Universidade Federal de Santa Catarina - Campus Blumenau
Departamento Engenharia de Controle, Automação e Computação **Avaliação 1**
BLU3202 — **Algoritmos e Estruturas de Dados** 2022/2

Aluno(a): .....

1. Numere os itens a seguir em ordem crescente de complexidade para o pior caso de execução, em que 1 indica o mais eficiente e 5 indica o menos eficiente.

   ( ) $O(n^2)$ ( ) $O(n)$ ( ) $O(nlog_2n)$
   ( ) $O(log_2n)$ ( ) $O(1)$

2. Sejam três algoritmos A_1, A_2 e A_3, cujas complexidades podem ser dadas, respectivamente, pelas expressões $C_{A_1}(n) = 2n + 3n^2 + 5$, $C_{A_2}(n) = 25n + 8$ e $C_{A_3}(n) = 2log_2n + 3$. Nas alternativas abaixo, assinale V nas que forem verdadeiras e F nas que forem falsas.

   ( ) Existe pelo menos um tamanho de entrada n maior que zero para o qual, no pior caso, A_1 é mais rápido que A_2

   ( ) A_2 será, no pior caso, mais rápido que A_1 independente do tamanho da entrada

   ( ) Os três algoritmos, se executados sob as mesmas condições, levarão mais que o dobro do tempo para processar uma entrada de tamanho $2n$ em comparação com uma entrada de tamanho n (para qualquer valor de n)

   ( ) Existe um valor x tal que, para qualquer entrada de tamanho $n > x$, A_3 será mais rápido que A_1

   ( ) Existe um valor x tal que, para qualquer entrada de tamanho $n > x$, A_2 será mais rápido que A_3

   ( ) Nenhum dos três algoritmos tem complexidade inferior à da inserção no início de uma lista encadeada

   ( ) Nenhum dos três algoritmos tem complexidade inferior à da inserção no fim de uma lista encadeada

3. O que será impresso pelo programa abaixo?

```
1  int main() {
2     int x=2, y=3, z, *t, *v, *w;
3     t = &x;
4     w = &y;
5     z = *t;
6     v = t;
7     *v = x + *w;
8     *w = z;
9     printf("%d,%d",x,y);
10 }
```

   Resposta: _____

4. O que será impresso pelo programa abaixo?

```
1
2  int main(){
3    int v[5] = {3,5,4,7,1};
4    for(int i=1;i<5;i++){
5      printf("%d-",*(v+i)+*(v+i-1)+(*v+i));
6    }
```

   Resposta: _____

5. No programa abaixo, o usuário informa o tamanho de um vetor `v` a ser criado. Esse tamanho é armazenado na variável `q` . O vetor deve armazenar os "q" múltplos de 5 $(5, 5 \times 2, \cdots, 5 \times q)$. Em seguida o programa deve imprimir os elementos do vetor. Preencha as lacunas abaixo para que o programa tenha esse comportamento. Utilize aritmética de ponteiros para manipular os elementos do vetor.

```
1  int main(){
2     int *v, q, i;
3     printf("Tamanho do vetor: ");
4     scanf("%d",&q);
5     v = ____________________;
6     for(int i=0;i<q;i++){
7        _______ = (q+1)*5;
8     }
9     for(i=0;i<q;i++){
10       printf("%d\n", _____);
11    }
12 }
```

   Linha 5: _____
   Linha 7: _____
   Linha 10: _____

6. Considere a função abaixo com os parâmetros $m = 1$ e $n = 2$:

```
1  int f(int m, int n){
2     if (m == 0) return n+1;
3     else if (n==0) return f(m-1,1);
4     else return f(m-1,f(m,n-1));
5  }
```

   Responda:

   - Qual será o valor retornado pela função? _____
   - Quantas chamadas recursivas são feitas pelo programa abaixo para excluindo-se a primeira chamada à função? _____

7. Qual será o valor da variável x após a execução do programa abaixo? ?

```
1  int f(int *v, int s){
2     if(s==1)
3        return (*v)*(*v);
4     else
5        return (*v)*(*v)+f(v+1,s-1);
6  }
7
8  int main(){
9     int v[4] = {1,2,3,4};
10    int x = f(v,4);
11    printf("%d\n",x);
12 }
```

Resposta: _____

8. Enumere as descrições abaixo conforme os tipos de listas a seguir. Observações: (1) Um mesmo tipo de lista pode se aplicar a mais de uma descrição; (2) Uma mesma descrição pode se referir a mais de um tipo de lista. Nesse caso, basta informar um tipo correto.

   1 Listas baseadas em vetores dinâmicos

   2 Listas encadeadas

   3 Listas duplamente encadeadas

   4 Listas circulares

   ( ) Estrutura de dados que pode ser percorrida em ambos os sentidos pois cada um de seus elementos aponta tanto para seu predecessor quanto para o sucessor.
   ( ) Pode-se acessar o primeiro elemento da lista diretamente a partir do último elemento, sem percorrer toda a lista.
   ( ) Requer que os elementos sejam movidos para posições anteriores ou posteriores no caso de exclusões e inclusões de elementos da lista;
   ( ) A partir de um determinado elemento da lista, só é possível acessar os seus sucessores pois os elementos não possuem apontamento para seus predecessores.
   ( ) Os itens da lista são armazenados de forma contígua na memória.
   ( ) Os itens da lista são armazenados de forma não contígua na memória.

Para as questões 9 a 10, considere a seguinte estrutura:

```
typedef struct lstItem{
   int dado;
   struct lstItem *next;
} listaItem;
```

9. Complete as lacunas da função abaixo para que ela faça a inserção de um item no *fim* da lista. Considerar que o parâmetro `l` é um ponteiro para o primeiro elemento da lista. A função deve retornar um ponteiro para o primeiro elemento da lista.

```
1  listaItem *insere_fim(listaItem *l,int dado){
2     listaItem *novo = malloc(_______________);
3     novo->dado = dado;
4     novo->next = NULL;
5     if(l==NULL){
6        return _________;
7     }
8     else{
9        listaItem *ultimo = l;
10       while(__________________){
11        ultimo = ________________;
12       }
13       ________________ = novo;
14    }
15    return l;
16 }
```

   Linha 2: _____
   Linha 6: _____
   Linha 10: _____
   Linha 11: _____
   Linha 13: _____

10. Complete as lacunas da função abaixo para que ela faça a inserção de um item no *início* da lista. Considerar que o parâmetro `l` é um ponteiro para o primeiro elemento da lista. A função deve retornar um ponteiro para o primeiro elemento da lista.

```
1  listaItem *insere_inicio(listaItem *l,int dado){
2     listaItem *novo = ____________________;
3     novo->dado = dado;
4     novo->next = _______;
5     l = _____;
6     return l;
7  }
```

   Linha 2: _____
   Linha 4: _____
   Linha 5: _____