Nathann Zini dos Reis 19.2.4004

Prova 02.

Questão 1.

Divisão e Conquista: Quick Sort

Vetor A = {4, 0, 7, 9, 1, 3, 5, 2, 6, 8}

Passo 1. - Escolher um piro de comparação. Pega a última posição,

A = | 4 | 0 | 7 | 9 | 1 | 3 | 5 | 2 | 6 | 8 |
0 1 2 3 4 5 6 7 8 9

Piro ← A[9]
i ← -1
j ← 0

Passo 2 - Varrer o vetor a partir do piro

i j →
(I) | 4 | 0 | 7 | 9 | 1 | 3 | 5 | 2 | 6 | 8 | // i representa o range do vetor menor que o piro e j o range maior

// A[j] ≤ Piro → i++

i j
(II) | 4 | 0 | 7 | 9 | 1 | 3 | 5 | 2 | 6 | 8 |

⋮

i j
(V) | 4 | 0 | 7 | 9 | 1 | 3 | 5 | 2 | 6 | 8 | // Para A[j] ≤ piro ele é trocado com o primeiro valor do range j que representa os maiores que o piro e aumenta o range dos menores

FORONI

/ / Nathann Zini dos Reis

Seg | Ter | Qua | Qui | Sex | Sáb | Dom

19.2.4007

(VI) [4|0|7|1|9|3|5|2|6|8] — i, j

(VII) [4|0|7|1|3|9|5|2|6|8] — i, j

(VIII) [4|0|7|1|3|5|9|2|6|8] — i, j

(IX) [4|0|7|1|3|5|2|9|6|8] — i, j

(X) [4|0|7|1|3|5|2|6|9|8] — i, j

// Para o último passo, como o pivô é o valor intermediário dos dois ranges, ele vai ser colocado após o final do range à esquerda, trocando os valores.

final [4|0|7|1|3|5|2|6|8|9] — i p i

// Essa função retornará o índice do vetor e chamará, de maneira recursiva, a mesma função para analisar o vetor A em 2 menores ranges:

A{início, ..., pivo-1} e A{Pivo+1, ..., final}

Os vetores finais serão:

Pivo 6 → [4|0|1|3|5|2|6|7]

Pivo → 2 → [0|1|2|4|3|5]

Pivo 1 → [0|1]

Pivo 5 [4|3|5]

Pivo 3 [3|4]

FORONI

Seg | Ter | Qua | Qui | Sex | Sáb | Dom

/ /

Nathann Zini dos Reis 19.2.4007

// Com isso, obtém-se o vetor ordenado em ordem crescente.

// Para verificar o pior caso, que ocorrerá quando o vetor já é ordenado, basta inicialmente embaralhar o vetor, evitando, assim que esteja ordenado.

Divisão quadrática: Bubble Sort

$A = \{4, 0, 7, 9, 1, 3, 5, 2, 6, 8\}$

// Em cada passo, analisa-se a posição do valor com a posição posterior, vai ser maior que a próxima, troca.

Faz isso para todas posições

i) [4|0|7|9|1|3|5|2|6|8]

ii) [0|4|7|9|1|3|5|2|6|8]

iii) [0|4|7|9|1|3|5|2|6|8]

iv) [0|4|7|9|1|3|5|2|6|8]

v) [0|4|7|1|9|3|5|2|6|8]

⋮

/ / Nathann Zini dos Reis

Seg Ter Qua Qui Sex Sáb Dom

19.2.4007

(ex)

0	4	7	1	3	5	2	6	9	8

(x)

0	4	7	1	3	5	2	6	8	9

// Ao final dessa execução, obtem-se o maior valor do vetor na última posição.

Agora executa-se mesmo passo a passo com o range de 0 até n-i (i vai de 0 até n)

Ao final das n execução do valor i, obtem-se o vetor ordenado em ordem crescente.

// Para verificar o pior caso, que é quando o vetor já está ordenado, basta criar uma variável auxiliar para contabilizar o número de trocas. Ao haver 0 trocas, o vetor já é ordenado e pode interromper a execução.

⇒ O método Quick Sort não é estável, enquanto o BubbleSort é estável.

