Questão 1

Suponha que temos dois algoritmos T_1 e T_2 para resolver um determinado problema, com os seguintes tempos de execução:

- $T_1(n) = 625n$;
- $T_2(n) = n^2$.

Responda:

1. Qual é a complexidade assintótica de cada algoritmo?
2. Para quais valores de n o algoritmo T_1 é mais eficiente que o T_2, e para quais valores$ T_2 se torna mais eficiente que T_1?

i) $O(T_1(n)) = O(n)$; $O(T_2(n)) = O(n^2)$

ii) $1 \leq n < 625$: $T_2(n)$ é mais eficiente

$625 \leq n$: $T_1(n)$ é mais eficiente.

Questão 2

O custo computacional $T_i(n)$ de diversos algoritmos é mostrado abaixo.
Qual a complexidade de cada algoritmo? Ponha-os em ordem crescente de complexidade.

- $T_1(n) = nlog(n) + log(n)$;
- $T_2(n) = 2n + n^3 + 25$;
- $T_3(n,k) = n + k$ onde $k \leq n$
- $T_4(n) = n + log(n)$;
- $T_5(n) = 100nlog(n) + n^3 + 100n$;
- $T_6(n) = 0.01nlog(n) + n(log(n))^2$;
- $T_7(n) = 2n + n^{0.5} + 0.5n^{1.25}$;
- $T_8(n) = 0.01n + 100n^2$;
- $T_9(n) = 100n + 0.01n^2$;
- $T_{10}(n) = T(n) = 2T(n-1) + 2$ para $n > 1$ e $T(1) = O(1)$; **Dica:** `método da árvore de recursão`

$T_1(n) = (n+1)\log(n) \Rightarrow O(n\log n)$.

$T_2(n) = 2n + n^3 + 25 \Rightarrow O(n^3)$

$T_3(n,K) = n + K \ (K \leq n) \Rightarrow O(n)$

$T_4(n) = n + \log n \Rightarrow O(n)$

$T_5(n) = 100n\log n + n^3 + 100n \Rightarrow O(n^3)$.

$T_6(n) = 0{,}01\, n\log n + n(\log n)^2 \Rightarrow O(n(\log n)^2)$

$T_7(n) = 2n + n^{0,5} + 0{,}5\, n^{1,25} \Rightarrow O(n)$

$T_8(n) = 0{,}01n + 100n^2 \Rightarrow O(n^2)$

$T_9(n) = 100n + 0{,}01n^2 \Rightarrow O(n^2)$

$T_{10}(n) = T(n) = 2T(n-1) + 2$, $n > 1$, $T(1) = O(1)$

$\Rightarrow O(2^n)$: $O(T(2)) = 2\,O(1) = O(2)$

$O(T(3)) = O(4) \ldots$

$O(n)$: T_3; T_4; T_7	$O(n^2)$: T_8, T_9
$O(n\log n)$: T_1	$O(n^3)$: T_2, T_5
$O(n(\log n)^2)$: T_6	$O(2^n)$: T_{10}

$O(n) < O(n\log n) < O(n(\log n)^2) < O(n^2) < O(n^3) < O(2^n)$

Questão 3

Calcule a complexidade dos algoritmos abaixo:

1. Loops em sequência

```
int a = 0, b = 0;
for (i = 0; i < n; i++) { n
    a = a + i;  1
}
for (j = 0; j < m; j++) { m
    b = b + j;  1
}
```

It:

$\sum_{i=0}^{n} 1 + \sum_{j=0}^{m} 1 = n + m$

$O(n+m)$, pois são independentes.

2. Loop com condicionais

```
float what2(int *arr, int n) {
    int a = 0;
    for (int i = 0; i < n; i++) { n
        if(arr[i] > 10) { todos
            for (int j = 0; j < n; j++) { n
                a += n / 2; 2
            }
        } else {
            printf("ok :(")
        }
    }
}
```

It (pior caso):

$\sum_{i=1}^{n} \sum_{j=1}^{n} 2 = \sum_{i=1}^{n} 2n = 2n^2 \Rightarrow O(n^2)$.

