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:

a) $n + (logn) = \theta(n)$

$$0 \leq c_1 n \leq n + logn \leq c_2 n$$

para $n = 2$:

$$0 \leq 2c_1 \leq 2 + log2 \leq 2c_2$$

$$2c_2 \leq 2 + log2 \leftrightarrow 2c_2 \leq 2 + 1 \leftrightarrow 2c_2 \leq 3 \leftrightarrow c_2 \leq \frac{3}{2}$$

$$2 + log2 \leq 2c_2 \leftrightarrow 2 + 1 \leq 2c_2 \leftrightarrow 3 \leq 2c_2 \leftrightarrow \frac{3}{2} \leq c_2$$

Logo:

$$0 \leq \frac{3}{2} n \leq n + logn \leq \frac{3}{2} n$$

R: Verdadeira.

b) $n^2 = o(n^3)$

$$lim \lim_{n \to \infty} \frac{n^2}{n^3} = 0$$

R: Verdadeira.

c) $(n+1)^2 = O(2n^2)$

$$0 \leq (n+1)^2 \leq 2n^2$$

$$n = 1 \rightarrow (1+1)^2 \leq 2.1^2 \rightarrow 2^2 \leq 2 \rightarrow Falsa$$

$$n = 2 \rightarrow (2+1)^2 \leq 2.2^2 \rightarrow 3^2 \leq 2.4 \rightarrow Falsa$$

$$n = 3 \rightarrow (3+1)^2 \leq 2.3^2 \rightarrow 4^2 \leq 2.9 \rightarrow Verdadeira$$

$$n = 4 \rightarrow (4+1)^2 \leq 2.4^2 \rightarrow 5^2 \leq 2.16 \rightarrow Verdadeira$$

Logo:

$$(n+1)^2 = O(2n^2) \text{ é verdade para } n_0 = 3$$

d) Se $f(n) = n - 300$ então $f(n) = \Omega(300n)$ e $f(n) = \Omega(300n)$

$$0 \leq 300n \leq n - 300 \leq 300n$$

Para Ω:

$$300n \leq n - 300$$

$$300n + 300 \leq n$$

$$300(n+1) \leq n$$

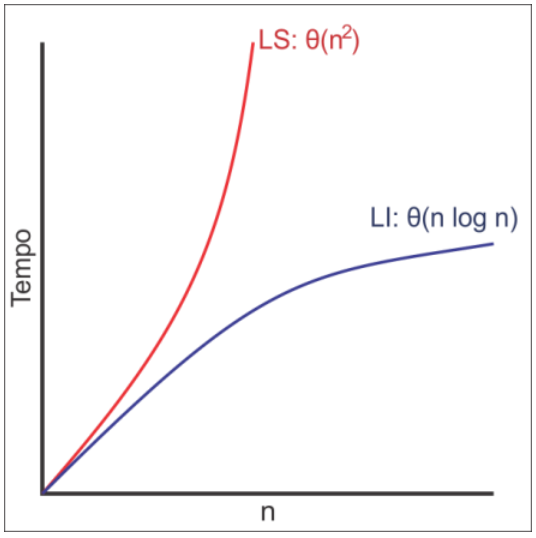
$$n + 1 \leq \frac{n}{300}$$

$$1 \leq \frac{n}{300} - n$$

Logo, $f(n) = \Omega(300n)$ 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:

a)

$$\sum_{l=1}^{10000}\sum_{i=1}^{n-5}\sum_{j=i+2}^{n/2}\sum_{k=1}^{n} 1 = 10000n^2 - 50000\text{n}$$

b)

$$T(n) = \begin{cases} 1 & se\ n = 1 \\ 3T\left(\frac{n}{3}\right) + 5n - 2 & se\ n > 1 \end{cases}$$

