**Aluno: Edwino Alberto Lopes Stein**

**Matricula: 1201324411;**

**Disciplina: Analise de Algoritmos;**

**Professor: Dr. Herbert Oliveira Rocha;**

**[QUESTÃO – 01] Para cada afirmação, indique se a mesma é falsa ou verdadeira, justificando sua resposta:**

para :

Logo:

**R: Verdadeira.**

**R: Verdadeira.**

Logo:

**é verdade para**

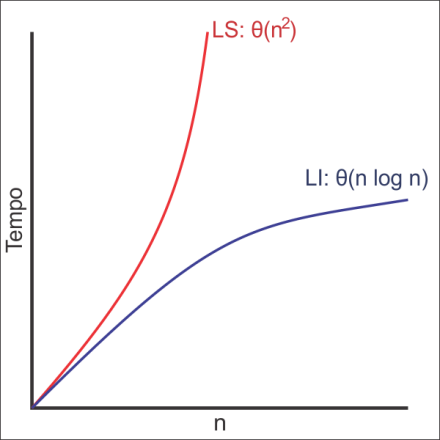
1. Se então e

Para :

**Logo, é Falsa.**

**[QUESTÃO – 02] Para o problema de ordenação: especificar o problema (descrição, instância, entrada, saída, objetivo), dar um exemplo gráfico e citar o limite inferior (LI) e superior (LS) do problema.**

* Descrição: Ordenar uma lista finita de números;
* Instância: (3, 7, 10, 9, 2, 1);
* Resultado: (1, 2, 3, 7, 9, 10);



**Quicksort**

**[QUESTÃO – 03] Obtenha a função de custo e a complexidade de tempo para os códigos presentados abaixo:**

Aplicando a recorrência observamos o padrão:

Para :

Aplicando a recorrência observamos o padrão:

Para :

**[QUESTÃO – 04] Descreva a técnica de divisão e conquista. Implemente um algoritmo utilizando divisão e conquista para encontrar o maior e o menor elemento em uma lista.**

A estratégia de divisão e conquista consiste em quebrar (divisão) o problema em problemas menores, e então essas partes menores são processadas de forma independente (conquista) para então solucionar o problema com as soluções das partes menores.

**Compilar e executar o programa:**

O código fonte do programa se encontra no arquivo *questao4.c*.