O quickSort é muito indicado para vetores de grande tamanho, enquanto o bubbleSort, embora eficaz, não é muito eficiente por ser muito custoso.

Escolhi os dois pelo contraste de eficácia

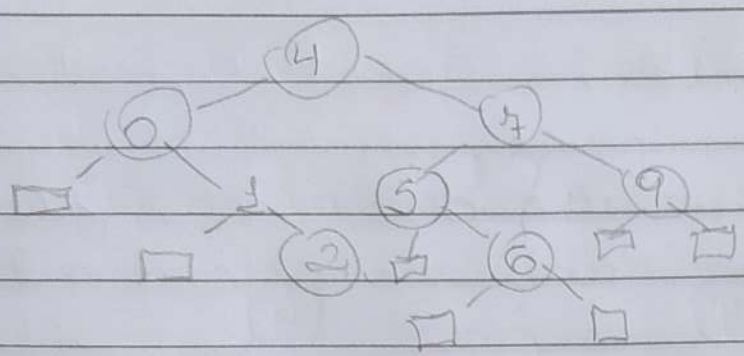
/ /

Seg Ter Qua Qui Sex Sáb Dom

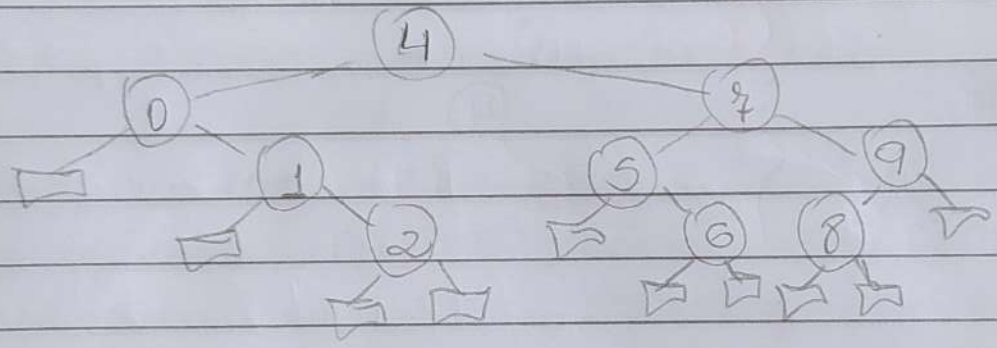
Nathann Zini dos Reis

19.2.4007

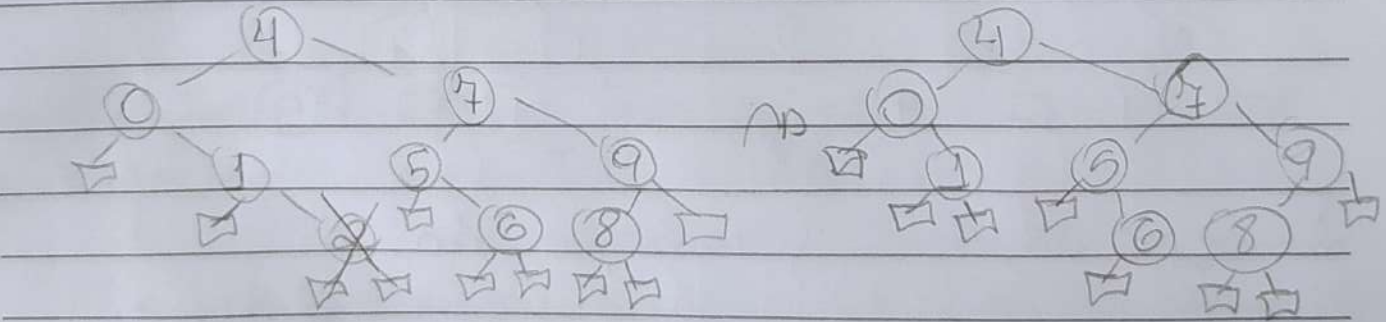
Inserção do 6



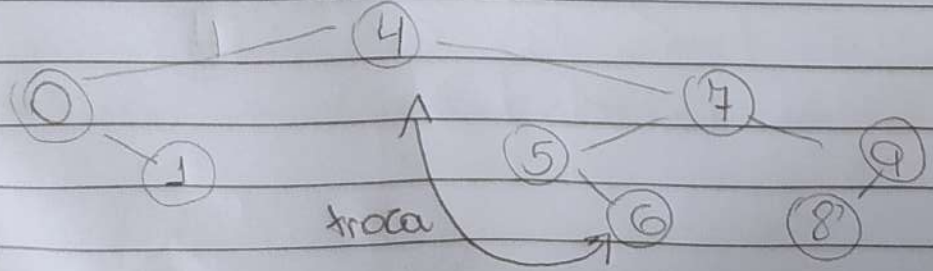
Inserção do 8



Remoção do 2



Remoção do 4 (MIN)

