Grupo I

1.1 [1] Utilizando a definição, prove que $f(n) = (n + 3)^2$ é $O(n^2)$.

1.1 [1]
Utilizando a definição, prove que f(n) = (n + 3)² é O(n²).

Prova que f(n) = (n + 3)² é O(n²)

Definição: f(n) é O(g(n)) se existirem constantes positivas c e n_0 tais que:
$f(n) \leq c \cdot g(n), \forall\, n \geq n_0$

Passo 1 — Expandir f(n)
$(n+3)^2 = n^2 + 6n + 9$

Passo 2 — Majorar cada termo em função de n²
Para $n \geq 1$, temos que $n \leq n^2$ e $1 \leq n^2$, portanto:
$6n \leq 6n^2$
$9 = 9 \cdot 1 \leq 9n^2$

Passo 3 — Construir a desigualdade
Somando tudo:
$n^2 + 6n + 9 \leq n^2 + 6n^2 + 9n^2 = 16n^2$

Passo 4 — Identificar as constantes
Das desigualdades acima, retiramos diretamente:
$c = 16\; n_0 = 1$

Conclusão
Com $c = 16$ e $n_0 = 1$, verificamos que:
$(n+3)^2 \leq 16n^2, \forall\, n \geq 1$

Pela definição de notação O, fica provado que: **f(n) = (n + 3)² é O(n²)**

1.2.1 [0.5] Para o seguinte par de funções $f(n)$ e $g(n)$, indique (apenas uma opção) se $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$ ou nenhum dos casos. Justifique a sua resposta com base apenas na ordem de grandeza relativa dos termos das funções.

$$f(n) = n^2 + 1,\ g(n) = \sqrt[5]{n^5} + 100$$

Nós temos que g(n) = n^(5/3) + 100

Comparando os termos denominadores, nós temos que:

Para f(n) = n^2 +1 -> n^2

Para g(n) = n^(5/3) + 100 -> n^(5/3)

Sabemos tambem que, n^2 > n^(5/3) quando n tende para infinito, onde temos o seguinte limite da razão:

limite (n -> infinito) f(n)/g(n) = limite_(n -> infinito) (n^2 + 1)/(n^(5/3) + 100) = infinito

por isso ficamos com:

limite_(n -> infinito) f(n)/g(n) = infinito => f(n) = w(g(n)) => f(n) = omega (g(n))

e por isso, podemos concluir então que a opção correta seria:

f(n) = omega (g(n))

1.2.2 [0.5] Para o seguinte par de funções $f(n)$ e $g(n)$, indique (apenas uma opção) se $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$ ou nenhum dos casos. Justifique a sua resposta com base apenas na ordem de grandeza relativa dos termos das funções.

$$f(n) = n^{4 \log n} + 1000,\ g(n) = n^{\log n^2}$$

Termos dominantes:

f(n) = n^(4log n) + 1000 -=> Ø (n^(4 log n))

g(n) = n^(log n^2) = n^(2 log n) => Ø(^(2 log n))

Nós temos que a funçao f(n) = n^(4log n) + 1000 cresce mais rapidamente do que g(n) = n^(log n^2), e que o termo dominante de f(n) é assintoticamente maior que o termo dominante de g(n)

Comparando então as ordens, temos que:

f(n) = n^(4 log n) / n^(2 log n) = n^(4 log n - 2 log n) = n^(2 log n)

Sabemos também que, n^(2 log n) -> infinito, quando, n -> infinito = limite_ (n->infinito) => limite_ (n->infinito) f(n)/g(n) = infinito

Concluímos então que f(n) = omega(g(n))

1.3 [1] Considere a complexidade do seguinte segmento de código em termos do nº $f(n)$ de operações aritméticas realizadas na variável a. Determine a expressão de $f(n)$ e indique a sua complexidade na notação $O(.)$.

```
for(a=0,i=1; i<=n; ++i)
  for(j=1; j<=i; ++j)
    a++;
```

Nós temos dois loops aninhados, um loop externo que vai de i = 1 até n, com n iterações e um loop intenro que vai de j = 1 até i, ou seja, o número de iterações depende do i.

Para cada i de 1 a n, o loop interno executa i vezes, logo temos:

Total de incrementos de a ao longo da execução do código é igual á soma de i de 1 até n, o que corresponde á formula da soma dos primeiros números inteiros positivos: (n(n+1))/2

Concluímos entao que a função será:

f(n) = (n(n+1))/2 = O(n^2)

Grupo II

2.1 Considere uma árvore de pesquisa binária (BST) inicialmente vazia.

2.1.1 [1] Insira na árvore as chaves 6, 11, 9, 2, 10, 13, 7, 1, 4, 8 pela ordem indicada. Desenhe a árvore obtida após efetuadas todas as inserções (total de 1 desenho). Justifique os passos intermédios / raciocínio apenas para as duas últimas inserções. Nas alíneas seguintes considere a árvore obtida como a árvore "original".