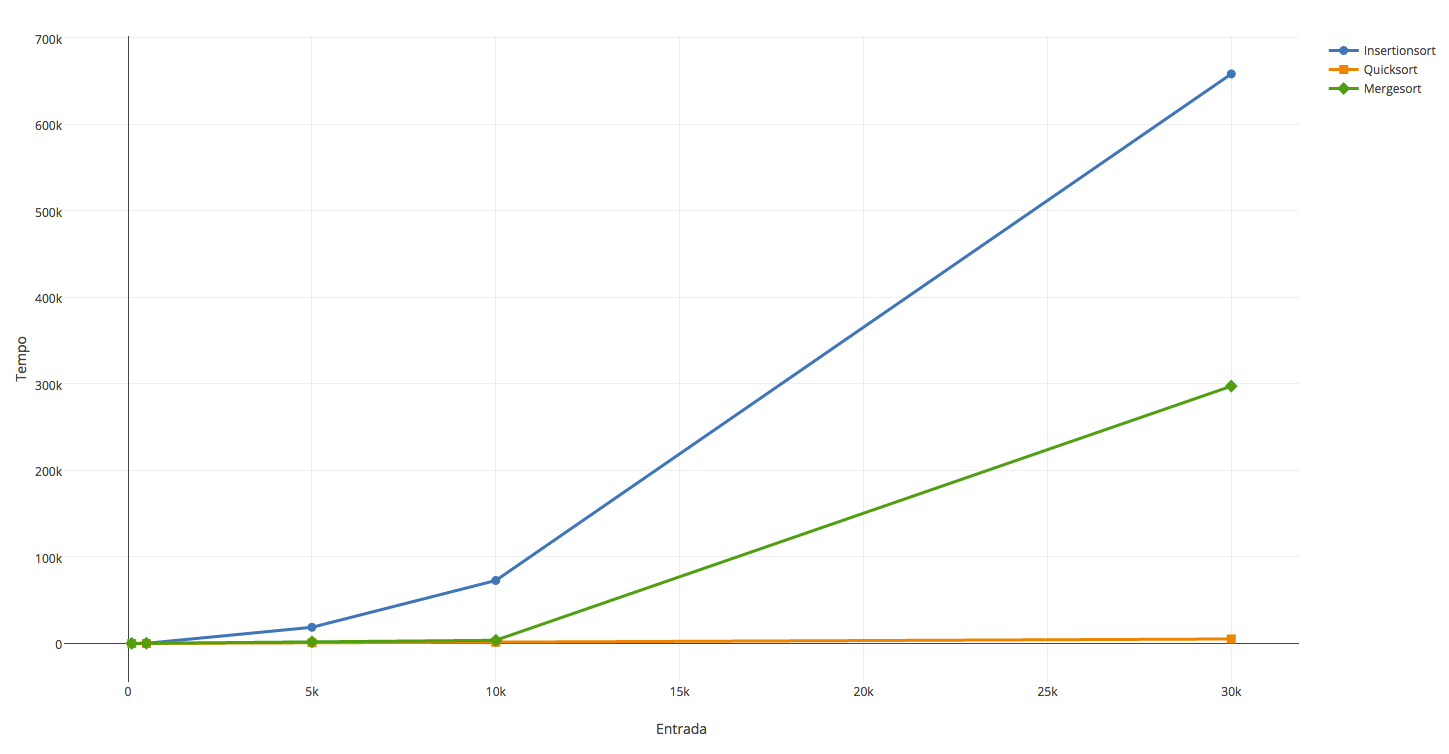
Para compilar e executar, basta utilizar o script run.sh e passar como parâmetro o arquivo *questao4.c*:

$ ./run.sh questao4.c

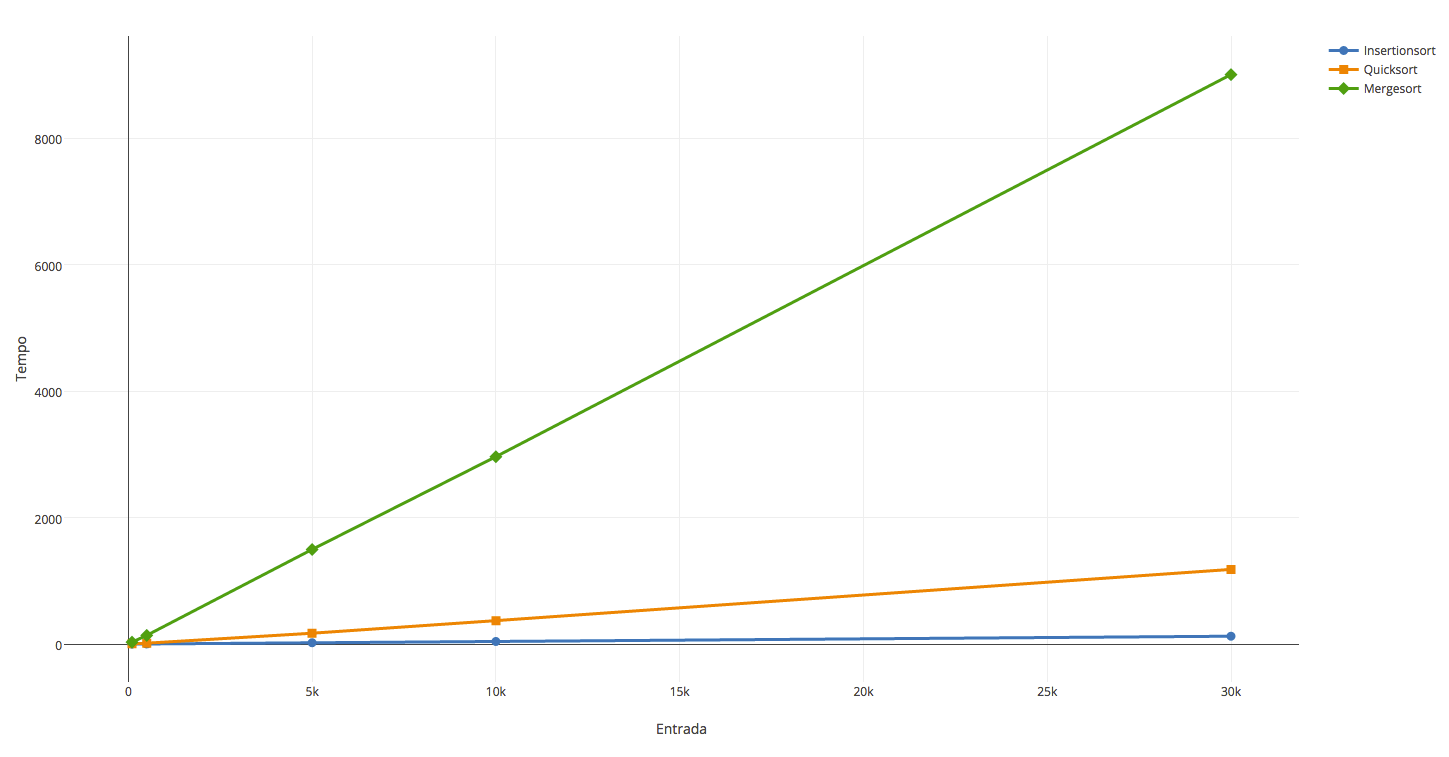
**[QUESTÃO – 05] Implemente os algoritmos de ordenação: Insertion Sort; QuickSort; e MergeSort. Apresente as complexidades dos algoritmos. Apresente um estudo empírico para analisar o tempo de execução dos algoritmos. Cada algoritmo de ordenação deve ser executado com entradas de diferentes tamanhos: 100, 500, 5000, 10000 e 30000. Também se deve utilizar diferentes configurações para cada tamanho de entrada para a ordenação: números aleatório; em ordem crescente; e em ordem decrescente. Crie um gráfico de linhas para cada configuração de entrada com os tempos de ordenação para comparar os algoritmos, onde cada linha do gráfico irá representar um algoritmo.**

