Avaliação RA02 – Relatório da Atividade

Resolução de Problemas Estruturados em Computação

Turma 4º A - Ciência da Computação - Manhã - 2023

Aluna: Thaíssa Vitória Calil

A avaliação é composta pela criação de dois códigos em java, o primeiro sendo a implementação de uma árvore binária de busca e o segundo sendo a implementação da árvore AVL, as duas com os métodos de inserção, exclusão e busca.

Descrição do código da árvore binária:

Classe Node: usada para criar um novo nó. Constrói nós com dados e os filhos da árvore(subárvores direita e esquerda).

Classe binaria: onde a maioria dos métodos estão:

1 - inserir(): adiciona um novo nó na árvore com recursão.

2 - deletar(): remove um nó da árvore.

3 - menorNE(): encontra o menor valor em uma subárvore.

4 - buscar(): busca um dado na árvore de acordo com o seu nó.

5 - imprimir(): imprime a árvore.

main(): insere 100 números inteiros aleatórios na árvore, possui um menu para que o usuário escolha entre inserção, exclusão, busca, impressão finalizar a aplicação.

Descrição do código da árvore AVL:

Classe Node: usada para criar um novo nó. Constrói nós com dados, altura e os filhos da árvore(subárvores direita e esquerda).

Classe avl: onde a maioria dos métodos estão:

Métodos privados:

1 - **altura():** Calcula a altura.

2 - **maiorValor():** Calcula o valor mais alto entre dois nós.

3 - **rotacaoD():** Faz a rotação para direita de um nó.

4 - **rotacaoE():** Faz a rotação para esquerda de um nó.

5 - **equilibrio():** Calcula o equilíbrio de um nó.

6 - **inserir():** Insere um dado e rebalanceia a árvore.

7 - **ordem():** Modo pré-ordem da árvore.

8 - **valorMinimo():** Acha o menor valor de uma subárvore.

9 - **deletarNode():** Exclui nós e rebalanceia a árvore.

10 - **buscarNaArvore():** Busca um dado na árvore.

Métodos públicos:

1 - **inserir():** Insere elemento a árvore.

2 - **deletar():** Exclui elemento da árvore.

3 - **buscar():** Busca dado e imprime a altura dele na árvore.

4 - **imprimir():** Imprime a árvore na forma pré-ordem.

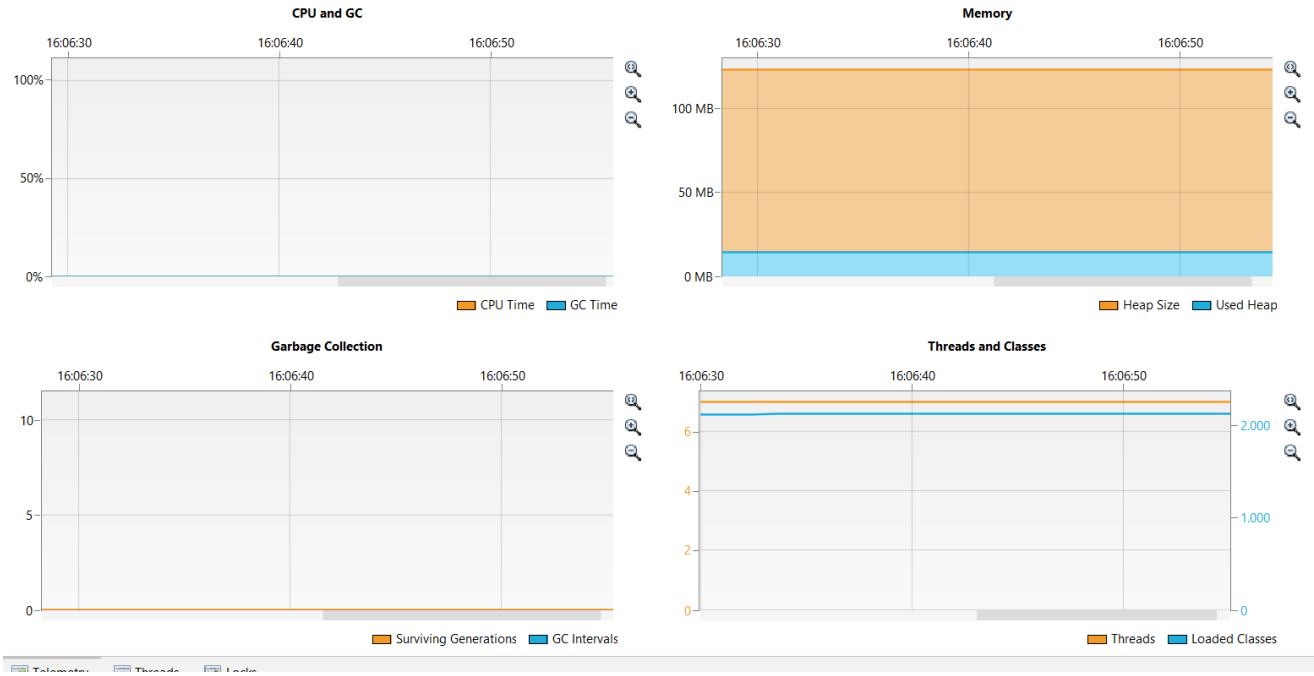
main(): Cria a árvore e gera e insere 100 números aleatórios nela, possui um menu para que o usuário escolha entre inserir, excluir, buscar, imprimir ou sair da aplicação.