---

6 — 1º) Insersão do 6 - Raiz

/ \ — 2º) Insersão do 11 - 11 > 6, filho direito de 6

2 11 — 3º) Insersão do 9 - 9 > 6, vai para a sub árvore direita, 9 < 11, filho esquerdo de 11

/ \ / \ — 4º) Insersão do 2 - 2 < 6, filho esquerdo de 6

1 4 9 13 — 5º) Insersão do 10 - 10 > 6, sub árvore direita, 10 < 11, 10 > 9, 10 é filho direito de 9

/ \ — 6º) Insersão do 13 - 13 > 6, 13 > 11, 13 é filho direito de 11

7 10 — 7º) Insersão do 7 - 7 > 6, 7 < 11, 7< 9,7 é filho esquerdo de 9

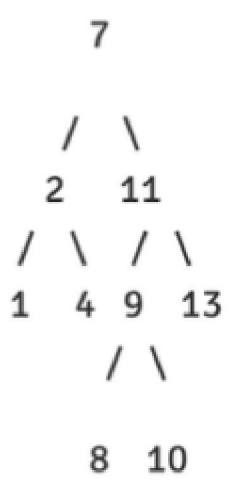
\ — 8º) Insersão do 1 - 1 < 6, 1 < 2, 1 é filho esquerdo de 2

8 — 9º) Insersão do 4 - 4 < 6, 4 > 2, 4 é filho direito de 2

10º) Insersão do 8

2.1.2 [1] Remova da árvore original a chave 6 utilizando o algoritmo de remoção por cópia (Deletion by Copying). Das várias opções possíveis escolha a que lhe parece melhor. Desenhe a árvore obtida justificando a sua escolha e os passos intermédios / raciocínio.

No algoritmo de remoção por cópia, quando se remove um nó com dois filhos, substitui-se o valor do nó a remover pelo valor do seu antecessor (maior valor da subarvore esquerda) ou sucessor (menor valor da subarvore direita), removendo-se depois o nó de onde se copiou o valor.
Para remover o nó 6 (raiz) existem duas opções:
1. Substitui pelo antecessor: o maior valor da subárvore esquerda, que é 4.
2. Substituir pelo sucessor: o menor valor da subarvore direita, que é 7.
Neste caso deve-se optar por 7 porque o nó 7 tem apenas um filho (8), o que torna a sua remoção mais simples.
Para remover tem-se:
1. Copia-se o valor de 7 para a posição da raiz (substituindo o 6).
2. Remove-se o nó original que continha o 7.
Como o nó 7 tem um filho direito (8), ao remove-lo, o seu filho sobre para ocupar a sua posição.
A árvore final fica:



2.1.3 [1] Explique em que consiste a operação de rotação de um nó em torno do seu parente. Efetue na árvore original uma rotação de 11 em torno do seu parente. Desenhe a árvore obtida justificando os passos intermédios / raciocínio.

A rotação de um nó em torno do seu parente é uma operação local que reequilibra a subárvore, e muda ligeiramente a sua estrutura sem violar as propriedades de vusca binária, existem dois tipos principais:

- Rotação à esquerda sobre um nó X, em que Y é filho direito de X ou seja o pivot, a subárvore esquerda de Y é transplantada para passar a ser o filho direito de X, e X passa a ser o filho esquerdo de Y.

- Rotação à direita é como a rotação à esquerda, mas troca-se a direita pela esquerda e viceversa.

11 — 1º) Identificar X e Y

/ \ — X = nó 6 , onde a rotação acontece, Y = nó 11, pivot, filho de X

6 13 — 2º) Transplante da árvore da esquerda de Y

/ \ — Y tem subárvore esquerda em 9, esta subárvore torna-se filho direito de X

2 9 — 3º) Mover X como filho esquerdo de Y

/ \ / \ — Se Y adota X como filho esquerdo, e X era a raiz

1 4 7 10 — 4º) Realocar as restantes subárvores inalteradas

\ — O filho esquerdo órigina de X o 2, permanece como filho esquerdo de X, e a subárvore direita de Y

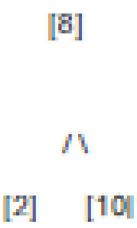
8 — o 13 continua como filho direito de Y

2.2 [2] Considere uma árvore B (B-Tree) de ordem 3 inicialmente contendo apenas o nó raiz com as chaves 2 e 10. Desenhe a árvore após cada uma das seguintes operações de inserção (I) e remoção (R) pela ordem indicada: I 8, 14, 12, 4, 6; R 10; (total de 6 desenhos). Justifique os passos intermédios / raciocínio para cada operação.

