Respostas da lista de rotação e balanceamento:

1)AVL é uma árvore binária balanceada, onde cada um de seus nós possuem fator de balanceamento. Quanto um nó possui " fb " diferente de -1, 0 e 1,
é aplicado uma rotação nesse nó para que a árvore seja balanceada.

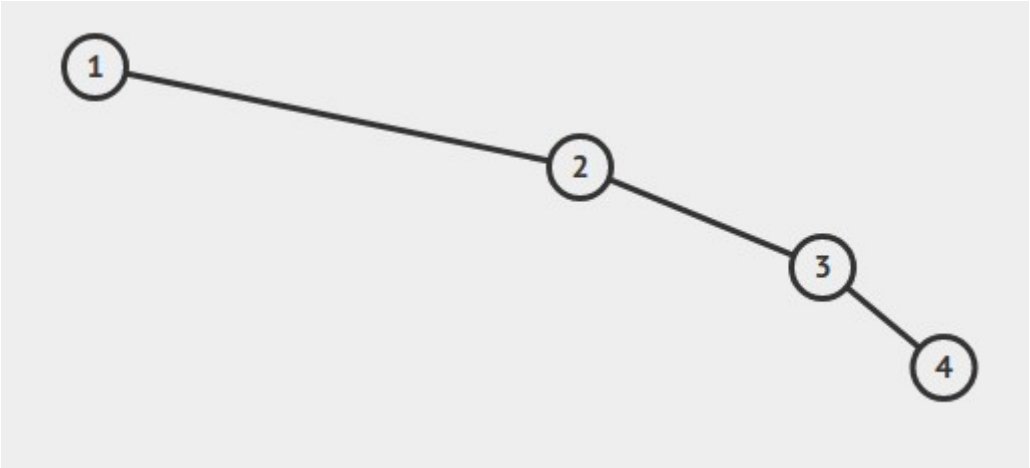
2)
a)Como o vetor de entrada é ordenado em crescente, basta usar o procedimento padrão de inserção em ABP, excluindo a travessia pela esquerda, pois os valores sempre serão
maiores que seu nó pai, logo a árvore crescerá linearmente para a direita.

b)Como o vetor de entrada é ordenado em crescente, se pegarmos o termo do meio do vetor e inserirmos como o raiz, os subvetores restantes, tanto a parte da esquerda quanto
da direita, ao serem inseridos seus elementos a árvore já estará balanceada (procedimento semelhante ao algorítmo de pesquisa do merge sort).

