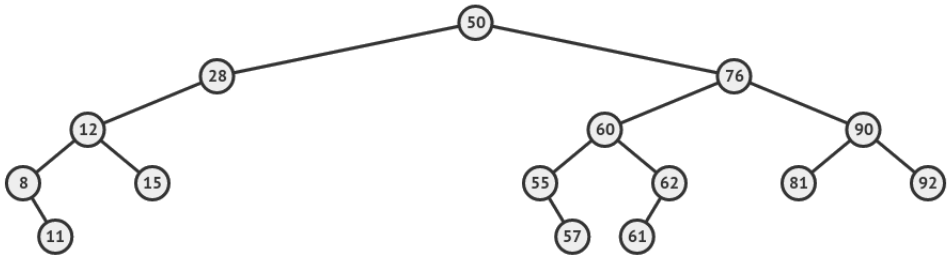
Pesquisa e Classificação de Dados
Lista 3(BST e AVL)

Prof. Ricardo Oliveira

OBS: Apenas o resultado final é apresentado. Realize também o desenvolvimento do exercício!

OBS2: Caso encontre algum erro, por favor avise o professor.

1.

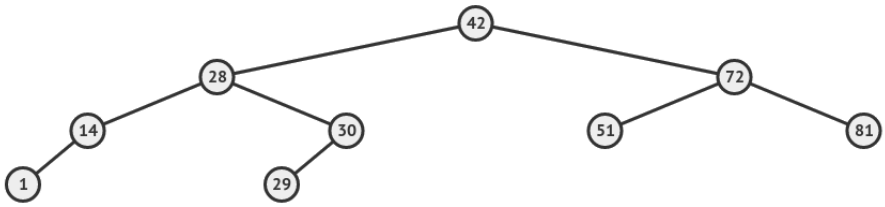


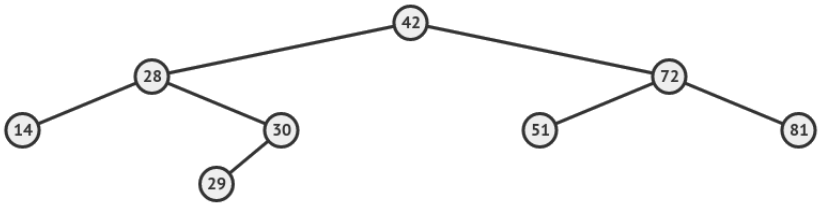
2. (a) 50
   (b) 11, 15, 57, 61, 81 e 92
   (c) 5
   (d) 3
   (e) 1
   (f) 0
   (g) 3
   (h) 0
   (i) 60 e 90
   (j) 11
   (k) não há
   (l) 12
   (m) 50
   (n) não há
   (o) 60
   (p) 50
   (q) 90
   (r) 8
   (s) não há
   (t) 15
   (u) 81 e 92
   (v) 12
   (x) 55, 60, 76, 50
   (w) 12, 28, 50
   (y) 8, 15, 11
   (z) 60, 90, 55, 62, 81, 92, 57, 61
   (a1) `50,28,12,8,11,15,76,60,55,57,62,61,90,81,92`
   (b1) `8,11,12,15,28,50,55,57,60,61,62,76,81,90,92`
   (c1) `11,8,15,12,28,57,55,61,62,60,81,92,90,76,50`

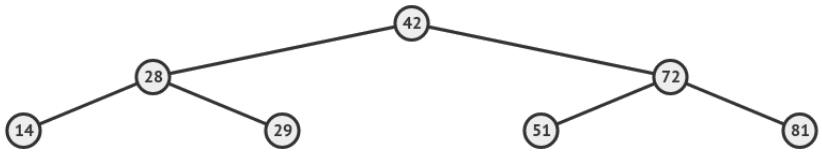
3. (a) 60
   (b) 60
   (c) 12
   (d) 50
   (e) 76
   (f) 76
   (g) 28

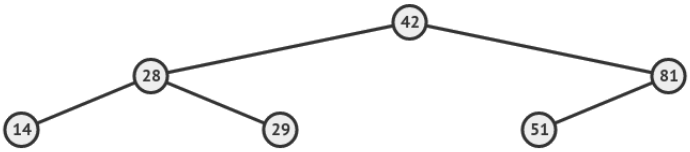
4. Uma possível solução é: mantenha dois ponteiros, inicialmente para os dois nodos dados. Troque um deles pelo seu pai ("subindo") até que ambos estejam no mesmo nível. Então, "suba" ambos simultaneamente até que apontem para o mesmo nodo. Retorne este nodo. Este algoritmo é $O(h)$, ou $O(N)$ no pior caso (árvore desbalanceada).

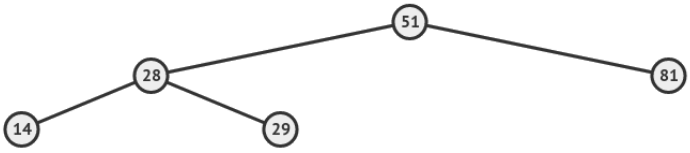
5. Leia `https://www.topcoder.com/community/competitive-programming/tutorials/range-minimum-query-and-lowest-common-ancestor/`.