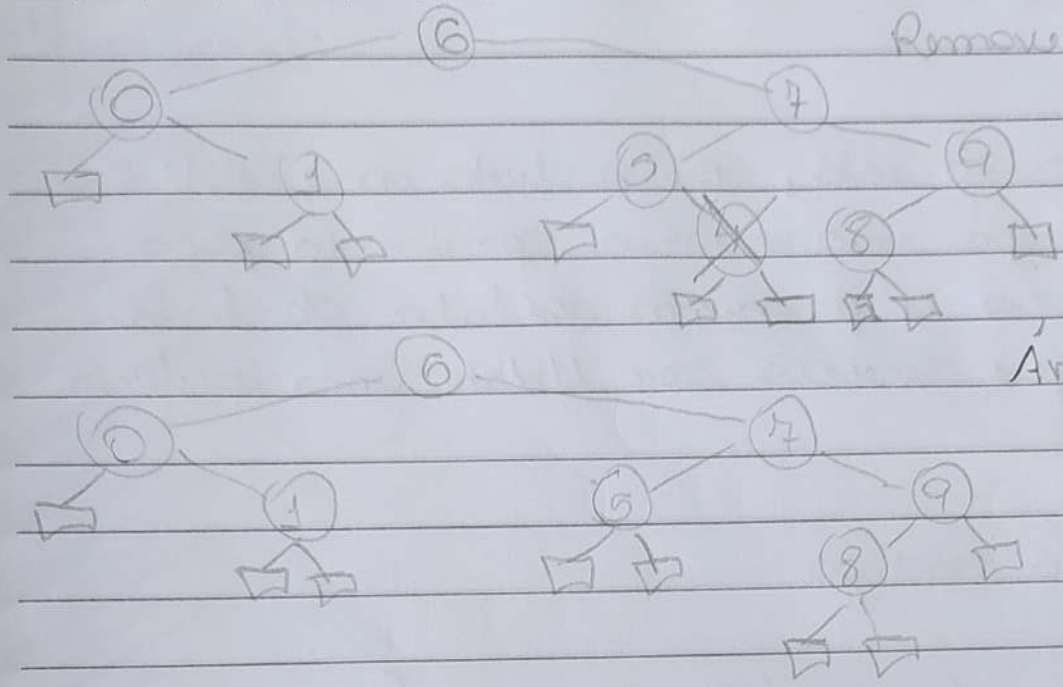


FORON

Seg Ter Qua Qui Sex Sáb Dom

/ /

Nathann Zini dos Reis 19.2.4007



Remove o 4

Árvore final

Seg | Ter | Qua | Qui | Sex | Sáb | Dom

/ /

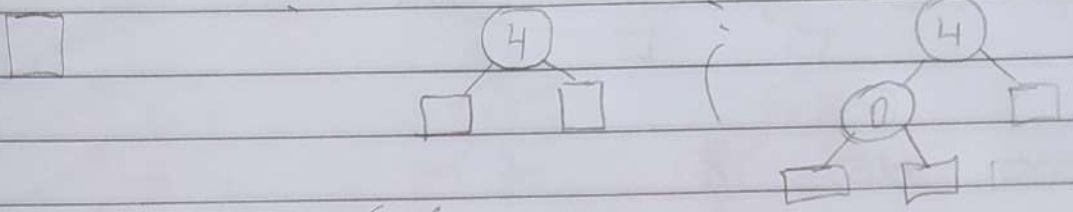
Nathann Zini dos Reis

19.2.4007

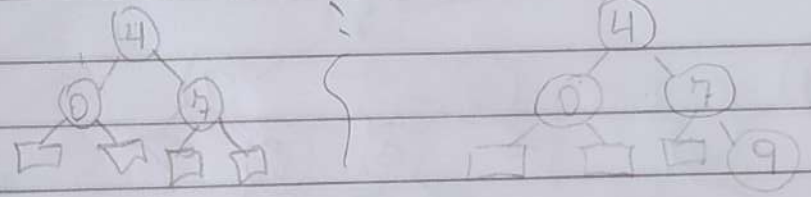
Questão 2.

