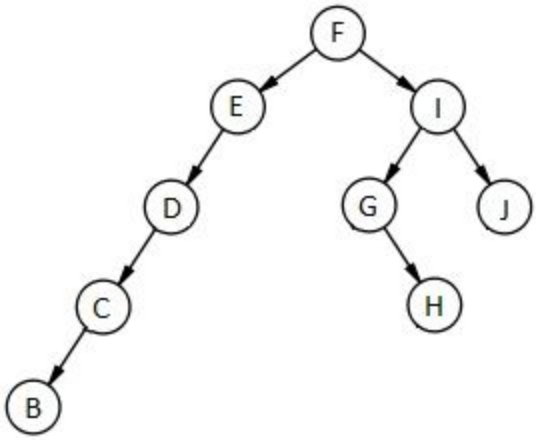
Gustavo Henrique Storti Salibi / RA: 174135 / Turma: F

Giordano Mattiello / RA: 173056/ Turma: F

Exercício em Dupla 02

Q1.



Q2.

a)

```
// será usada para ordenar ou com relação ao peso ou com relação à altura
typedef struct arvore{
        float altura, peso;
        struct arvore *esq, *dir;
} Arvore;
```

b)

```
// a função deverá ser chamada duas vezes, com o valor "caso" sendo 1 para altura e 2 para peso
  void inserir(Arvore **raiz, float altura, float peso, int caso){
        if(*raiz == NULL){
                *raiz = malloc(sizeof(Arvore));
                (*raiz)->esq = NULL;
                (*raiz)->dir = NULL;
                (*raiz)->altura = altura;
                (*raiz)->peso = peso;
        }else{
                if (caso == 1) { // ordena por altura
                        if(valor < (*raiz)->altura)
                                inserir(&(*raiz)->esq, altura, peso, 1);
                        else
                                inserir(&(*raiz)->dir, altura, peso, 1);
                } else if (caso == 2)  { // ordena por peso
                        if(valor < (*raiz)->peso)
                                inserir(&(*raiz)->esq, altura, peso, 2);
                        else
                                inserir(&(*raiz)->dir, altura, peso, 2);
                }
    }
```

c)

```
/* Função melhor utilizada para quando “r” está próximo de “n”, pois não é necessário criar uma lista
encadeada, o que aumentaria a quantidade de operações. E a complexidade, neste caso, será sempre
O(n) */
void imprimirA (Arvore *raiz, float inicio, float fim) {
        if (raiz != NULL) {
                imprimirA(raiz->dir, inicio, fim);

                if (raiz->altura > inicio && raiz->altura < fim) {
                        printf (“%f  “, raiz->peso);
                }

                imprimirA(raiz->esq, inicio, fim);
        }
}
```

```
/* Função melhor utilizada quando “r” é muito menor do que “n”, pois o intervalo de “a” até “b”
provavelmente estará antes do final da árvore e não será necessário percorrê-la até o fim e a sua ordem de
complexidade será melhor, para os casos gerais, do que a função anterior */
void imprimirB(Arvore *raiz, float inicio, float fim, int estado, NO **lista) {
    if(raiz != NULL) {
        if (raiz->valor == inicio) {
                estado = 1;
        }

        imprimirB(raiz->esq, inicio, fim, estado, lista);

        if (estado == 1) {
                // insere ordenadamente, com relação ao peso, os  elementos em uma lista encadeada
                insereOrdenado(lista, raiz);
        }

        imprimirB(raiz->dir, inicio, fim, estado, lista);

        if (raiz->valor == fim) {
                estado = 0;
        }

        // imprime os elementos da lista encadeada, do maior para o menor (com relação ao peso)
        imprimirLista(&(*lista));
}
```