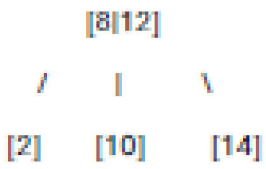
Temos uma arvore B de ordem 3 (m=3). Máximo de chaves por nó = 2. Mínimo de chaves por nó (não raiz) = 1. Estado inicial: [2|10]

Inserir 8: O i deve ser inserido no nó raiz, entre 2 e 10. O nó fica com 3 chaves [2|8|10], o que causa um split. A chave do meio (8) é promovida para se tornar a nova raiz.

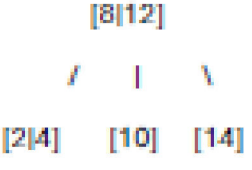
Inserir 14: Começamos na raiz 8. 14 > 8 então descemos para o filho direito (10). O 14 é inserido neste nó que tem capacidade.



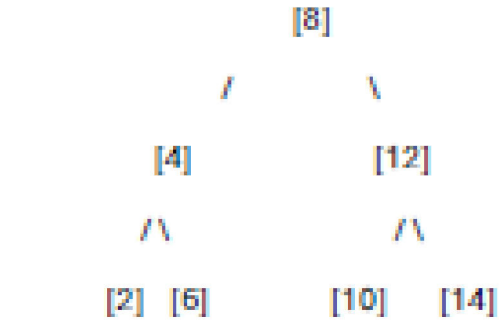
Inserir 12: Começamos na raiz i. 12 > 8, descemos para o filho direito [10|14]. A inserção de 12 resulta no nó temporário [10|12|14]. Esta nó tem 3 chaves e precisa de um split. A chave do meio (12) é promovida ao nó pai (8) que se tonra [8|12]



Inserir 4: Começamos na raiz [8!12]. 4 < 8 então descemos para o filho esqueerdo [2]. O 4 é inserido neste nó, que tem capacidade.



Inserir 6: Começamos na raiz [8|2]. 6 < 8, descemos para filho [2|4]. A inserção de 6 resulta no nó [2|4|6] que sofre um split. A chave do meio (4) é promovida para o pai [8|12], que se tonra [4|8|12]. Este nó pai tamb+em excede a capacidade máxima e sofre um split. A chave do meio 8 torna-se a nova raiz.



Remover 10: O 10 está num nó folha [10] que tem o numero minmo de chaves (1). A remoção causa um underflow. O seu irmão 14 tambem tem o minimo de chaves, então uma rotação não é possivel. Temos de fazer uma fusão (merge).

- fundimos o nó 10 (agora vazio) com o seu irmão 14 e a chave sepradora do pai (12). Isto cria o ovo nó [12|14] e esvazia o n ó pai [12]. - O nó [12] (agora vazio) causa um underflow de nivel superior. O seu irmão 4 tem o minimo de chaves então temos de fundir novamente. - Fundimos o nó vazio, o irmão [4] e a chave separadora da raiz (8). A raiz [8] fica vazio e é removida, diminuindo a altura da arvore. O nó fundido [4|8] torna-se a nova raiz, adotando os filhos dos nós fundidos.

Grupo III

3. Considere o vetor [8 6 9 2 1 3 5 4 7]. Ordene o vetor utilizando o algoritmo de ordenação indicado. Explique de um modo geral o funcionamento do algoritmo. Indique justificando a sequência de vetores obtida correspondente às iterações principais do algoritmo.

3.1 [2] Algoritmo de ordenação por inserção (Insertion Sort).

o algoritmo insertion sort ele percorre o vetor da esqueda para a direita e vai inserindo cada elemento na posição correta do suvetor á esqueda que já está ordenado. Por exemplo:

[**8** 6 9 2 1 3 5 4 7]

[6 **8** 9 2 1 3 5 4 7]

[6 8 **9** 2 1 3 5 4 7]

[2 6 8 9 **1** 3 5 4 7]
[1 2 6 8 9 **3** 5 4 7]
[1 2 3 6 8 9 **5** 4 7]
[1 2 3 5 6 8 9 **4** 7]
[1 2 3 4 5 6 8 9 **7**]
[1 2 3 4 5 6 7 8 9]

Ficando assim com o vetor e cada elemento na sua posição correta

3.2 [2] Algoritmo de ordenação rápida (Quick Sort).

O algoritmo quick sort escolhe um pivô e separa os elementos menores á esquerda e os maiores á direita, e aplica o meso recursivamente:

Por exemplo:

pivô 7

menores que 7 -> [6 2 1 3 5 4]

maiores que 7 -> [8 9]

nova ordem: [6 2 1 3 5 4 7 8 9]

subvetor [6 2 1 3 5 4] - pivô 4

maiores que 4 -> [6 5]

menores que 4 -> [2 1 3]

nova ordem: [2 1 3 4 6 5]

subvetor [2 1 3] -> pivo 3

menores que 3 -> [2 1]

nova ordem : [1 2 3]

subvetor 4 [6 5] - pivo 5

maiores que 5 -> [6]

nova ordem [5 6]

resultado final:

[1 2 3 4 5 6 7 8 9]