Análise de desempenho

Árvore Binária:

Binária – 100 números

Profile do NetBeans:



Tempo de Execução:

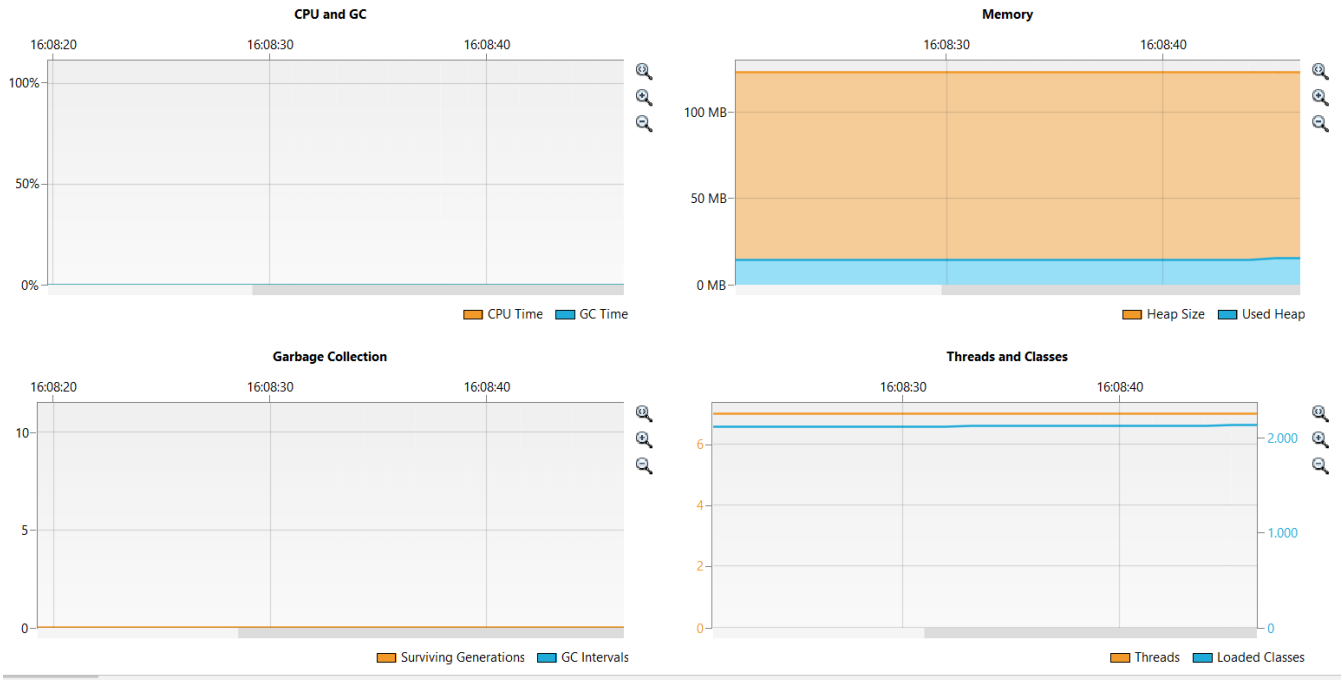
```
Digite uma posição para descobrir o seu dado:
66
Encontrado na posição 66: Node{dado=699}
Tempo de execução da busca: 4495 ms
```

```
Digite um número para deletar da árvore:
669
Tempo de execução da exclusão: 2538 ms
```

```
937
942
947
958
968
986
989
Tempo de execução da impressão: 3 ms
```

