoritmos de complexidade n*Log n sempre vencem a partir de valores grandes devido ao calculo de sua complexidade e a lógica utilizada para seu funcionamento.

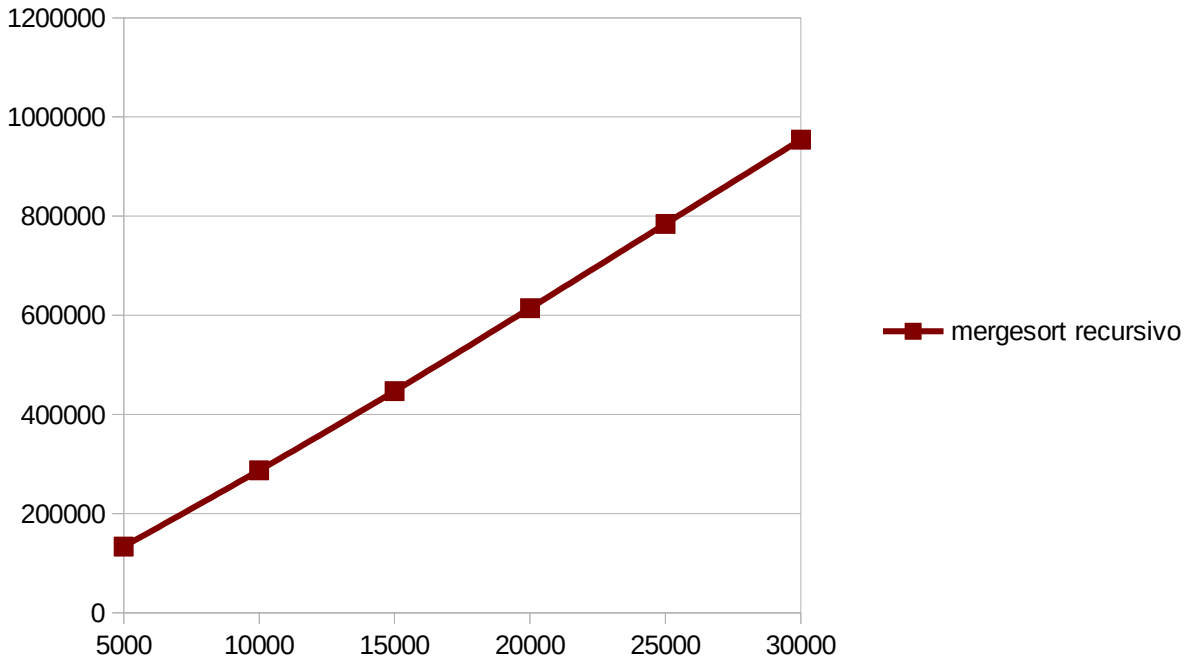
5. Gráficos

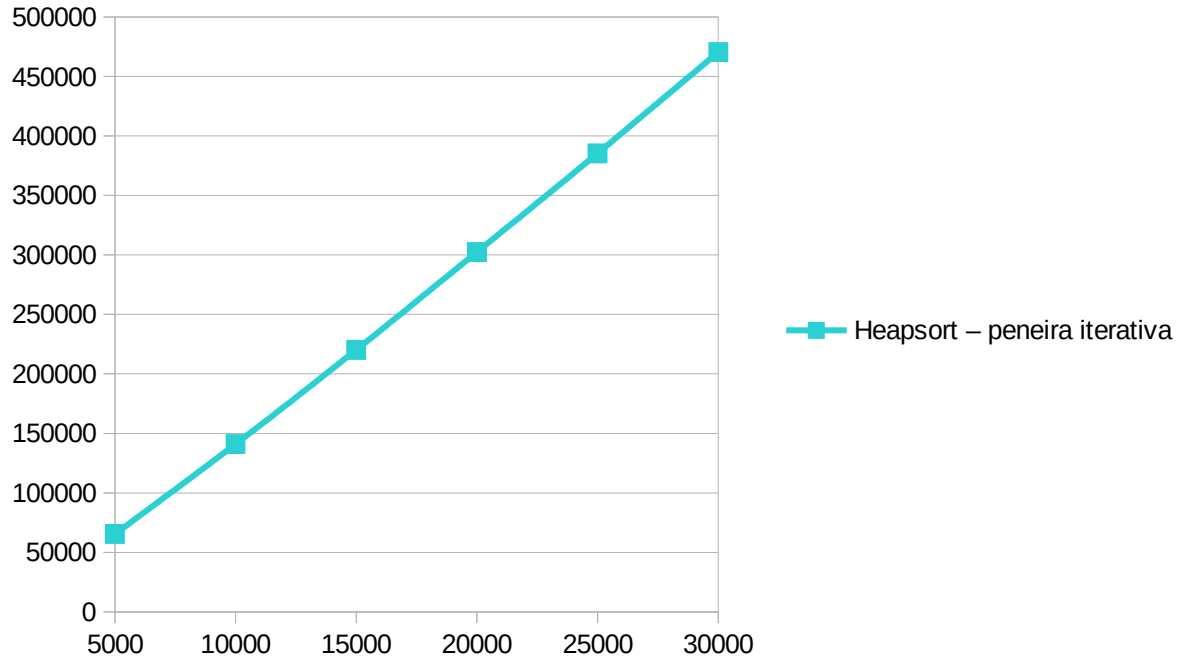

