Questão 1

Escreva uma função que percorre uma árvore binária de busca e retorna uma lista ligada ordenada em ordem crescente contendo todos os elementos da árvore.

```
typedef struct no_arv {
  int v;
  struct no_arv *esq, *dir;
} No_arv;
```

```
typedef struct no_lista {
  int v;
  struct no_lista *prox;
} No_lista;
```

```
No_lista* monta_lista(No_arv *arv);
```

Para que você obtenha nota integral nesta questão, sua solução, além de correta

- não deve realizar mais que um único percurso na árvore,
- não deve utilizar variáveis globais ou vetores auxiliares,
- mas pode utilizar funções auxiliares.

Versão ineficiente

```
/* Percorre a lista referente à sub-árvore esquerda
   para fazer a concatenação entre as listas. */
No_lista* monta_lista(No_arv* arv) {
    if (arv == NULL)
      return NULL;
    No_lista* lesq = monta_lista(arv->esq);
    No_lista* l =  malloc (sizeof(No_lista));
    l->v = arv->v;
    l->prox = monta_lista(arv->dir);
    if (lesq == NULL)
      return l;
    No_lista* aux = lesq;
    while (aux->prox != NULL)
      aux = aux->prox;
    aux->prox = l;
    return lesq;
}
```

Versão eficiente 1

```
/* Cria listas, indicando um apontador para o início e outro
   para o final da lista. Desta forma, é possível fazer a concatenação
   de forma eficiente.
*/

void monta_aux(No_arv *arv, No_lista** prim, No_lista** ult) {

   No_lista *l =  malloc (sizeof(No_lista));
   l->v = arv->v;

   if (arv->esq) {
      No_lista *u_esq;
      monta_aux(arv->esq, prim, &u_esq);
      u_esq->prox = l;
   }
   else
     *prim = l;
```

```
   if (arv->dir)
     monta_aux(arv->dir, &l->prox, ult);
   else
     *ult = l;
}

No_lista* monta_lista(No_arv* arv) {
    if (arv == NULL)
       return NULL;
    No_lista *prim, *ult;
    monta_aux(arv, &prim, &ult);
    ult->prox = NULL;
    return prim;
}
```

Versão eficiente 2

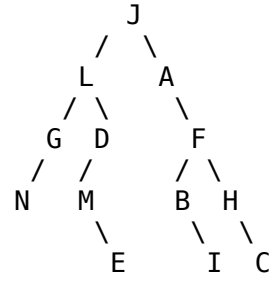
```
/* Esta versão faz um percurso inordem invertido na árvore
   e vai montando a lista do último nó até o primeiro.
 */
No_lista* aux_monta_lista(No_arv* arv, No_lista* prox) {
  No_lista *l =  malloc (sizeof(No_lista));
  l->v = arv->v;
  if (arv->dir != NULL)
    l->prox = aux_monta_lista(arv->dir, prox);
  else
    l->prox = prox;
  if (arv->esq != NULL)
    return aux_monta_lista(arv->esq, l);
  else
    return l;
}

No_lista* monta_lista(No_arv* arv) {
    if (arv == NULL)
       return NULL;
    return aux_monta_lista(arv, NULL);
}
```

Questão 2

Desenhe a árvore binária que apresenta os percursos descritos abaixo.

```
Pré-ordem: J L G N D M E A F B I H C
In-ordem:  N G L M E D J A B I F H C
```



Questão 3

Considere a seguinte declaração para os nós de uma árvore 2-3, com as restrições apresentadas em aula. Escreva uma função que imprime todos os elementos desta árvore em ordem decrescente.

```
 typedef struct no23{
  int nch;                       /* Número de chaves (1 ou 2) no nó         */
  struct no23 *esq, *cen, *dir;  /* Apontadores esquerdo, central e direito */
  int chesq, chdir;              /* Chaves esquerda e direita               */
} No23;

void imprime(No23* n) {
  if (n != NULL) {
    if (n->nch == 2) {
       imprime(n->dir);
       printf ("%d ", n->chdir);
    }
    imprime(n->cen);
    printf ("%d ", n->chesq);
    imprime(n->esq);
  }
}
```

Questão 4

O objetivo do programa abaixo é (i) iniciar uma lista ligada com nó cabeça, indicando que esta está vazia, (ii) inserir um elemento no início desta lista (após o nó cabeça) e (iii) imprimir o valor deste elemento. O código em C apresentado tem um comportamento bem definido ou sua execução poderia ser interrompida devido a um erro de acesso à memória? Caso você considere que este programa tem um ou mais erros, indique as linhas que contêm problemas, explique-os e mostre como eles poderiam ser corrigidos de maneira que o código possa funcionar adequadamente, mantendo seus objetivos e divisão em funções. Você não deve alterar as partes do código que estejam funcionando adequadamente.

```
 1: #include <stdlib.h>
 2: #include <stdio.h>
 3:
 4: typedef struct no *ap_no;
 5: struct no {
 6:   int v;
 7:   ap_no prox;
 8: };
 9:
10: void inicia(ap_no p) {
11:   p = (ap_no) malloc (sizeof (struct no));
12:   p->v = -1;
13:   p->prox = NULL;
14: }
15:
16: void insere_inicio(ap_no p, int v) {
17:   ap_no n = (ap_no) malloc (sizeof (struct no));
18:   n->v = v;
19:   n->prox = p->prox;
20:   p->prox = n;
21: }
22:
23: int main() {
24:   ap_no p;
25:   inicia(p);
```

```
26:   insere_inicio(p, 10);
27:   printf ("Valor do primeiro elemento: %d\n", p->prox->v);
28:   return 0;
29: }
```

Há um erro na função inicia, pois ela não consegue alterar o valor de `p` no corpo do `main`. A função insere_inicio não tem este problema pois o nó cabeça não será alterado. O código correto seria:

```
 1: #include <stdlib.h>
 2: #include <stdio.h>
 3:
 4: typedef struct no *ap_no;
 5: struct no {
 6:   int v;
 7:   ap_no prox;
 8: };
 9:
10: void inicia(ap_no *p) {
11:   *p = (ap_no) malloc (sizeof (struct no));
12:   (*p)->v = -1;
13:   (*p)->prox = NULL;
14: }
15:
16: void insere_inicio(ap_no p, int v) {
17:   ap_no n = (ap_no) malloc (sizeof (struct no));
18:   n->v = v;
19:   n->prox = p->prox;
20:   p->prox = n;
21: }
22:
23: int main() {
24:   ap_no p;
25:   inicia(&p);
26:   insere_inicio(p, 10);
27:   printf ("Valor do primeiro elemento: %d\n", p->prox->v);
28:   return 0;
29: }
```