Binária – 500 números

Profile do NetBeans:



Tempo de Execução:

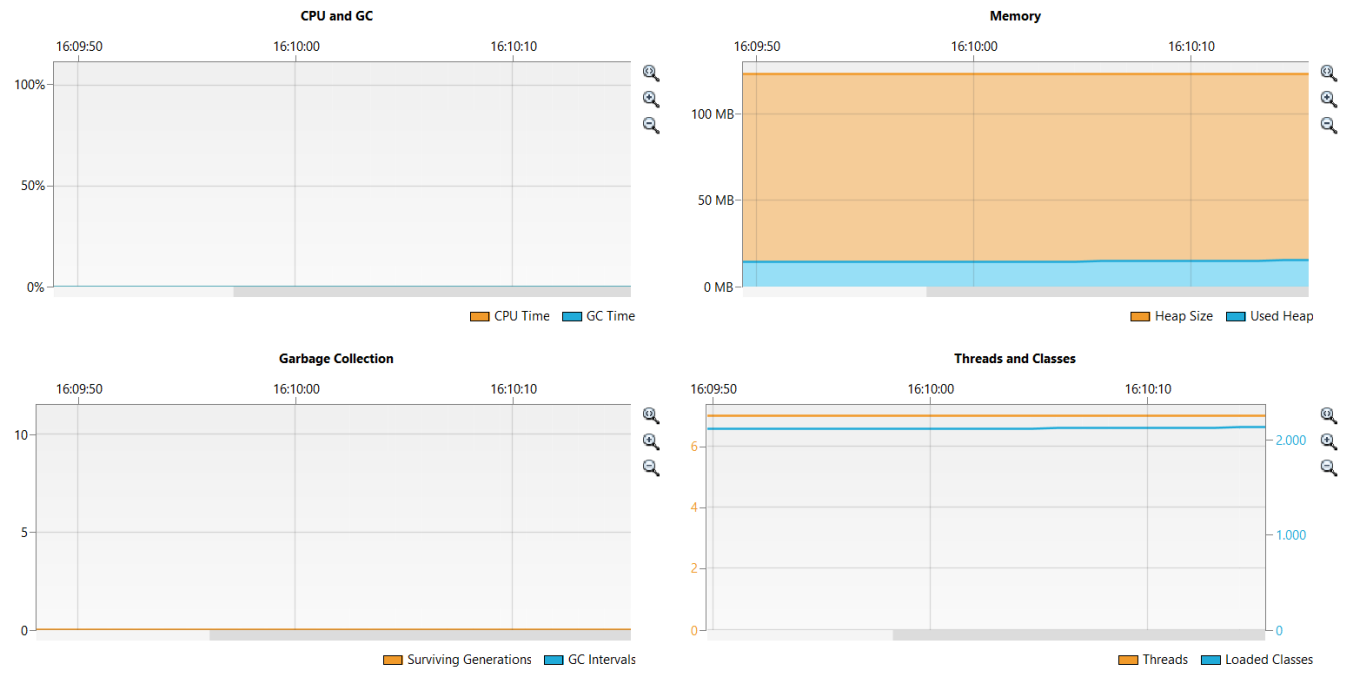
```
Digite uma posição para descobrir o seu dado:
450
Encontrado na posição 450: Node{dado=9090}
Tempo de execução da busca: 3324 ms
```

```
Digite um número para deletar da árvore:
9090
Tempo de execução da exclusão: 2172 ms
```

```
9912
9925
9950
9952
9959
9960
9969
9993
Tempo de execução da impressão: 9 ms
```

Binária – 1000 números

Profile do NetBeans:



Tempo de Execução:

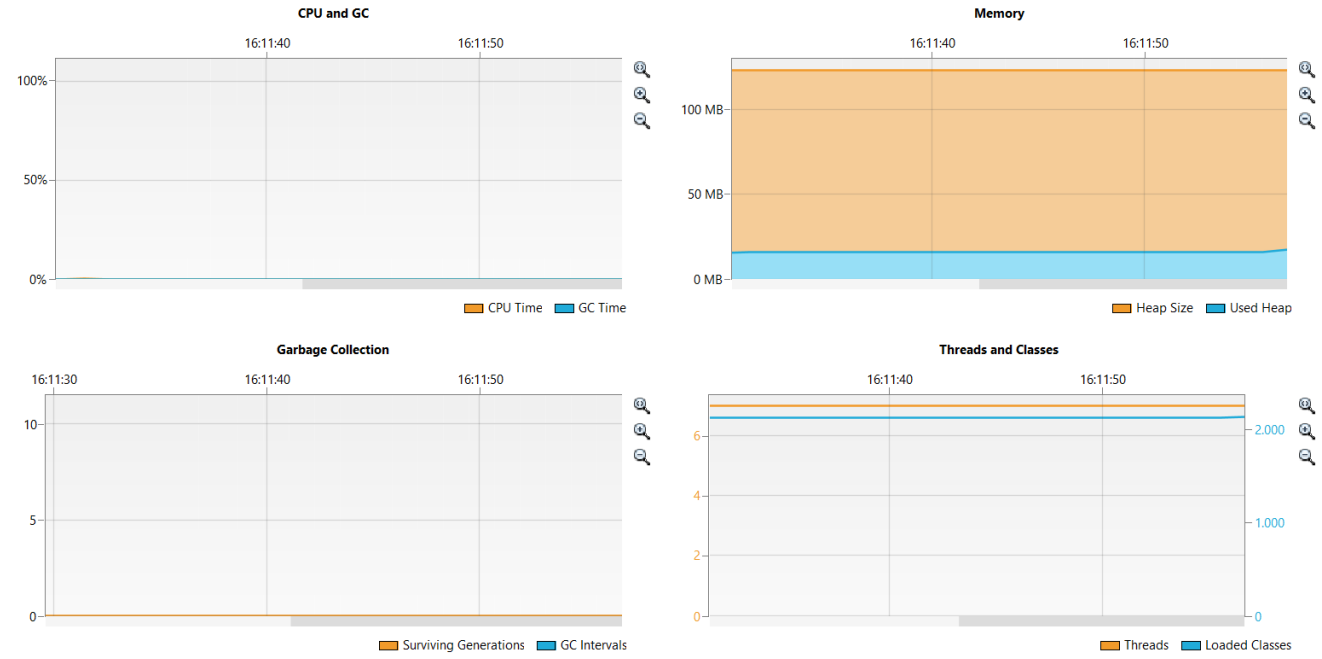
```
Digite uma posição para descobrir o seu dado:
888
Encontrado na posição 888: Node{dado=9435}
Tempo de execução da busca: 3035 ms
```

```
Digite um número para deletar da árvore:
9435
Tempo de execução da exclusão: 2234 ms
```

```
9930
9933
9936
9943
9945
9948
9955
9967
9989
Tempo de execução da impressão: 12 ms
```

Binária – 10.000 números

Profile do NetBeans:



Tempo de Execução:

```
Digite uma posição para descobrir o seu dado:
8888
Encontrado na posição 8888: Node{dado=93047}
Tempo de execução da busca: 3952 ms
```

```
Digite um número para deletar da árvore:
93047
Tempo de execução da exclusão: 1528 ms
```

```
99931
99933
99938
99946
99950
99951
99959
99984
99988
Tempo de execução da impressão: 195 ms
```

Binária – 20.000 números

Profile do NetBeans:

Tempo de Execução:

```
Digite uma posição para descobrir o seu dado:
18000
Encontrado na posição 18000: Node{dado=909921}
Tempo de execução da busca: 3327 ms
```