250000000
200000000
150000000
100000000
50000000
0
5000
10000
15000
20000
25000
30000
inserção iterativo

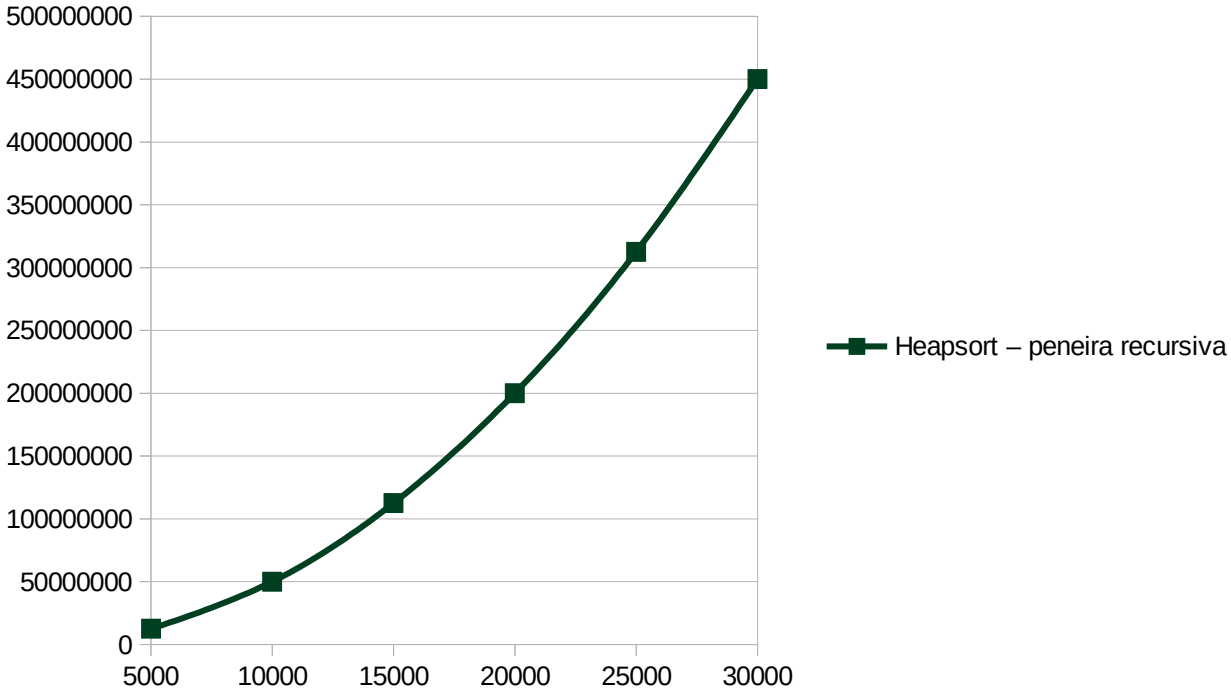

250000000
200000000
150000000
100000000
50000000
0
5000
10000
15000
20000
25000
30000
inserção recursivo

