UTFPR Câmpus Pato Branco
Disciplina: **AE23CP - Algoritmos e Estruturas de Dados II - Turmas 3CPA e 3CPB**
Curso: **Engenharia de Computação - 3^o Período**
Prof^a. Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

APS02 - 2^a Lista de exercícios
Métodos de Ordenação e Desempenho

Os algoritmos abstratos de três conhecidos métodos de ordenação: Bolha (Fig. 1), Quicksort (Fig. 2) e Merge-sort (Fig. 4) são listados a seguir.

```
1  Algoritmo bolha(vetor[1..n])
2  inicio
3     para i de 1 até n faça
4       para j de 1 até n-i faça
5          se(vetor[j] > vetor[j+1])então
6           inicio
7             aux <- vetor[j];
8             vetor[j]  <- vetor[j+1];
9             vetor[j+1] <- aux;
10          fim_Se
11 fim
```

Figura 1: Algoritmo abstrato do método Bolha

Ao analisar a complexidade de tais algoritmos, chega-se aos seguintes resultados, segundo análise assintótica:

	Bolha	Quicksort	Merge-sort
Pior Caso	$O(n^2)$	$O(n^2)$	$O(nlogn)$
Caso Médio	O(n2)	O(n log n)	O(n log n)

Para esta atividade pede-se a realização de uma análise empírica, onde as tarefas seguintes deverão ser realizadas:

1. Implemente os métodos apresentados;
2. O programa deve solicitar ao usuário a dimensão do vetor a ser criado;
3. Faça uma função que inicialize o vetor criado com números randômicos;
4. Faça uma função para calcular o tempo de execução de cada método;
5. Ordene o vetor usando os métodos implementados e informe os respectivos tempo de execução e o tamanho de cada vetor.
6. Grave as informações referentes às execuções em um arquivo .txt, informando o tamanho do vetor e os respectivos tempos de execução dos métodos.

```
Algoritmo QuickSort(X[1..n], IniVet, FimVet)
var
   int i, j, pivo, aux
início
   i <- IniVet;
   j <- FimVet;
   pivo <- X[(IniVet + FimVet)/2];
      repita
          enquanto ((X[i] < pivo) e (i<FimVet)) faça
             i <- i + 1;
          fimEnquanto
          enquanto ((X[j] > pivo) e (j > Inivet)) faça
             j <- j - 1;
          fimEnquanto
          se (i <= j) então
              aux  <- X[i];
              X[i] <- X[j];
              X[j] <- aux;
              i <- i + 1;
              j <- j - 1;
          fimSe
      enquanto(i <= j);
      se (j > IniVet) então
            QuickSort(X, IniVet, j)
      fimSe
      se (i < FimVet) então
            QuickSort(X, i, FimVet)
      fim
```

Figura 2: Algoritmo abstrato Quicksort

7. Faça os experimentos com vetores de tamanho: 100, 1.000, 5.000, 10.000 (desde que haja espaço de memória para tal).

   Observação: Note que os métodos devem ordenar o mesmo vetor, portanto, tenha o cuidado de não ordenar o vetor por um método e passar o vetor ordenado como parâmetro de entrada para outro método de ordenação.

   Considerações:

   - Os algoritmos apresentados são abstratos, portanto devem ser adaptados para a linguagem C;
   - Uma forma de calcular o tempo de execução de uma operação ou função é utilizar a função `clock()`, mas sinta-se à vontade para utilizar outra solução.

Fazer o *uplod* dessa atividade no sítio da disciplina no moodle, identificado como: <NomeSobrenome> - APS02 Turma (A ou B).

```
1  Algoritmo Merge(A[1..n], p, q, r)
2   início
3     n1 <- q - p + 1;
4     n2 <- r - q;
5     criar vetores L[1..n1+1] e R[1..n2+1];
6     para i <- 1 até n1 faça
7          L[i] <- A[p+i-1];
8     fim_para;
9     para j <- 1 até n2 faça
10         R[j] <- A[q+j];
11    fim_para;
12    L[n1+1] <- infinito;
13    R[n2+1] <- infinito;
14    i <- 1;
15    j <- 1;
16    para k <- p até r faça
17       se L[i] <= R[j] então
18          A[k] <- L[i];
19          i <- i+1;
20       senão
21          A[k] <- R[j];
22          j <- j+1;
23       fim_se;
24    fim_para;
25  fim
```

Figura 3: Algoritmo abstrato Merge

```
1  Algoritmo Merge-sort(A[1..n], p, r)
2   início
3     se p < r então
4        q <- (p+r)/2;
5        Merge-sort(A, p, q);
6        Merge-sort(A, q+1, r);
7        Merge(A, p, q, r);
8     fim_se
9   fim
```

Figura 4: Algoritmo abstrato Merge-sort