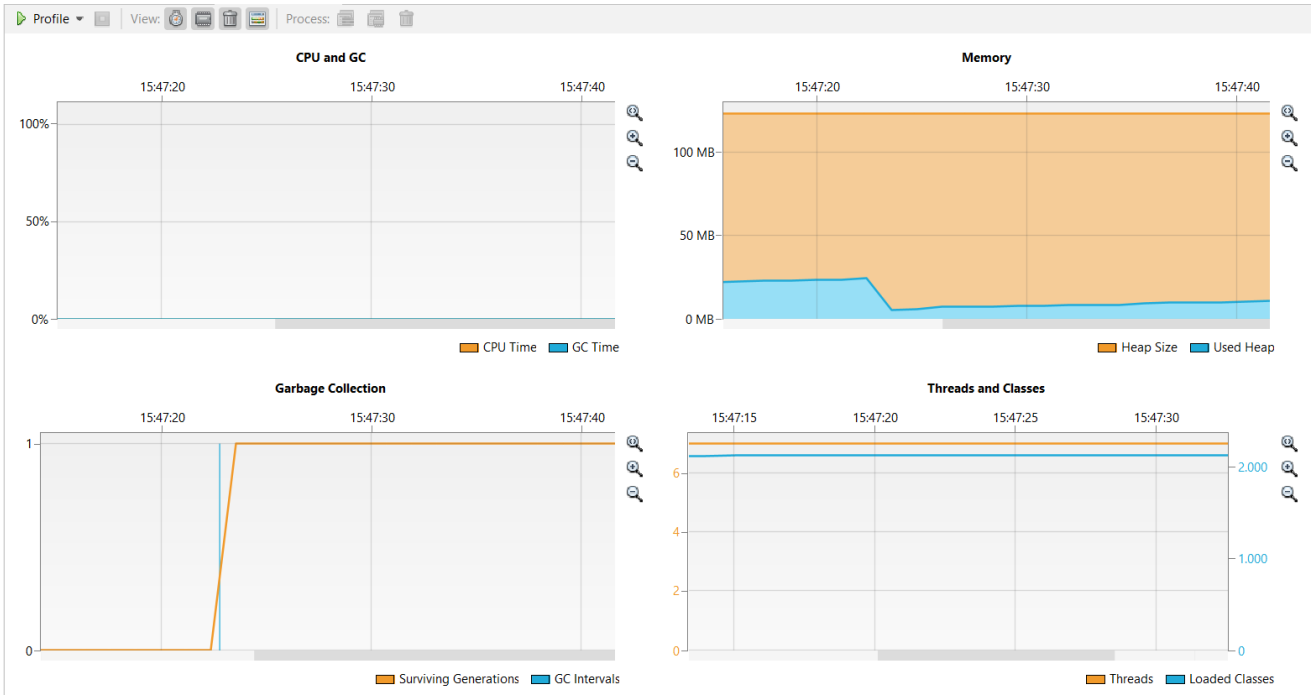
```
Digite um número para deletar da árvore:
909921
Tempo de execução da exclusão: 1456 ms
```

```
 99973
 99977
 99979
 99980
 99992
 99995
 Tempo de execução da impressão: 358 ms
```

Árvore AVL

AVL – 100 números

Profile do NetBeans:



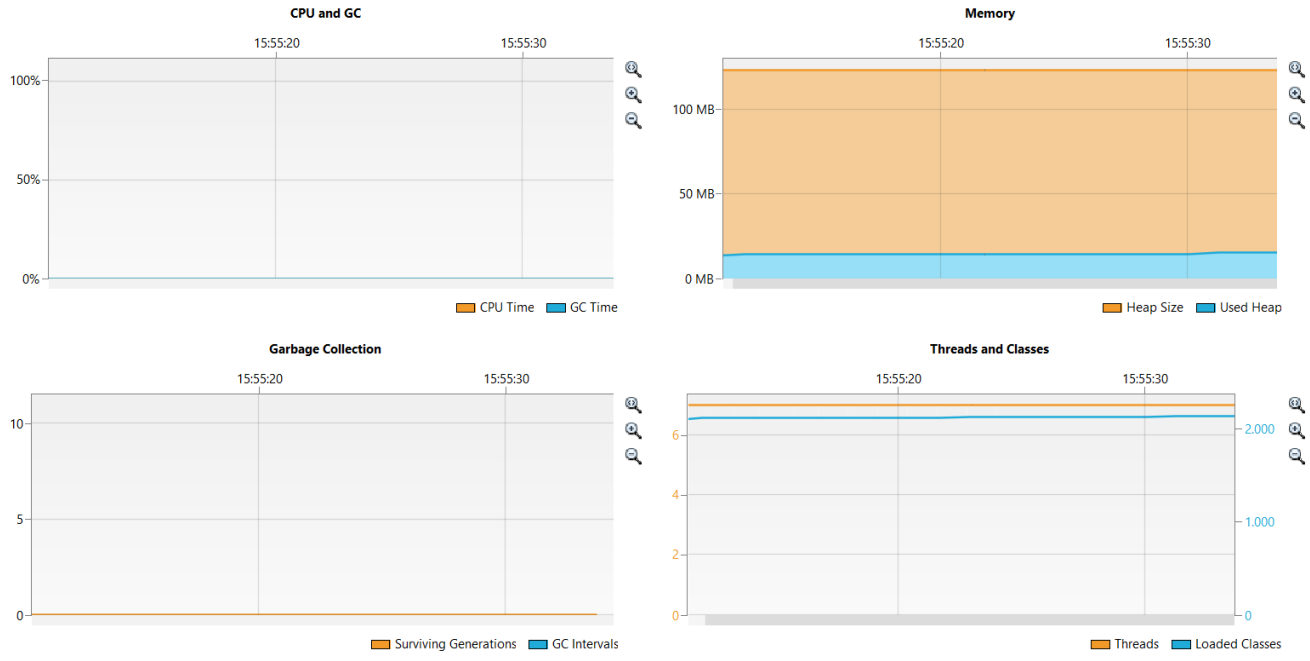
Tempo de Execução:

```
Digite uma posição para descobrir o seu dado:
                                          Digite um número para deletar da árvore:
618
Número 618 encontrado no node com altura 5    618
Tempo de execução da busca: 2478 ms           Tempo de execução da exclusão: 1230 ms

896
879
901
931
925
987
971
Tempo de execução da impressão: 1 ms
```

AVL – 500 números

Profile do NetBeans:



Tempo de Execução:

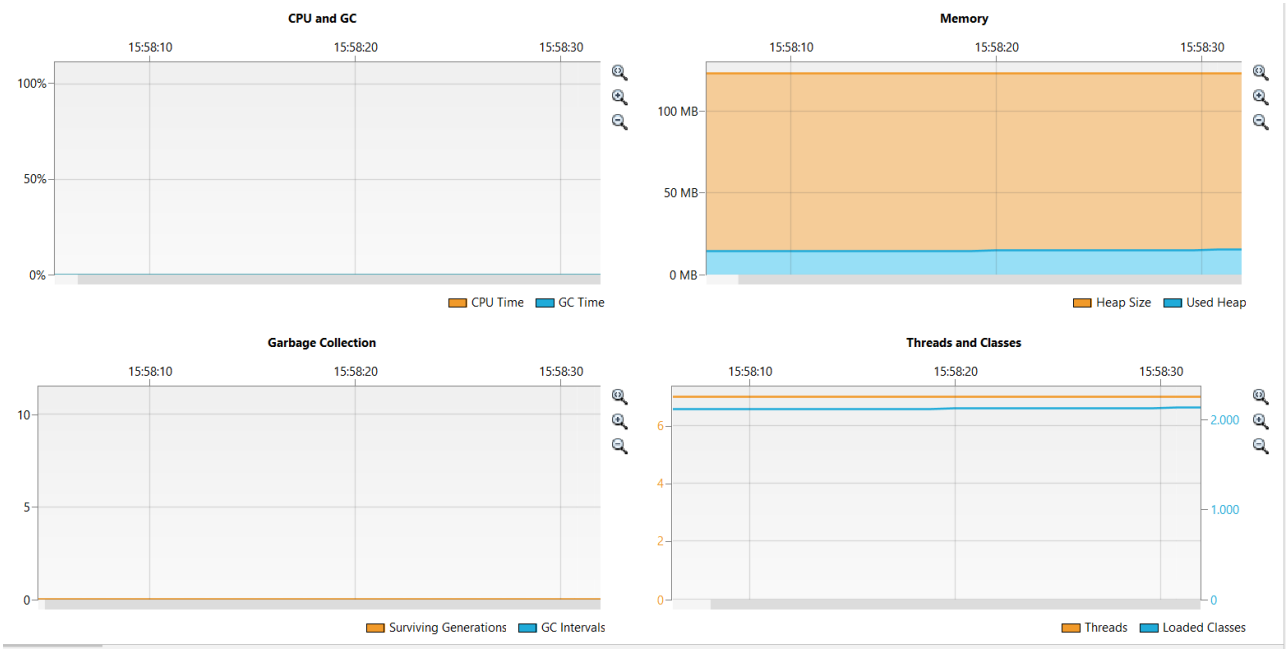
```
Digite uma posição para descobrir o seu dado:
9257
Número 9257 encontrado no node com altura 3
Tempo de execução da busca: 1642 ms
```

```
Digite um número para deletar da árvore:
9257
Tempo de execução da exclusão: 954 ms
```

```
9819
9855
9862
9934
9922
9941
9950
Tempo de execução da impressão: 15 ms
```