Aplicando a recorrência observamos o padrão:

$$3^k T\left(\frac{n}{3^k}\right) + k\frac{5n}{3} - 2\sum_{i=0}^{k-1} 3^i$$

Para $k = \log_3 n$:

$$\frac{n}{2} - n\log_3 n\frac{5}{3} + \frac{3}{2}$$

c)

$$\sum_{i=1}^{n-2}\sum_{j=i+1}^{n}\sum_{k=1}^{j} 1 = \frac{n^3}{3} - \frac{3n^2}{2} + \frac{17n}{2} - 1$$

d)

$$T(n) = \begin{cases} 1 & se\ n = 0 \\ 2T(n-1) + 1 & se\ n > 1 \end{cases}$$

Aplicando a recorrência observamos o padrão:

$$2^k T(n-k) + \sum_{i=0}^{n-1} 2^i$$

Para $k = n$:

$$2^{n+1} - 1$$

[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
```

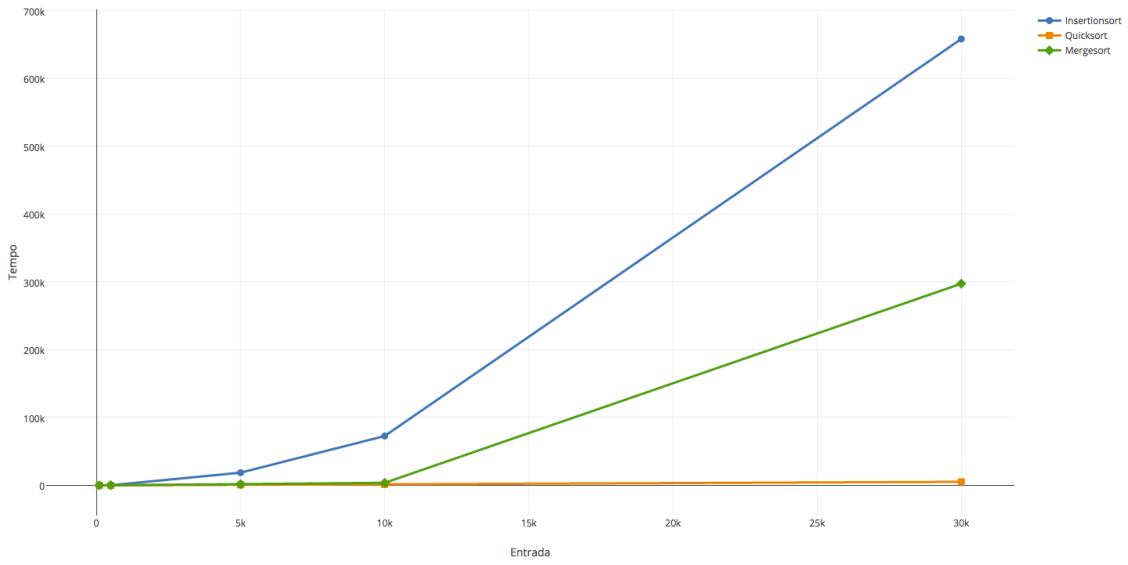
[QUESTÃO – 05] Implemente os algoritmos de ordenação: Insertion Sort; QuickSort; e MergeSort. Apresente as complexidades dos algoritmos. Apresente um estudo empírico para analisar o tempo de execução dos