LISTA 2 - FRANCISCO BRAZ

1-)

a-) Verifica se a matriz é igual a sua transposta. Então se trocarmos as linhas pelas colunas, teremos o mesmo resultado. O algoritmo faz isso comparando os elementos à direita da diagonal principal com os elementos à esquerda da diagonal principal.

b-) Seria a parte do código que está mais aninhada nos loops.

se A[i, j] <> A[j,i] então retorne false

c-)

De acordo com as simplificações vistas na aula 05, podemos focar nas operações principais do algoritmo, que seriam aquelas que consomem mais tempo para executar. Nesse exemplo a operação principal seria a explicitada no item b dessa questão. Portanto, nosso cálculo de complexidade seria:

```
para i = 1 até n-1 faça
  para j = i+1 até n faça
     se A[i, j ] <> A[j, i] então retorne False
```

T(max) = $T_{para} + T_{para} + T_{cond}$ = (n-1)*1 + (n-1)*n*1 + (n-1)* n *(1 + 1 + 1 +1)

$$= n-1 + n^2 - n + 4n^2 - 4n$$

$$= 5n^2 - 4n - 1$$

Vamos considerar somente o termo inicial da fórmula, já que os termos de ordem mais baixa (quando comparados ao de ordem mais alta) são considerados relativamente insignificantes para grandes valores da nossa variável n. Como as constantes também são menos significativas para grandes entradas, iremos ignorar o coeficiente do nosso termo de ordem mais alta. Assim, ficaríamos somente com o termo n^2. Nossa complexidade seria, portanto O(n^2).

De um modo mais formal:

$$\sum_{i=1}^{n-1} * \sum_{j=i+1}^{n} * c = \sum_{i=1}^{n-1} * c * (n-(i+1)+1)$$

$$= \sum_{i=1}^{n-1} * c * (n-i)$$

$$= c * \frac{n*(n-1)}{2}$$

$$= O(n^2)$$

2-)
a-)
Se considerarmos que o vetor já está ordenado, precisaríamos, somente, pegar o primeiro elemento e fazer uma subtração do segundo elemento. Então teríamos uma complexidade O(1).

Pseudocódigo:

```
algoritmo CalculaIntervalo(A, n)
{-Entrada: vetor A de n elementos em ordenado em ordem crescente
 Saída: valor da diferença entre maior e menor elemento do vetor-}

início
        retorna A[n] - A[1]
fim
```

Caso fosse necessário ordenar o vetor, teríamos:
Pseudocódigo:

```
algoritmo CalculaIntervalo(A, n)
{-Entrada: vetor A de n elementos em ordem arbitrária
 Saída: valor da diferença entre maior e menor elemento do vetor-}

início
        função Troca(A, i, menor):vazio
            início
                aux:=A[i]
                A[i] = A[menor]
                A[menor]:= aux
```

```
        fim

    função OrdenaCrescente(A, n): vazio
            início
                    para i=1 até n-1 faça
                        início
                            menor := i
                            para j = i+1 até n faça
                                    se A[j] < a[menor] então menor := j
                            Troca(A, i, menor)
                         fim
            fim

    função intervalo(A, n): inteiro
        início
            OrdenaVetor(A,n)
            retorna A[n] - A[1]
        fim
fim
```

Complexidade:

$T(max) = T_{para} + T_{para} + T_{cond} = (n-1)*1 + (n-1)*n*1 + (n-1)*n*(1 + 1 + 1 + 1 + 1)$

$= n-1 + n^2 - n + 5n^2 - 5n$

$= 6n^2 - 4n - 1$

Utilizando a mesma lógica explicitada no item c da questão 1, teríamos que nossa complexidade seria: $O(n^2)$.

b-)

Pseudocódigo:

algoritmo CalculaIntervalo(A, n)
{-Entrada: vetor A de n elementos em ordem arbitrária
Saída: valor da diferença entre maior e menor elemento do vetor-}

```
início
        função menor(A,n): inteiro
                início
                        menor:=1
                        para i = 1 até i = n-1 faça
                           início
                                prox := i+1
                                se A[prox] < A[menor] então menor := prox
                           fim
                        retorna A[menor]
                fim


        função maior(A,n): inteiro
                início
                        maior:=1
                        para i = 1 até i = n-1 faça
                           início
                                prox := i+1
                                se A[prox] > A[maior] então maior := prox
                           fim
                        retorna A[maior]
                fim
        função intervalo(A, n): inteiro
                retorna menor(A,n) - maior(A,n)

fim
```

Complexidade:

$T_{paraMenor} + T_{condMenor} + T_{paraMaior} + T_{condMaior}$

$= n*1 + n*(1 + 1 + 1 + 1 + 1 +1) + n*1 + n*(1 + 1 + 1 + 1 + 1 +1)$

$= n + 6n + n + 6n$

$= 14n$

Como as constantes também são menos significativas para grandes entradas, iremos ignorar o coeficiente do nosso termo de ordem mais alta. Assim, ficaríamos somente com o termo n. Nossa complexidade seria, portanto, O(n).