Valor M = {4, 0, 0, 7}

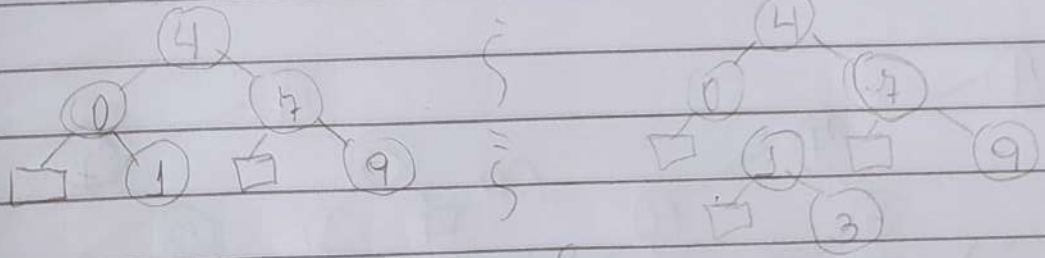
Árvore vazia ; Inserção do 4 ; Inserção do 0



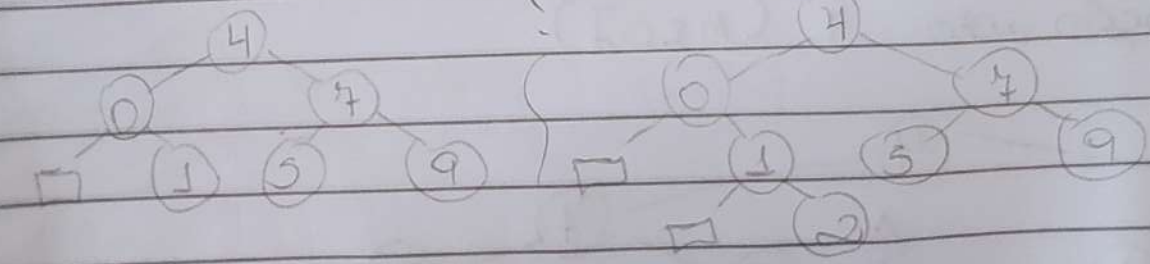
Inserção do 7 ; Inserção do 9



Inserção do 1 ; Inserção do 3



Inserção do 5 ; Inserção do 2



Seg Ter Qua Qui Sex Sáb Dom

/ /

Nathann Zini dos Reis

Questão 3. 19.2.4007

$E = H(C) = ((C \text{ div } 50) + 1) \bmod 10$

$C = 4007$ $\quad 4007 / 50 = 80$

$80 + 1 = 81$

$81 \bmod 10 = 1$ //

$H(C) = 1$

// Como a posição 1 está preenchida, vai percorrer o vetor e inserir na próxima posição livre.

O endereçamento aberto funciona com a inserção de um elemento na posição que será definida por um calculo mod tamanho da tabela. Caso a posição não esteja vazia, pega a proxima vazia.

No pior caso, vai adicionar na ultima posição da tabela, ou seja o custo assintótico vai ser O(n), pois terá que percorrer toda a tabela.

1 1

Seg Ter Qua Qui Sex Sáb Dom

Nathann Zini dos Reis 19.2.4007

Questão 4

Do modo como está, caso a lista seja vazia, a cabeça não retorna apontando para NULL, mas para a próxima célula da lista. E o fim, que deveria ser NULL, não recebeu NULL.

Correção do código

```
void Tlista_limpar (Tlista * l) {

Tcelula *aux = l -> cabeça;

While (aux != NULL) {

  Tcelula *prox = aux -> prox;
  free (aux);
  aux = prox;

}

l -> fim = l -> cabeça;
l -> tam = 0;

}
```