**Complexidades:**

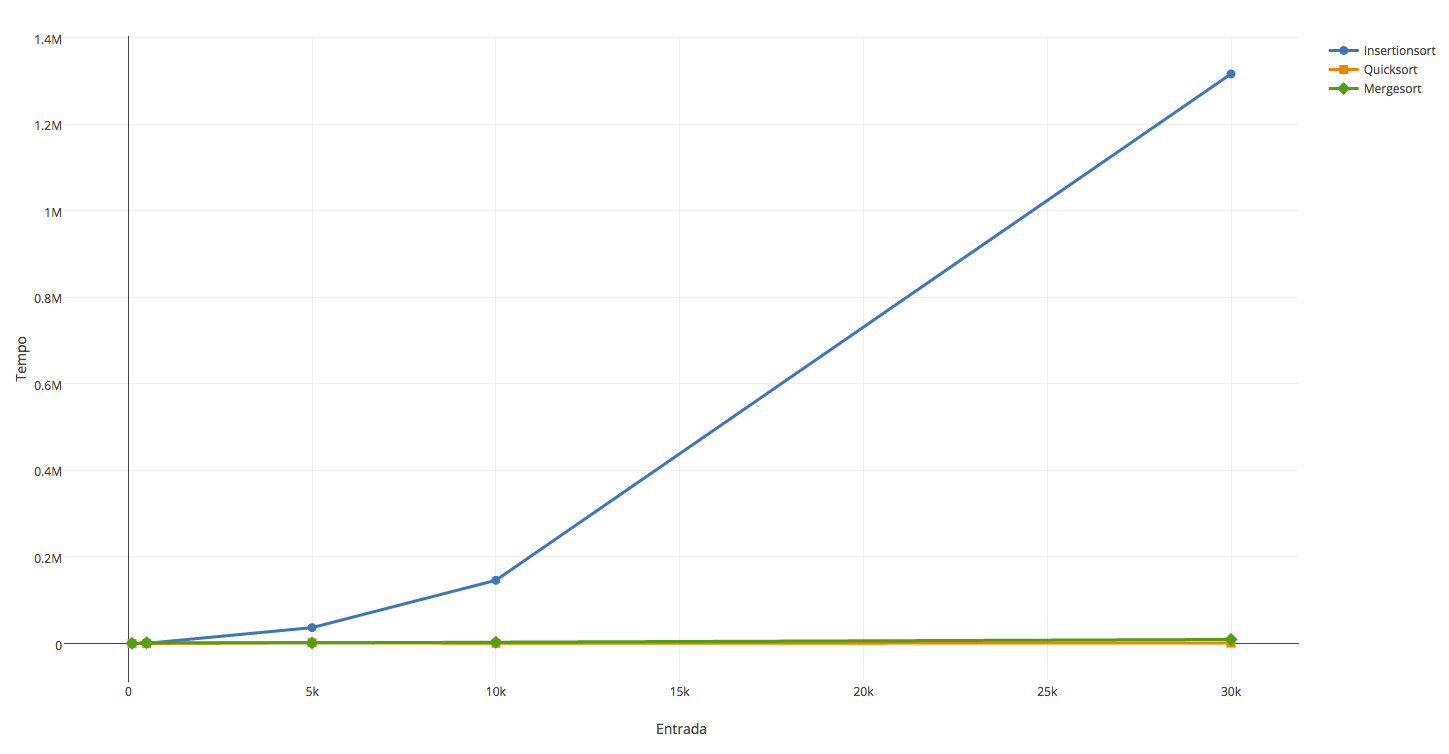
* InsertionSort:
  + Pior caso: ;
  + Melhor caso: ;
  + Caso médio: ;
* QuickSort:
  + Pior caso: ;
  + Melhor caso: ;
  + Caso médio: ;
* MergeSort:
  + Pior caso: ;
  + Melhor caso: ;
  + Caso médio: ;



**Entradas aleatórias (tempo em microssegundos)**



**Entradas em ordem crescente (tempo em microssegundos)**

****

**Entradas em ordem decrescente (tempo em microssegundos)**

**Compilar e executar o programa:**

Os códigos do programa se encontram nos arquivos:

* *questao5.c*: Arquivo do programa principal;
* *sorts/src/quicksort.c*: Implementação do Quicksort;
* *sorts/src/mergesort.c*: Implementação do Mergesort;
* *sorts/src/insertionsort.c*: Implementação do Insertionsort;
* *sorts/src/swap.c*: Implementação da função de troca de valores inteiros;

Para compilar e executar, basta utilizar o script run.sh e passar como parâmetro o arquivo *questao5.c*, seguidos dos parâmetros esperados pelo programa:

$ ./run.sh questao5.c <algoritmo> <arquivo\_de\_entrada> <tamanho\_do\_buffer>

Onde:

* **<algoritmo>:**Algoritmo de ordenação desejado (quicksort, mergesort ou insertionsort);
* **<arquivo\_de\_entrada>:** Arquivo de entrada contendo os valores que serão ordenados. Este arquivo pode ser gerado utilizando o programa *generate.c*;
* **<tamanho\_do\_buffer>:** Quantidade máxima de valores esperados pelo programa;

Exemplo:

$ ./run.sh questao5.c quicksort input-1000-random.txt 1000

OBS.: Os resultados do programa serão gravados em arquivos no diretório output.