algoritmos. Cada algoritmo de ordenação deve ser executado com entradas de diferentes tamanhos: 100, 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:

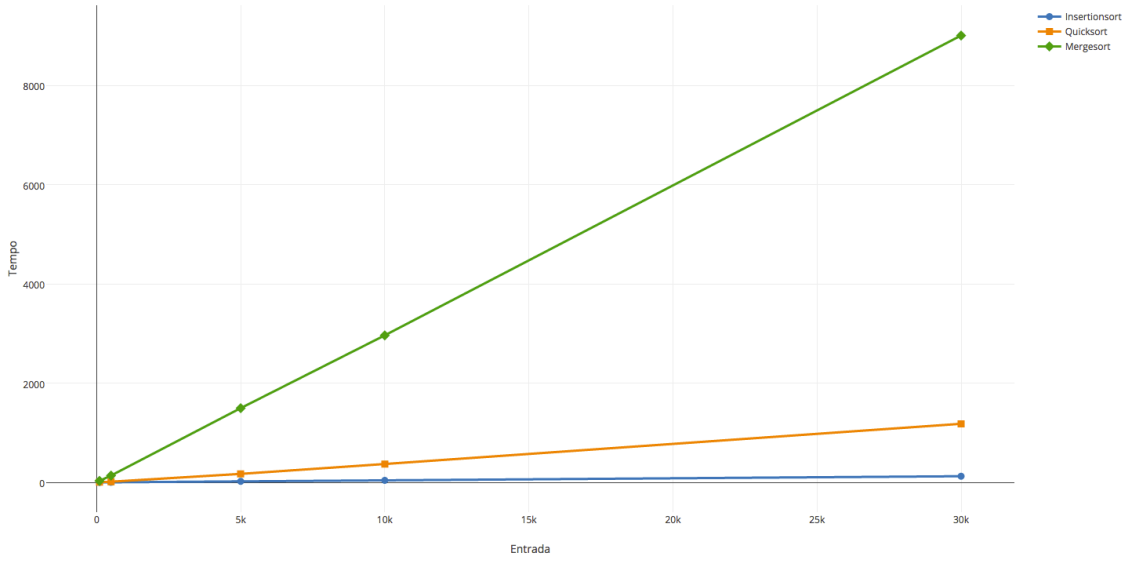
- InsertionSort:
  - Pior caso: $\theta(n^2)$;
  - Melhor caso: $\theta(n^2)$;
  - Caso médio: $\theta(n^2)$;

- QuickSort:
  - Pior caso: $\theta(n^2)$;
  - Melhor caso: $\theta(nlogn)$;
  - Caso médio: $\theta(nlogn)$;

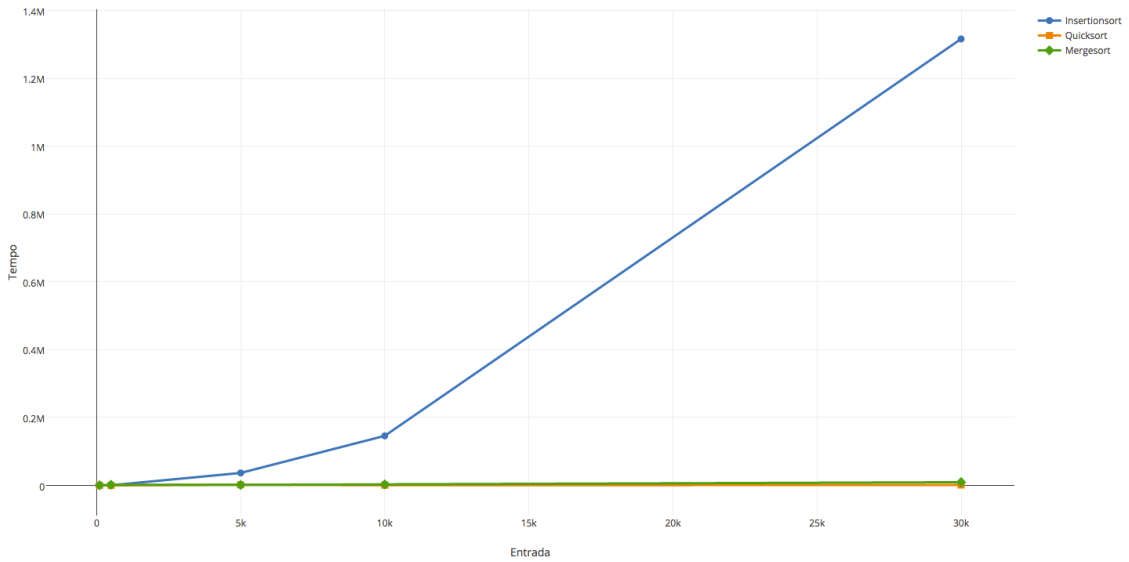
- MergeSort:
  - Pior caso: $\theta(nlog2n)$;
  - Melhor caso: $\theta(nlogn)$;
  - Caso médio: $\theta(nlogn)$;



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.

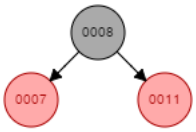
- A raiz é sempre preta;
- Nós vermelhos sempre devem ter filhos pretos;
- Nós vazios ou nulos (NULL) sempre são pretos;
