- Chave de Respostas -

1) Calcule a complexidade assintótica de **pior caso** dos algoritmos Merge Sort e Quicksort.
Obs.: Não é necessário fazer a verificação, somente resolver.

- **Merge Sort**:

```
int MergeSort(int *vet, int ini, int fim){
    int meio;
    if(ini < fim){
        meio = (ini + fim)/2;
        MergeSort(vet, ini, meio);        ⇒   T(n/2)
        MergeSort(vet, meio+1, fim);      ⇒   T(n/2)
        Merge(vet, ini, meio, fim);       ⇒   O(n)
    }
}
```

- Condição Básica e Relação de Recorrência referentes ao algoritmo:

$$\begin{cases} T(1) = 1, & \text{para} \quad n = 1 \\ T(n) = 2T(\frac{n}{2}) + O(n), & \text{para} \quad n > 1 \end{cases}$$

- Expandindo a Relação de Recorrência, temos:

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + n \\ &= 2[2T(\frac{n}{4}) + \frac{n}{2}] + n \\ &= 2^2T(\frac{n}{4}) + n + n \\ &= 2^2[2T(\frac{n}{8}) + \frac{n}{4}] + n + n \\ &= 2^3T(\frac{n}{8}) + n + n + n \\ &= 2^3[2T(\frac{n}{16}) + \frac{n}{8}] + n + n + n \\ &= 2^4T(\frac{n}{2^4}) + 4n \end{aligned}$$

- Assim, após k passos, temos:

$$T(n) = 2^kT(\frac{n}{2^k}) + n.k \qquad (1)$$

- Ao final esperamos obter:

$$\begin{aligned} \frac{n}{2^k} &= 1 \qquad (\textit{A execução finaliza em } T(1), \textit{ i.e., quando } \frac{n}{2^k} \textit{ atingir } 1) \\ 2^k &= n \\ \log_2 2^k &= \log_2 n \\ k\log_2 2 &= \log_2 n \\ k &= \log_2 n \qquad (2) \end{aligned}$$

- Substituindo (2) em (1):

$$\begin{aligned} T(n) &= 2^{\log_2 n}\,T(\frac{n}{2^{\log_2 n}}) + n\log_2 n \qquad (2^{\log_2 n} = n^{\log_2 2} = n) \\ &= n^{\log_2 2}\,T(\frac{n}{n^{\log_2 2}}) + n\log_2 n \\ &= nT(\frac{n}{n}) + n\log_2 n \qquad (T(\frac{n}{n}) = 1) \\ &= n\log_2 n + n \qquad \Rightarrow \qquad O(n\log_2 n) \end{aligned}$$

***Obs.**: $O(n\log_2 n)$ é a complexidade para todos os casos (melhor, médio e pior), pois o Merge Sort sempre divide o vetor ao "meio" (independente dos números presentes no vetor).

- **Quicksort**:

```
int Quicksort(int *vet, int ini, int fim){
    int meio;
    if(ini < fim){
        meio = particiona(vet, ini, fim);    ⇒ O(n)
        Quicksort(vet, ini, meio-1);         ⇒ T(n − 1) //Só diminui o tamanho do vetor em uma unidade.
        Quicksort(vet, meio+1, fim);         ⇒ T(0) //Pivô na última posição: não há "lado direito" do vetor.
    }
}
```

- Condição Básica e Relação de Recorrência referentes ao algoritmo:

$$\begin{cases} T(1) = 1, & \text{para} \quad n = 1 \\ T(n) = T(n-1) + O(n), & \text{para} \quad n > 1 \end{cases}$$

- Expandindo a Relação de Recorrência, temos:

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &= T(n-3) + (n-2) + (n-1) + n \\ &= T(n-4) + (n-3) + (n-2) + (n-1) + n \end{aligned}$$

- Assim, após k passos, temos:

$$T(n) \quad = \quad T(n-k) + (n-(k-1)) + (n-(k-2)) + (n-(k-3)) + \ldots + (n-(k-k)) \tag{3}$$

- Ao final esperamos obter:

$$\begin{aligned} n - k &= 1 \qquad \textit{(A execução finaliza em } T(1)) \\ k &= n - 1 \end{aligned} \tag{4}$$

- Substituindo (4) em (3):

$$\begin{aligned} T(n) &= T(n-(n-1)) + (n-(n-2)) + (n-(n-3)) + (n-(n-4)) + \ldots + (n-(n-1-n+1)) \\ &= T(1) + 2 + 3 + 4 + \ldots + n \qquad (T(1) = 1) \\ &= 1 + 2 + 3 + 4 + \ldots + n \\ &= \frac{n(n+1)}{2} \\ &= \frac{n^2+n}{2} \qquad \Rightarrow \qquad O(n^2) \end{aligned}$$