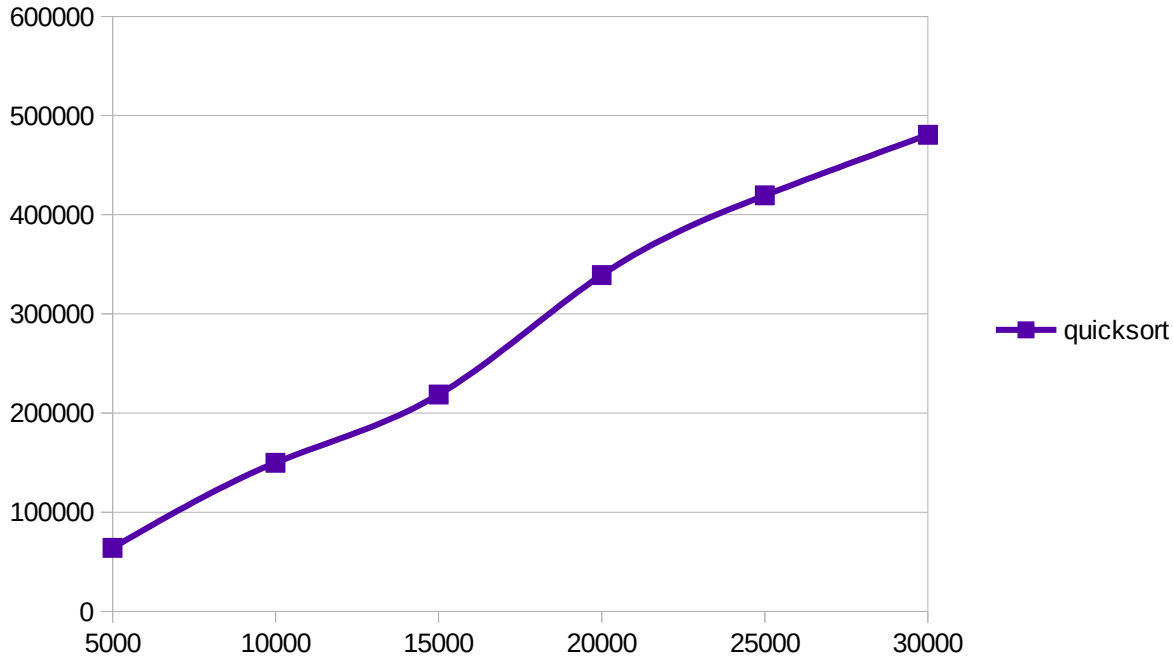

500000000
450000000
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
5000
10000
15000
20000
25000
30000
seleção