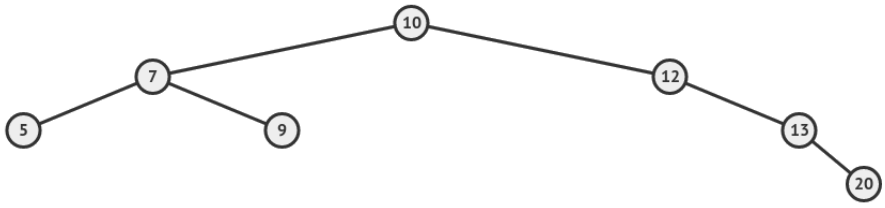
6. 

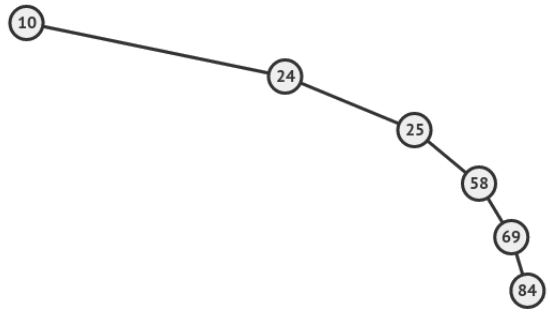
7. (a) 

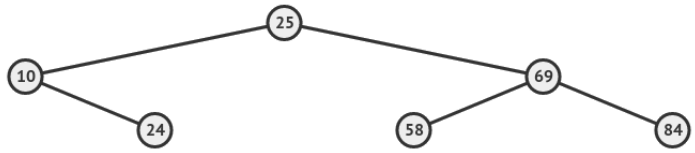
(b) 

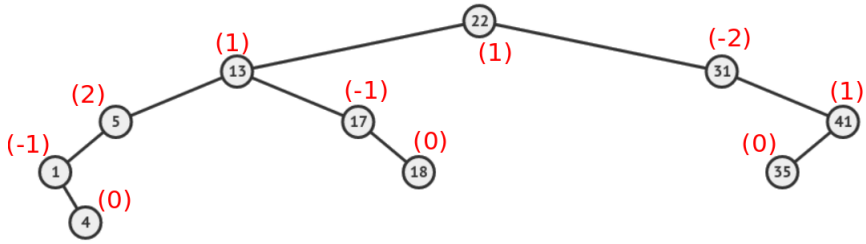
(c) 

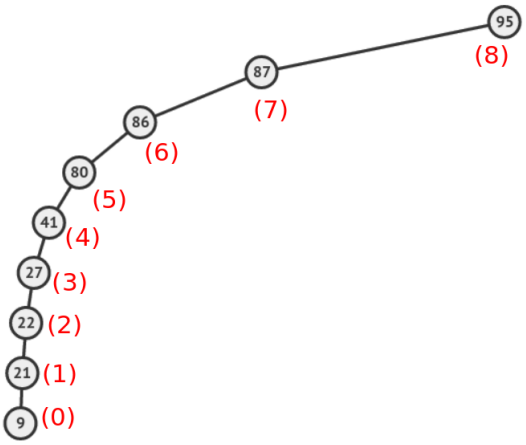
(d) 

8. 

9. 

10. Ordene o vetor; Insira os elementos na BST um a um, em ordem.
11. Ordenação: $O(N \lg N)$ (se feita por comparação). N inserções de tempo $O(N)$ cada; Total: $O(N \lg N + N^2) = O(N^2)$. É possível alterar o algoritmo para, após a ordenação, construir a BST em $O(N)$, mantendo a complexidade total em $O(N \lg N)$ (como?).
12. 

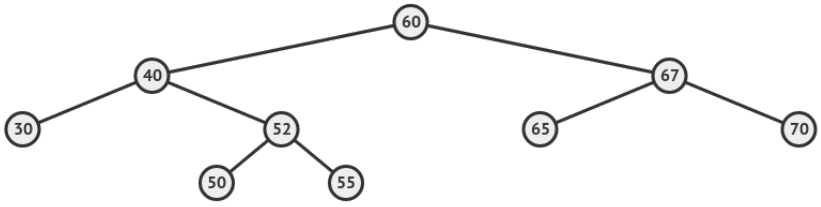
13. Ordene o vetor; Insira como raiz da árvore o elemento no meio do vetor; Construa as subárvores esquerda e direita recursivamente, com a primeira e segunda metade do vetor, respectivamente (análogo à versão recursiva da busca binária).
14. Ordenação: $O(N \lg N)$ (por comparação). A análise da função recursiva indicará a complexidade $O(2^{\lg N} = N)$. Total: $O(N \lg N + N) = O(N \lg N)$.
15. Uma possível solução é: determine o maior elemento em A (ou o menor em B) e faça dele a raiz da nova árvore. O restante da árvore A (resp. B) se torna então a subárvore esquerda (resp. direita) da raiz.
16. Obter o maior elemento de A (resp. o menor de B) tem custo $O(h(A))$ (resp. $O(h(B))$), onde $h(X)$ indica a altura da árvore X. Reconstruir a árvore tem custo $O(1)$. É possível fazer com que todo o processo tenha custo $O(1)$ se ponteiros para os extremos das árvores A e B forem mantidos juntamente com as árvores.
17. (a) 