- Todo caminho partindo de um nó qualquer até suas folhas, passa ela mesma quantidade de nós pretos;

Exemplo:

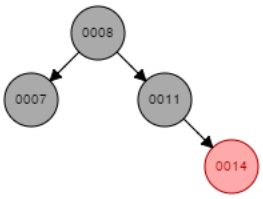
1. Inserir 11:

2. Inserir 7:

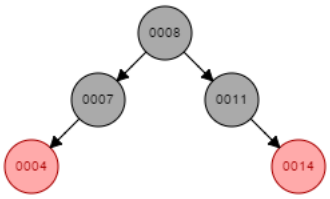
3. Inserir 8:



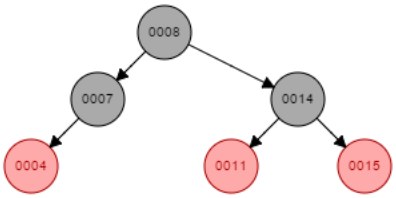
4. Inserir 14:



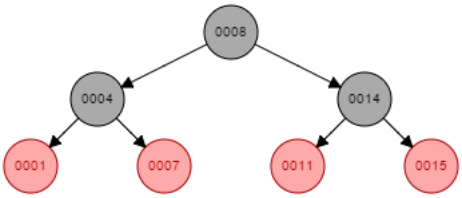
5. Inserir 4:



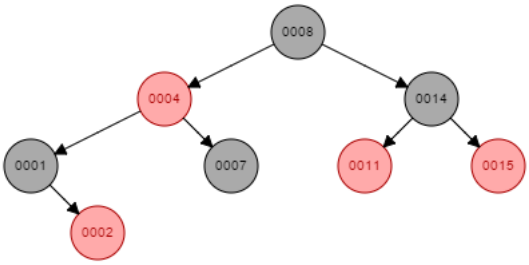
6. Inserir 15:



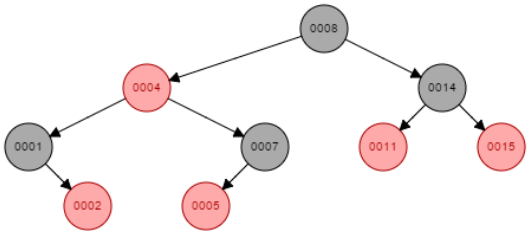
7. Inserir 1:



8. Inserir 2:



9. 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: $\theta(\log n)$;
  - Melhor caso: $\theta(\log n)$;
- Árvore vermelha e preta:
  - Pior caso: $\theta(\log n)$;
  - Melhor caso: $\theta(logn)$;

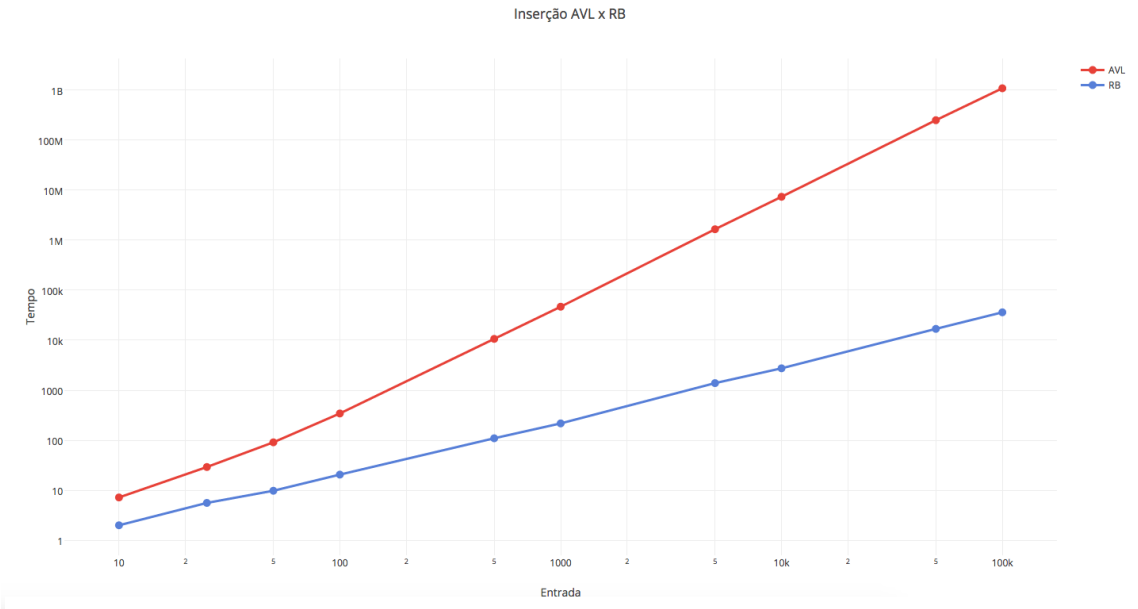


Gráfico inserção AVL vs Red-black (tempo em microssegundos)

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.