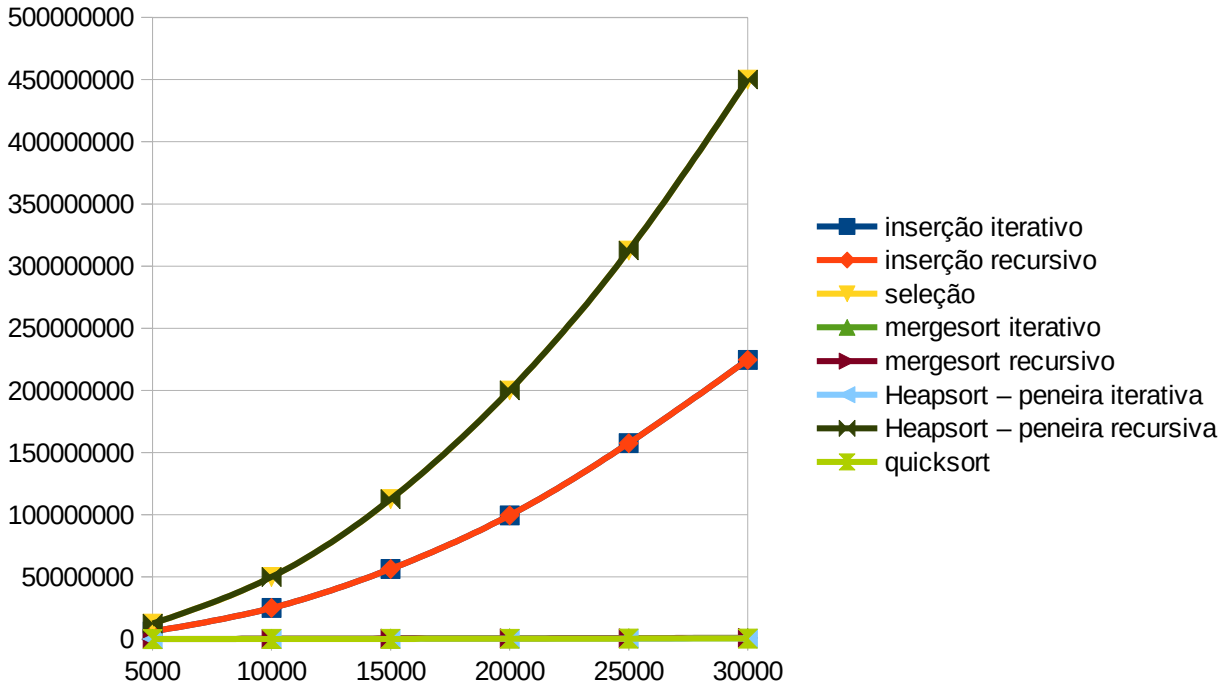

1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
5000
10000
15000
20000
25000
30000
mergesort iterativo

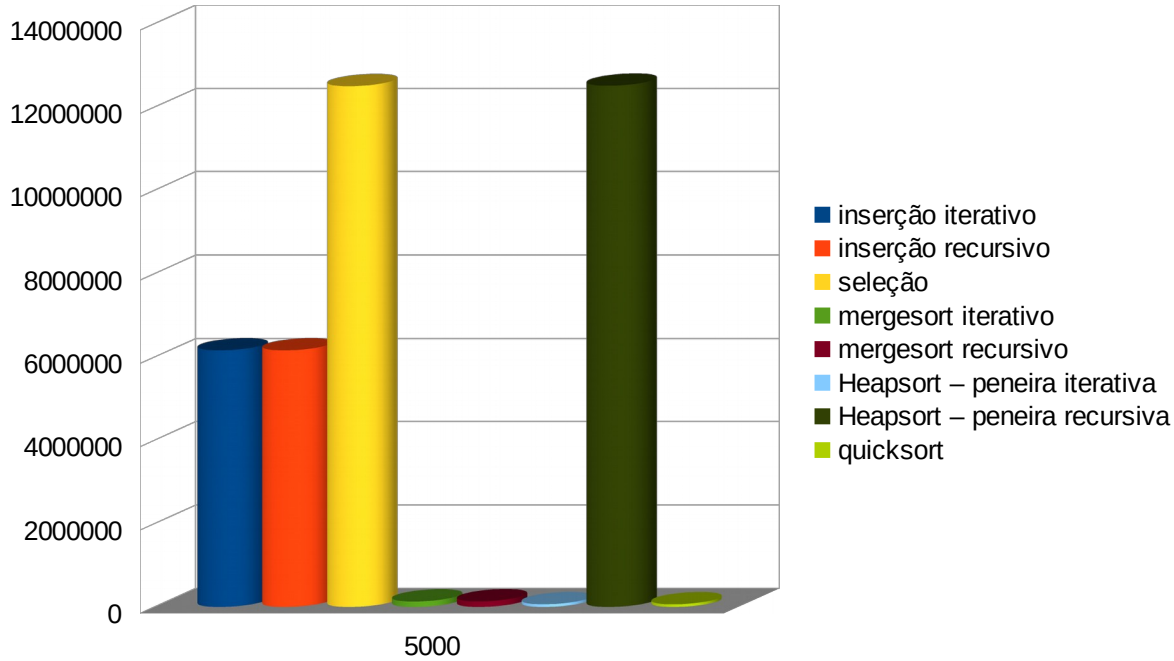

1200000
1000000
800000
600000
400000
200000
0
5000
10000
15000
20000
25000
30000
mergesort recursivo


500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
5000
10000
15000
20000
25000
30000
Heapsort – peneira iterativa


500000000
450000000
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
5000
10000
15000
20000
25000
30000
Heapsort – peneira recursiva


600000
500000
400000
300000
200000
100000
0
5000
10000
15000
20000
25000
30000
quicksort


500000000
450000000
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
5000
10000
15000
20000
25000
30000
inserção iterativo
inserção recursivo
seleção
mergesort iterativo
mergesort recursivo
Heapsort – peneira iterativa
Heapsort – peneira recursiva
quicksort


14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
5000
inserção iterativo
inserção recursivo
seleção
mergesort iterativo
mergesort recursivo
Heapsort – peneira iterativa
Heapsort – peneira recursiva
quicksort