**[QUESTÃO – 06] Descreva os passos para ordenação de um vetor usando o algoritmo Quick Sort.**

1. Se o vetor tiver apenas um elemento, o mesmo já está ordenado;
2. Caso contrário, selecione um elemento para ser o pivô;
3. Passe para esquerda do pivô todos os elementos menores que ele;
4. Passe para direita do pivô todos os elementos maiores que ele;
5. Repita recursivamente a partir do passo 1, porem agora passando como parâmetro os valores do inicio do vetor até o pivô;
6. Repita recursivamente a partir do passo 1, porem agora passando como parâmetro os valores do pivô até o final do vetor;

**[QUESTÃO – 07] Descreva as regras de balanceamento em uma árvore vermelho e preto (red and black). Adicionalmente, apresente de forma gráfica a inserção dos seguintes valores em uma árvore vermelho e preto: 11; 7; 8; 14; 4; 15; 1; 2; 5.**

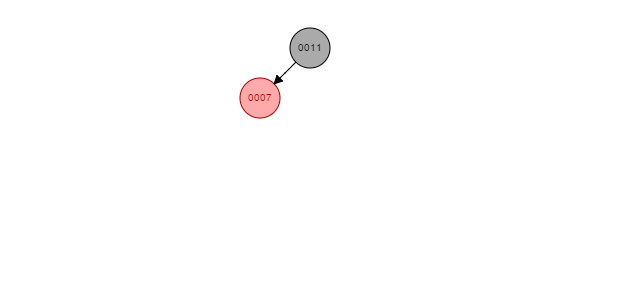
* A raiz é sempre preta;
* Nós vermelhos sempre devem ter filhos pretos;
* Nós vazios ou nulos (NULL) sempre são pretos;
* Todo caminho partindo de um nó qualquer até suas folhas, passa ela mesma quantidade de nós pretos;

**Exemplo:**

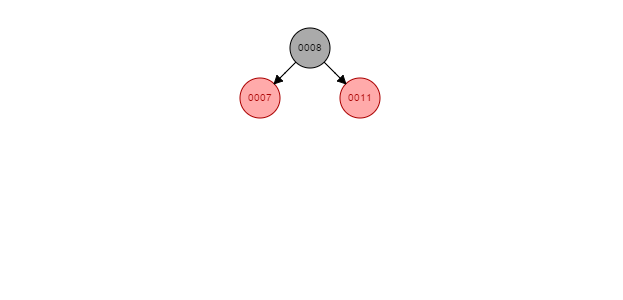
1. Inserir 11:



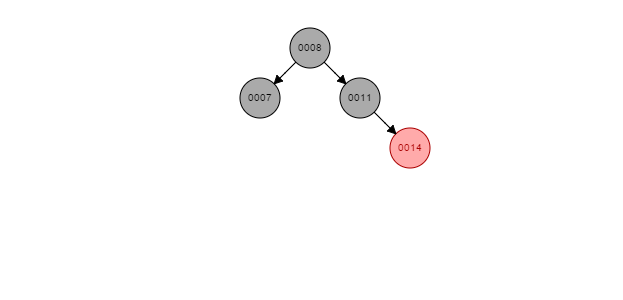
1. Inserir 7:



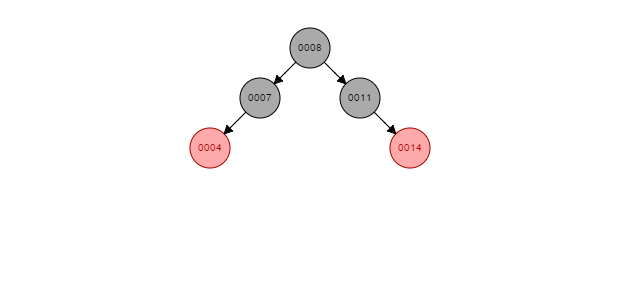
1. Inserir 8:



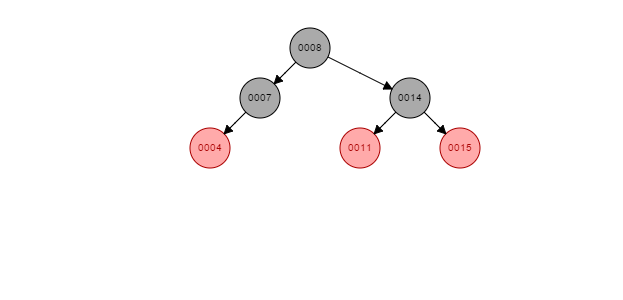
1. Inserir 14:



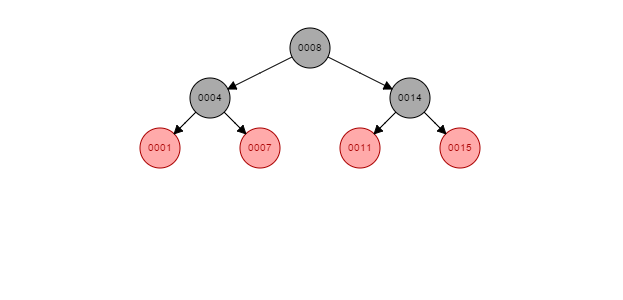
1. Inserir 4:



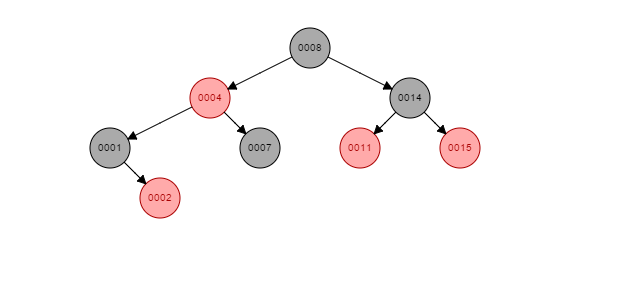
1. Inserir 15:



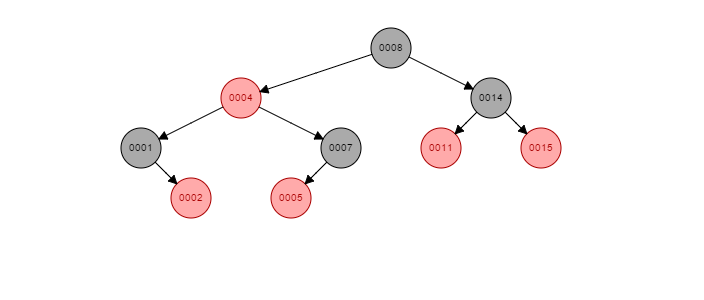
1. Inserir 1:



1. Inserir 2:



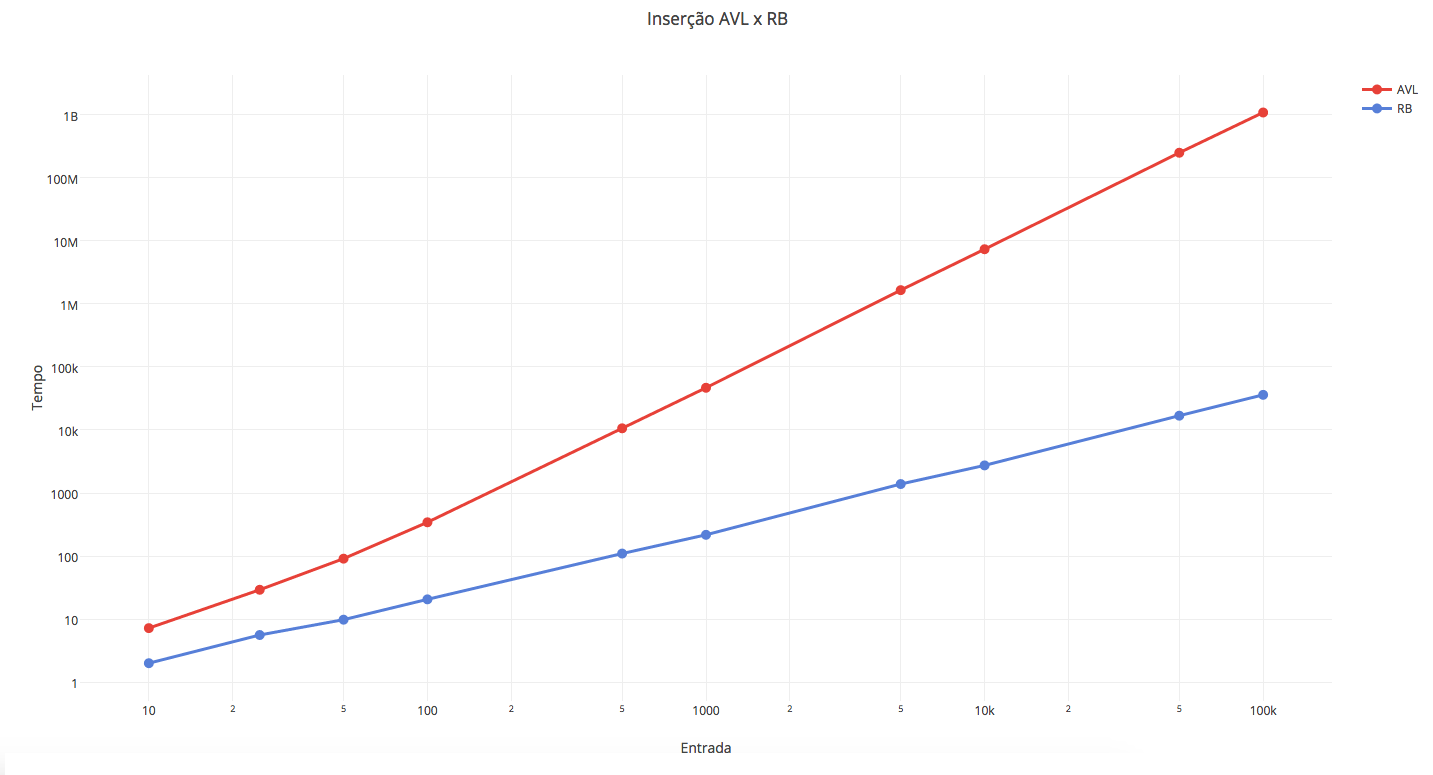
1. Inserir 5:



**[QUESTÃO – 08] Implemente a operação de inserção da árvore AVL e árvore vermelho e preto. Apresente um estudo empírico para obter custos de inserção na medida em que o número de elementos da árvore aumenta. Gere gráficos para mostrar o custo médio de inserção para tamanhos distintos de N (exemplo: de 10 a 1000000). Apresente uma análise de comparação entre árvore AVL e árvore vermelho e preto em relação ao tempo de execução. Adicionalmente, apresente a complexidade da operação de inserção da árvore AVL e árvore vermelho e preto.**

**Complexidades:**

* Árvore AVL:
  + Pior caso: ;
  + Melhor caso: ;
* Árvore vermelha e preta:
  + Pior caso: ;
  + Melhor caso: ;



**Gráfico inserção AVL vs Red-black (tempo em microssegundos)**

**Compilar e executar o programa:**

Os códigos do programa se encontram nos arquivos:

* *questao8.c*: Arquivo do programa principal;
* *structs/src/avlTree.c*: Implementação da Árvore AVL;
* *structs/src/binaryTree. c*: Implementação da Árvore Binária de Busca;
* *structs/src/RBTree.c*: Implementação do Árvore vermelha e preta;

Para compilar e executar, basta utilizar o script *run.sh* e passar como parâmetro o arquivo *questao8.c*, seguidos dos parâmetros esperados pelo programa:

$ ./run.sh questao8.c <arvore> <arquivo\_de\_entrada> <tamanho\_do\_buffer>

Onde:

* **<arvore>:** Tipo da árvore que a ser utilizada (avltree ou rbtree);
* **<arquivo\_de\_entrada>:** Arquivo de entrada contendo os valores que serão inseridos. Este arquivo pode ser gerado utilizando o programa *generate.c*;
* **<tamanho\_do\_buffer>:** Quantidade máxima de valores esperados pelo programa;

Exemplo:

$ ./run.sh questao8.c avltree input-1000-random.txt 1000

OBS.: Os resultados do programa serão gravados em arquivos no diretório output.