3. Loops duplo

```
int a = 0;
for (int i = 0; i < n; i++) { 1:n
    for (int j = n; j > i; j--) { i:n
        a += i + j; 2
    }
}
```

It:

$\sum_{i=1}^{n} \sum_{j=i}^{n} 2 = \sum_{i=1}^{n} 2(n-i+1) = \sum_{i=1}^{n} 2n - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1$

$= 2n^2 - n(n+1)/2 + n = 4n^2/2 - (n^2+n)/2 + 2n/2 =$

$= (3n^2+n)/2 \Rightarrow O(n^2)$

4. Loops duplo com constantes

```
float what4(int *arr, int n) {
    int a = 0;
    for (int i = 0; i < 1000; i++) { 1000
        for (int j = 0; j < 5000; j++) { 5000
            a += i + j; 2
        }
    }
}
```

It:

$\sum_{i=1}^{1000} \sum_{j=1}^{5000} 2 = \sum_{i=1}^{1000} 2.5000 = 2.5000.1000$ (cte)

$\Rightarrow O(1)$

5. Loops com crescimento em progressão geométrica

```
int a = 0;
for (int i = n/2; i <= n; i++) {
    for (int j = 2; j <= n; j = j * 2) {
        a += i + j; 2
    }
}
```

Primeiro loop: $\lfloor n/2 \rfloor$ até n. Spg, são $n - n/2 + 1$ iterações, $O(n)$.

Segundo loop: $2, 4, 16, \dots, n : 1, 2, \dots, \lfloor \log n \rfloor$. Spg, são $\log n$ iterações, $O(\log n)$. Total: $O(n \log n)$.

6. Loop com crescimento logarítmico

```
int a = 0, i = n;
while (i > 0) {
    a += i; 2 ]
    i /= 2; 1 ] 3
}
```

Loop: $n \lfloor n/2 \rfloor \lfloor n/4 \rfloor, \dots, 1 : 1, 2, \dots, \lfloor \log n \rfloor$.

Logo, $O(\log n)$.

7. Loop duplo com redução logarítmica

```
int a = 0, i = n;
while (i > 0) {
    for (int j = 0; j < i; j++) {
        a += i; 2
    }
    i /= 2; 1
}
```

Loops: 1 a n; 1 a $\lfloor n/2 \rfloor$; 1 a $\lfloor n/4 \rfloor$; ...; 1

Iterações (spg): $n + n/2 + n/4 + \dots + 1$

$\Rightarrow O(n)$.

8. Soma dos elementos de um vetor

```
float soma(float *arr, int n) {
    float total = 0;
    for (int i = 0; i < n; i++) { n
        total += arr[i]; 1
    }
    return total;
}
```

T:

$\sum_{i=1}^{n} 1 = n \rightarrow O(n)$

9. Busca sequencial

```
int buscaSequencial(int *arr, int n, int x) {
    for (int i = 0; i < n; i++){
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

piores casos: tá no último ou não tá

T (pior caso):

$\sum_{i=1}^{n} 1 = n \Rightarrow O(n)$.

10. Busca binária

```
int buscaBinaria(int *arr, int x, int i, int j) {
    if (i >= j) {
        return -1;
    }

    int m = (i + j) / 2;
    if (arr[m] == x) {
        return m;
    } else if ( x < arr[m] ) {
        return buscaBinaria (arr, x, i, m-1);
    } else {
        return buscaBinaria (arr, x, m+1, j);
    }
}
```

A cada chamada, reduzimos m pela metade $m, \lfloor m/2 \rfloor, \lfloor m/4 \rfloor, \ldots, 1$

$\Rightarrow O(\log n)$.

11. Multiplicação de matrizes

```
void multiplicacaiMatriz(float **a, float **b, int n, int p, int m, float **x) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            x[i][j] = 0.0;
            for (int k = 0; k < p; k++) {
                x[i][j] += a[i][k] * b[k][j]; 2
            }
        }
    }
}
```

T:

$\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{p} 2 = 2nmp \Rightarrow$

$\Rightarrow O(nmp)$, pois os loops são independentes.