AVL – 1000 números

Profile do NetBeans:



Tempo de Execução:

```
Digite uma posição para descobrir o seu dado:

9528
Número 9528 encontrado no node com altura 1
Tempo de execução da busca: 1283 ms
```

```
Digite um número para deletar da árvore:

9528
Tempo de execução da exclusão: 1551 ms
```

```
  99339
  99225
  99193
  99314
  99279
  99628
  99607
  99392
  99718
  Tempo de execução da impressão: 14 ms
```

AVL – 10.000 números

Profile do NetBeans:

Tempo de Execução:

```
Digite uma posição para descobrir o seu dado:
99242
Número 99242 encontrado no node com altura 3
Tempo de execução da busca: 2366 ms
```

```
Digite um número para deletar da árvore:
99242
Tempo de execução da exclusão: 1356 ms
```

```
99946
99952
99988
99972
99968
99974
99993
99992
99996
Tempo de execução da impressão: 138 ms
```

AVL – 20.000 números

Profile do NetBeans:

Tempo de Execução:

```
Digite uma posição para descobrir o seu dado:
997272
Número 997272 encontrado no node com altura 2
Tempo de execução da busca: 1253 ms
99959
99964
99983
99982
99973
99993
99991
99995
Tempo de execução da impressão: 460 ms
```

```
Digite um número para deletar da árvore:
997272
Tempo de execução da exclusão: 1299 ms
```

Análise Crítica

Desempenho: Vemos com os testes feitos acima, na árvore binária na parte de inserção na árvore foi mais demorada em todos os tamanhos, pela sua simplicidade e a sem ter a necessidade de balanceamento, a inserção da prória é mais rápida, na AVL é um pouco mais lenta, mas pelo fator de balanceamento que possui a mais que a binária é compreensivél.
Já no aspecto comparando a busca e a exclusão, notamos na árvore binária que é bem mais demorado a busca do que a exclusão, pela parte do meu código, acredito que seja pelo método diferente e dependente do nó em que o úsuaria está pesquisando, o quanto mais distante do começo da árvore, o mais demorado vai ser, e na parte de deletar, é mais direto por pegar o valor do número do que a posição, tambêm dependendo o quão distante o dado está do começo da árvore. Na árvore AVL tem a mesma ideia, quanto menor a altura do node, mais rápido será encontrado o dado, aonde a exclusão será mais rápide pela maneira mais direta em que o dado é encontrado.

Árvore Binária: A árvore binária é a mais simple, não tem balanceamento que nem a árvore AVL, mas serve seu propósito se for uma árvore mais simples e pequena, como o começo do ensino de árvores em um ambiente escolar.

Árvore AVL: A árvore AVL é a mais complexa tendo um código com mais linhas que a árvore binária, com o rebalanciamento depois de cada adição e exclusão de dados, é mais eficiente em árvore maiores ou até menores, dependendo no nívem de modificação do usuário.

Conclusão:

Na minha opinião, a árvore AVL é a melhor em todos os aspectos necessários para códigos mais complexos e mais utilizados, a árvore binária serve bem como maneira introdutória para o aprendizado de árvores, mas não é possível escolher ela se a questão for melhor desempenho e execução da aplicação, mesmo que seja uma árvore que precisa de um código maior e mais tempo de execução na parte de inserir e deletar, no final vale a pena os milissegundos a mais. Na grande maioria dos casos, acredito que a árvore AVL seja a melhor escolha.