A. Análise de Complexidade: Big-O

Determine a ordem de complexidade dos algoritmos traduzidos em cada um dos seguintes blocos de código em linguagem C.

a)

```
for (k = 0; k < n; k++) {
    j = 0;
    while (j <= k)
        j = j + 1;
}
```

b)

```
    for (k = 0; k < n; k++) {
        j = k;
        while (j < n)
            j = j + 1;
    }
```

c)

```
    for (k = 0; k < n; k++) {
        for (j = 0; j < n; j = j + 2)
            printf("%d\n", j);
    }
```

d)

```
    for (k = 0; k < n; k++) {
        for (j = 0; j < n; j = j + n/2)
            printf("%d\n", j);
    }
```

e)

```
fim =  n % 10;
for (k = 0; k < fim; k++) {
    for (j = k; j < n; j++)
        printf("%d\n", j);
}
```

f)

```
cont = 0;
for (k = 0; k < n; k++) {
    p = k % 10;
    for (j = 0; j <= p; j++)
        cont++;
}
```

g)

```
soma = 0;
for (k = 0; k < n; k++)
    for (j = 0; j == k; j++)
        soma = soma + X[k][j];
```

B. Algoritmos de ordenação e de pesquisa em arrays 1D

1. Determinar as ordens de complexidade dos algoritmos de ordenação estudados nas aulas teóricas (ver apontamentos sobre "Algoritmos de ordenação em arrays 1D").

2. Determinar as complexidades amortizadas (sobre as estruturas de dados) dos algoritmos de ordenação estudados nas aulas teóricas (ver apontamentos sobre "Algoritmos de ordenação em arrays 1D").

3. Determine a ordem de complexidade dos algoritmos de pesquisa estudados nas aulas teóricas (ver apontamentos sobre "Algoritmos de pesquisa em arrays 1D").

4. Construa uma versão mais otimizada do algoritmo de ordenação por borbulhagem (BubbleSort), a partir da versão estudada nas aulas teóricas.

5. Implementar um programa em C que realize as seguintes ações (pela ordem indicada):
- construir um array 1D **A** com **N** números inteiros gerados aleatoriamente entre 10 e 50, em que N é um número inteiro entre 1 e 30 gerado aleatoriamente
- mostrar (no monitor) o array **A**
- gerar aleatoriamente um número **num** entre 10 e 50
- pesquisar o número **num** no array **A**
- mostrar no monitor o índice do número num no array **A**, caso exista
- ordenar o array **A** por ordem crescente, usando um dos algoritmos estudados
- mostrar (no monitor) o array **A** ordenado
- pesquisar o número **num** no array **A** (agora ordenado)
- mostrar no monitor o índice do número **num** no array **A**, caso exista
- determinar e mostrar o maior elemento do array **A** usando **apenas** instruções de atribuição
- determinar e mostrar o menor valor do array **A** usando **apenas** instruções de atribuição
- determinar e mostrar a quantidade de elementos do array **A** iguais ao menor elemento (otimizar o processo)
- determinar e mostrar o segundo maior elemento do array **A** (otimizar processo)

Nota: usar as bibliotecas fornecidas na página web da disciplina (**Algoritmos**)

6. Implementar um programa em C que realize as seguintes ações (pela ordem indicada):
- construa um array 1D **A** com 30 números inteiros gerados aleatoriamente entre 10 e 30
- mostre o array **A** no monitor
- atribua à variável **num** um número inteiro gerado aleatoriamente entre 10 e 30
- verifique se **num** existe no array **A** e em que **posição** se encontra, usando o algoritmo de Pesquisa Sequencial
- mostre o resultado obtido (não existe ou existe na posição k)
- determine o número de elementos do array **A** que são maiores do que **num**

Nota: usar as bibliotecas fornecidas na página web da disciplina (**Algoritmos**)

7. Implementar um programa em C que realize as seguintes ações (pela ordem indicada):
- construa um array 1D **A** com **50** números inteiros gerados aleatoriamente entre 10 e 20
- mostre o array **A** no monitor
- atribua à variável **num** um número inteiro gerado aleatoriamente entre 10 e 20
- determine a **posição** de um elemento de A que seja igual ao valor de **num** (se existe), usando o algoritmo de Pesquisa Binária (versão recursiva)
- mostre o resultado obtido (não existe ou existe na posição k)
- se o valor de **num** existe em A, então
  - determine o maior elemento de **A** menor que **num**
  - determine o menor elemento de A maior que **num**
  - determine a quantidade de elementos de **A** que são iguais ao valor de **num**
  - mostre os resultados obtidos

Nota: usar as bibliotecas fornecidas na página web da disciplina (**Algoritmos**)

8. Implementar o seguinte algoritmo de ordenação que só funciona se não existirem elementos repetidos no array (ordenação por contagem): percorrer o array e contar para cada elemento do array, quantos são os elementos menores que ele; se um elemento tiver **k** elementos menores então a sua posição no array ordenado será **k+1**.

9. Implementar um programa em C que realize as seguintes ações (pela ordem indicada):
- construa um array 1D **A** com 40 números inteiros gerados aleatoriamente entre 10 e 50
- mostre (no monitor) o array **A**
- ordene o array **A** por ordem crescente, usando um dos algoritmos estudados
- mostre (no monitor) o array **A** ordenado
- remova todos os elementos do array **A** iguais ao valor do **maior** elemento
- mostre no monitor o array **A** final

Nota: usar as bibliotecas fornecidas na página web da disciplina (**Algoritmos**)

10. Escrever um programa em C que recebendo um array de números inteiros positivos, os ordene por ordem decrescente, segundo o número de ocorrências de cada um no array.

Exemplo:

Entrada: 1 5 5 5 3 3 2 5 1 3 1 1 5

Saída: 5 5 5 5 5 1 1 1 1 3 3 3 2