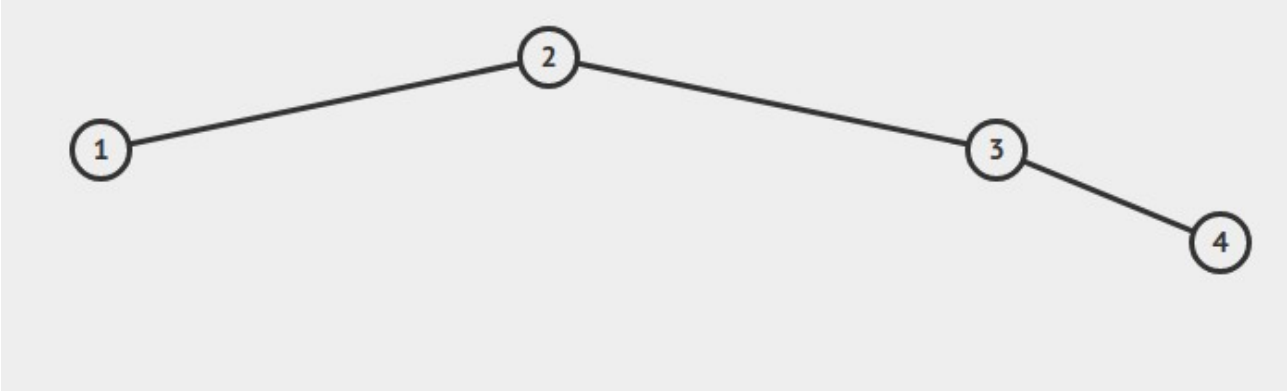
```
typedef struct ArvBinaria {
  int chave;
  struct ArvBinaria *esq;
  struct ArvBinaria *dir;
}NoArvBinaria;

NoArvBinaria *insere_ABP(NoArvBinaria **raiz, int k);
void insere_ABP_balanceada(int ini, int fim, int *v, NoArvBinaria **raiz);
NoArvBinaria *aloca_no(int k);

int main(void) {
  int v[5] = {1, 2, 3, 4, 5}, i = 0, ini = 0, fim = 0;
  NoArvBinaria *raiz = NULL, *raiz2 = NULL;
  printf("=>Inserindo valores do vetor, formando uma ABP degenerada:\n");
  for (i = 0; i < 5; i++) {
    printf("=>Inserindo %d\n", v[i]);
    insere_ABP(&raiz, v[i]);
  }
  printf("=>Inserindo valores do vetor, formando uma ABP totalmente balanceada:\n");
  ini = 0;
  fim = 4;
  insere_ABP_balanceada(ini, fim, v, &raiz2);
}

NoArvBinaria *aloca_no(int k) {
  NoArvBinaria *novoNo = NULL;
  novoNo = (NoArvBinaria*)malloc(sizeof(NoArvBinaria));
  if (novoNo == NULL) return NULL;
  else {
    novoNo->chave = k;
    novoNo->esq = novoNo->dir = NULL;
    return novoNo;
  }
}

NoArvBinaria *insere_ABP(NoArvBinaria **raiz, int k) {
```

```
  if (*raiz == NULL) {//Aloca o raiz
    *raiz = aloca_no(k);
    return *raiz;
  }
  else if ((*raiz)->chave < k) return insere_ABP(&(*raiz)->dir, k);//Insere na direita da arvore
  else return insere_ABP(&(*raiz)->esq, k);//Insere na esquerda da arvore
}

void insere_ABP_balanceada(int ini, int fim, int *v, NoArvBinaria **raiz) {
  if (ini <= fim) {//Verifica se existem elementos no vetor
    int meio = (ini+fim)/2;
    insere_ABP(&(*raiz), v[meio]);//Insere o raiz, sendo este o valor do meio do vetor
    insere_ABP_balanceada(ini, meio-1, v, &(*raiz));//Percorre o subvetor da esquerda
    insere_ABP_balanceada(meio+1, fim, v, &(*raiz));//Percorre o subvetor da direita
  }
}
```

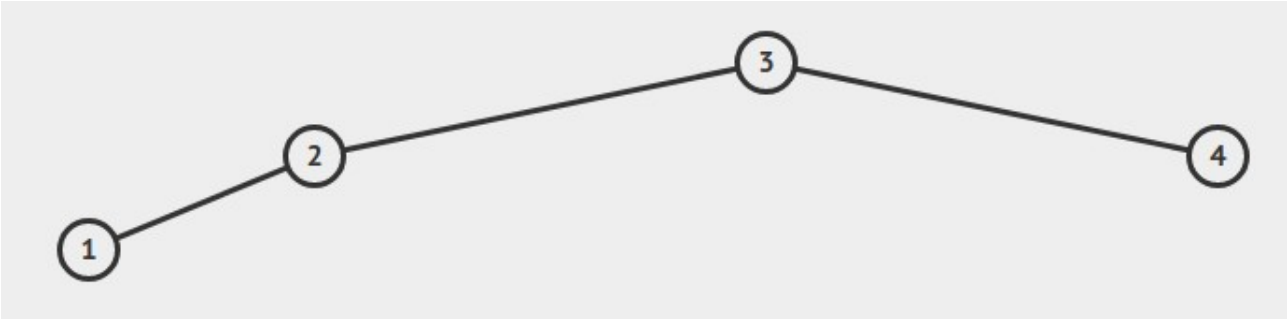
3)
a)ABP com os valores 1, 2, 3, 4:



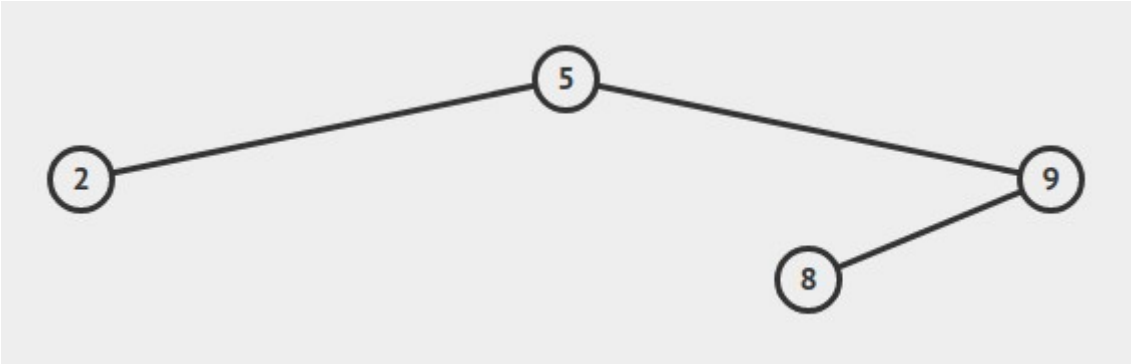
Rotação esquerda no 1:



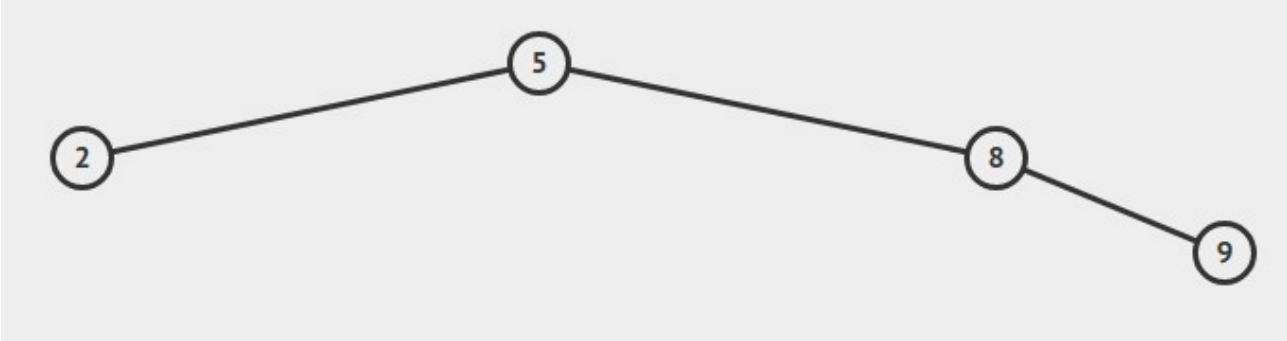
Rotação esquerda no 2:



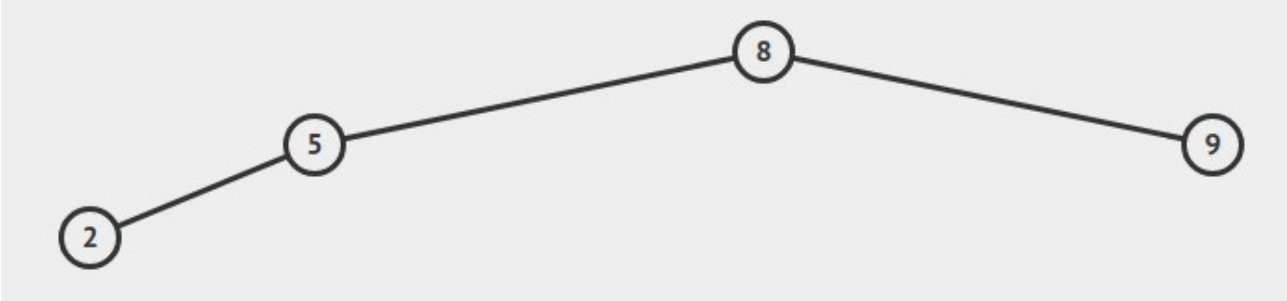
b)ABP com os valores 5, 2, 9, 8:



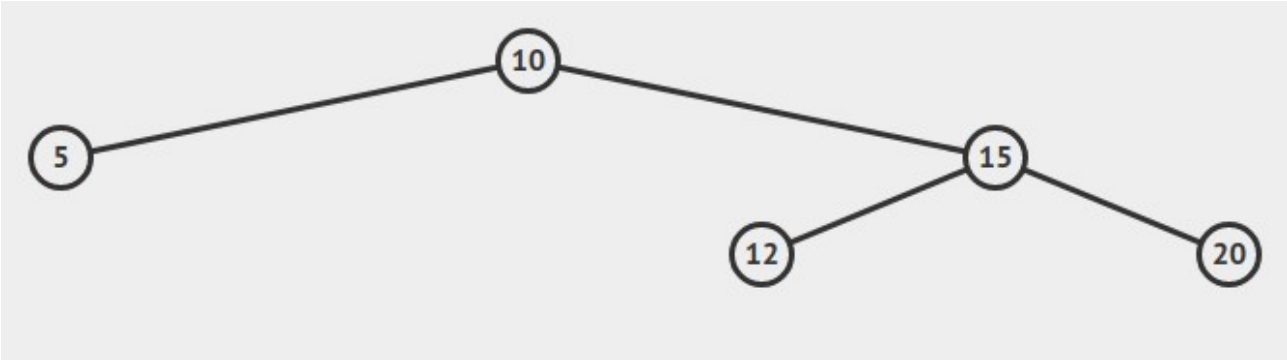
Rotação direita no 9:



Rotação esquerda no 5:

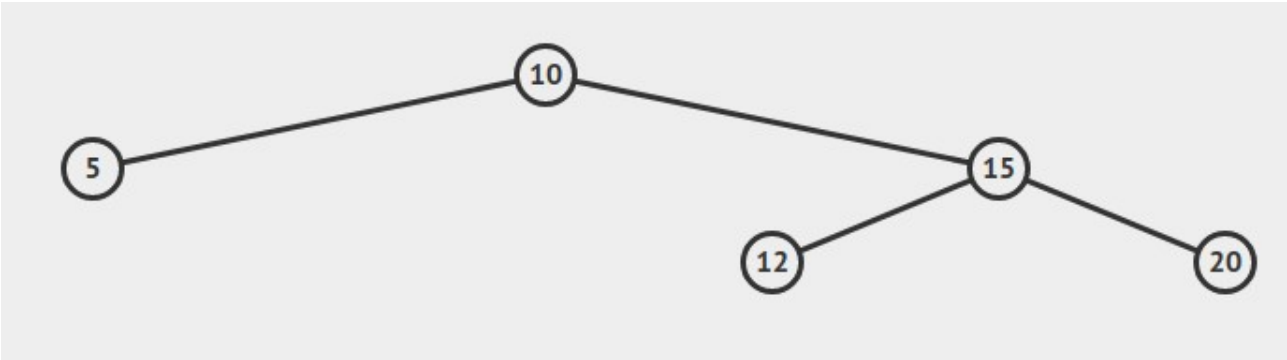


c)ABP com os elementos 10, 5, 15, 12, 20:

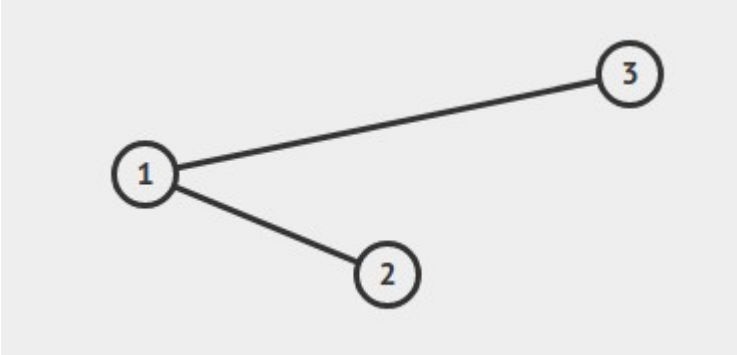


Rotação esquerda no 10:

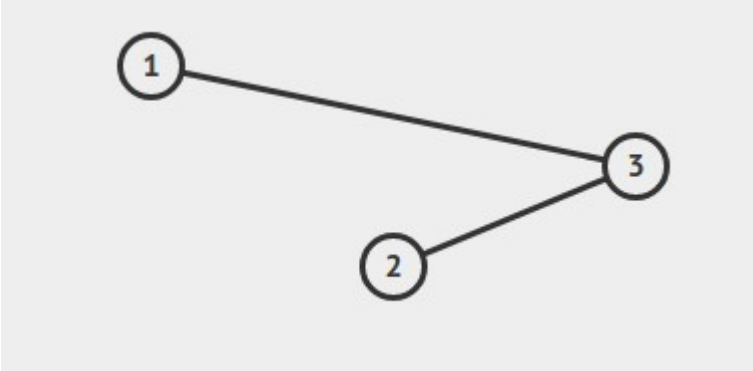
Rotação direita no 15:



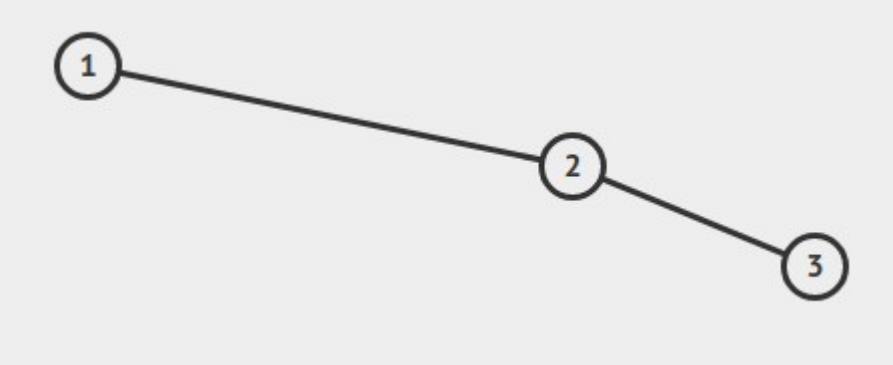
4)ABP inserida com a sequência 3, 1, 2:



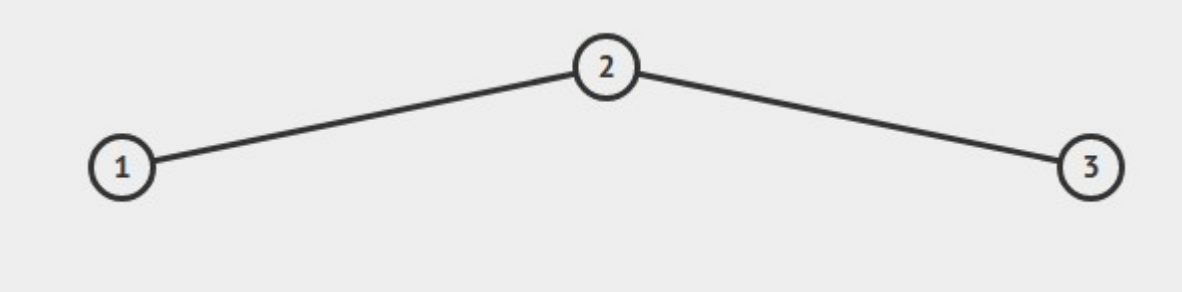
Rotação no 3 para a direita:



Rotação no 3 para a direita:



Rotação no 1 para a esquerda:



5) Existem 4 tipos de rotações, sendo eles
Rotação simples a esquerda:
Supondo que existem 3 nós inseridos na sequencia 1, 2 e 3, uma rotação simples a esquerda em 1 balancearia a arvore, deixando ela como 2, 1 e 3.
Rotação simples a direita:
Seguindo o mesmo raciocínio da primeira, porém mudando o lado de rotação, rotacionando a árvore para a direita. Tomando como exemplo uma arvore com 3 nós inseridos na sequência 3, 2 e 1, uma rotação simples a direita balancearia esta, deixando ela como 2,1 e 3.
Rotação dupla a esquerda:
Quando existe, por exemplo, uma árvore com nós filhos em diferentes direções, diferente das primeiras situações demonstradas aqui, no caso da esquerda, a rotação dupla a esquerda será o mesmo que fazer uma rotação para a esquerda e depois uma para a direita, corrigindo a inserção
Rotação dupla a direita:

O mesmo da última questão, porém será necessário primeiro rotacionar a direita, e depois para a esquerda.