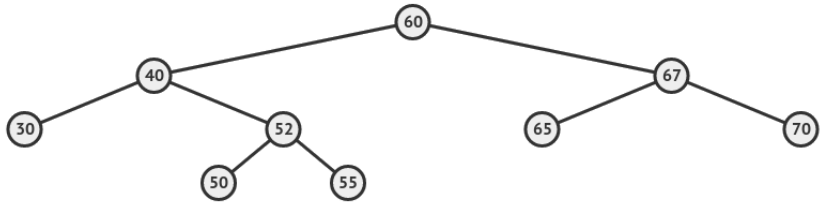
(b)

18. Como discutido em aula.
19. Como ideia, note que a operação de inserção aumenta a altura de qualquer subárvore em, no máximo, 1 (uma) unidade. Desta forma, a diferença entre as alturas de duas subárvores (isto é, o balanceamento de um dado nodo) aumenta em, no máximo, 1 (uma) unidade em módulo. Como está sendo mantida uma árvore AVL, o balanceamento de todo nodo anteriormente à inserção é 0 ou ± 1, o que faz com que, após a inserção (e anteriormente às rotações), o balanceamento seja no máximo $\pm(1+1) = \pm 2$. A operação de remoção é análoga.
20. (a) Por absurdo. Suponha que z tem balanceamento -2. Sejam h_{esq} e h_{dir}, respectivamente, a altura da subárvore esquerda e direita de z após a inserção. Desta forma, tem-se $h_{esq} - h_{dir} = -2$, isto é, $h_{esq} = h_{dir} - 2$. Como o novo elemento foi inserido na subárvore esquerda, tem-se $h_{esq} = h'_{esq} + 1$, onde h'_{esq} denota a altura da subárvore esquerda *antes* da inserção. Desta forma, tem-se $h'_{esq} + 1 = h_{dir} - 2$, isto é, $h'_{esq} - h_{dir} = -3$, indicando que o nodo z tinha balanceamento -3 anteriormente à inserção. Isto é um absurdo conforme exercício anterior.

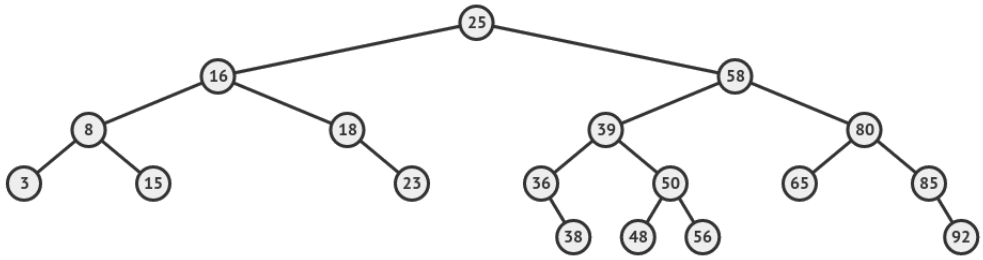
    (b) análogo.
21. – Rotação LL após inserção de 70;
    – Rotação RR após inserção de 30;
    – Rotação LR após inserção de 67;
    – Rotação RL após inserção de 52.



22. Ocorrem rotações LL após as inserções de 3,5,6,7 e 9.



23. Nenhuma rotação é realizada. O resultado é uma árvore cheia.
24. – Rotação LR após inserção de 58;
    – Rotação LL após inserção de 36;
    – Rotação LL após inserção de 85;
    – Rotação RL após inserção de 8;
    – Rotação RR após inserção de 18;
    – Rotação RL após inserção de 39;
    – Rotação LL após inserção de 23;
    – Rotação RL após inserção de 15.



25. Mantenha em cada nodo um campo extra indicando o tamanho (em número de nodos) de toda a (sub)árvore enraizada no mesmo. Desta forma, analise o tamanho das subárvores da raiz: se a subárvore esquerda tem exatamente $k-1$ nodos, retorne a raiz; se a subárvore esquerda tem menos de $k-1$ nodos, analise recursivamente para mesma; caso contrário, analise recursivamente para a subárvore direita (ajustando o valor de k de acordo).
26. O algoritmo acima tem complexidade $O(h)$ que, em uma árvore AVL, é $O(\lg N)$.
27. Depende do algoritmo apresentado. Para o dado acima, o balanceamento da raiz da nova árvore pode ser $h(A) - h(B)$. A árvore resultante não é uma AVL se as alturas das árvores A e B diferem em 2 ou mais unidades.
28. Depende do algoritmo apresentado. Para o dado acima, a mera remoção do maior elemento da árvore A pode torná-la desbalanceada, mesmo sendo uma folha. Considere um caso em que seu pai tem uma subárvore esquerda de altura 2. O balanceamento do seu pai é 1 antes da remoção, mas passa a ser 2 após.