3-)

a-)

Pseudocódigo:

```
algoritmo Expressao(S, n)
{-Entrada: de uma string S com n caracteres
 Saída: retorna “Correto” ou “Incorreto” de acordo com a expressão-}

início
        funcao verificarExpressao(S, n): booleano
                início
                        para i = 1 até i = n faça
                           início
                                se S[i] = ‘(‘ então qtdPA := qtdPA  + 1
                                senão se S[i] = ‘)‘ então qtdPF := qtdPF  + 1
                                senão se S[i] = ‘[‘ então qtdCA := qtdCA  + 1
                                senão se S[i] = ‘]‘ então qtdCF := qtdCF  + 1

                                se qtdCF > qtdPF && qtdCF > qtdCA
                                então retorna 0

                                se qtdPF > qtdCF && qtdPF > qtdPA
                                então retorna 0
                           fim

                        se qtdCF <> qtdCA && qtdPF <> qtdPA
                        então retorna 0

                        retorna 1
                fim

                funcao resultadoVerificacao(S, n): vazio
                   início
                        verificacao:= verificaExpressao(S, n)
```

se vericacao = 1 então imprima(“Correto”)
senão imprima(“Incorreto”)
fim

fim

b-)
Complexidade de tempo:
$T(max) = T_{para} + T_{cond} + T_{cond} + T_{cond} + T_{cond} + T_{cond} + T_{cond}$
$= n*1 + n*28$
$= 29n$
Como as constantes também são menos significativas para grandes entradas, iremos ignorar o coeficiente do nosso termo de ordem mais alta. Assim, ficaríamos somente com o termo n. Nossa complexidade seria, portanto, O(n).

Complexidade de espaço:
1 inteiro para armazenar a variável i
1 inteiro para armazenar a variável qtdCA
1 inteiro para armazenar a variável qtdCF
1 inteiro para armazenar a variável qtdPA
1 inteiro para armazenar a variável qtdPF
1 inteiro para armazenar a variável verificacao
f(n) = 1 + 1 + 1 + 1 + 1 + 1
= 6
Nossa complexidade de espaço seria O(1).

c) Implementação no arquivo checarExpressao.c

4-)
a-)
Pseudocódigo:
algoritmo vetorParImpar(S, n)
{-Entrada: vetor A de n elementos em ordem arbitrária
Saída: vetor com elementos par em ordem crescente e elementos ímpar em ordem descrescente-}

```
início
        função separarVetor(A, n, vetorPar, vetorImpar)
                inicio
                        j := 0
                        k := 0
                        para i = 0 até i = n faça
                           início
                                se A[i] % 2 = 0
                                então
                                        vetorPar[j] = vetor[i]
                                        j := j + 1
                                senão
                                        vetorImpar[k] = vetor[i]
                                        k := k + 1
                           fim
                fim


        função OrdenaVetor(A, n): vazio
          início
              para i=1 até n-1 faça
                início
                   menor := i
                   maior := i
                   para j = i+1 até n faça
                       se vetorPar[j] < vetorPar[menor] então menor := j
                       se vetorImpar > vetorImpar[maior] então maior := j
                   TrocaMenor(A, i, menor)
                   TrocaMaior(A, i, maior)
                fim
              fim

        Função novoVetor(A, n, vetorPar, vetorImpar)
                início
                        i := 0
                        j := 0
```

```
                para K = 0 até K = n faça
                    início
                        se A[i] % 2 = 0
                        então
                                A[i] = vetorPar[i]
                                i := i + 1
                        senão
                                A[k] = vetorImpar[j]
                                j := j + 1
                    fim
            fim

        função principal(A, n)
            início
                vetorPar[n/2]
                vetorImpar[n/2]
                separarVetor(A, n, vetorPar, vetorImpar)
                ordenarVetor(A, n/2, vetorPar, vetorImpar)
                novoVetor(A, n, vetorPar, vetorImpar)
            fim
fim
```

b-)

Complexidade de tempo:

$T(max) = T_{separaVetor} + T_{ordenaVetor} + T_{novoVetor}$

$T_{separaVetor} = T_{para} + T_{cond} = n*1 + n*14 = 15n$

$T_{ordenaVetor} = T_{para} + T_{para} + T_{cond} =(n-1)*3 + (n-1)*n*1 + (n-1)*n*10 = 11n^2 - 8n - 3$

$T_{novoVetor} = T_{para} + T_{cond} = n*1 + n*14 = 15n$

$T(max) = 11n^2 + 22n - 3$

Vamos considerar somente o termo inicial da fórmula, já que os termos de ordem mais baixa (quando comparados ao de ordem mais alta) são considerados relativamente insignificantes para grandes valores da nossa variável n. Como as constantes também são menos significativas para grandes entradas, iremos ignorar o coeficiente do nosso termo de ordem mais alta. Assim, ficaríamos somente com o termo n^2. Nossa complexidade seria, portanto O(n^2).

Complexidade de espaço:

1 inteiro para armazenar a variável i na função separarVetor
1 inteiro para armazenar a variável i na função ordenaVetor
1 inteiro para armazenar a variável i na função novoVetor
1 inteiro para armazenar a variável j na função separarVetor
1 inteiro para armazenar a variável k na função separarVetor
1 inteiro para armazenar a variável j na função novoVetor
n inteiros para armazenar o vetorPar
n inteiros para armazenar o vetorImpar

f(n) = 1 + 1 + 1 + 1 + 1 + 1 + n + n
= 2n + 6
Nossa complexidade de espaço seria O(n).

c-) Implementação no arquivo vetorParImpar.c