# INF 112 – Prática 5, Insertion, Selection e Merge – 2018/2

**Exercício 0:** A implementação do *merge sort* que vimos em sala realiza a alocação de um vetor toda vez que a função *merge* é chamada. Modifique a função vista de forma que apenas uma alocação seja realizada.

Obs: A assinatura da sua função de ordenação (a que será usada por um usuário de sua função) deve ser a seguinte:

```
void mergeSort(int *v, int n);
```

**Exercício 1:** Modifique o código escrito no Exercício 0. Para vetores pequenos (e.g., tamanho no máximo 10), ao invés do *merge sort* fazer uma nova chamada recursiva, o *insertion sort* deve ser utilizado.

**Exercício 2:** (Para Casa) Gere um vetor de inteiros de tamanho *k* (*k* = 1000, 10000, 100000, 1000000). Para cada vetor, compare o tempo de execução das três implementações do *merge sort*: a vista em sala, a do Exercício 0 e a do Exercício 1.

**Exercício 3:** Escreva um programa, baseado no *selection sort*, para encontrar os *k* menores valores de um vetor de inteiros. Seu método deve ter complexidade $O(kn)$, onde *n* é o tamanho do arranjo.

**Exercício 4:** Em sala, vimos o conceito de inversão, a qual ocorre quando um par de elementos de um arranjo está fora de ordem. Faça um programa, baseado no *insertion sort*, para contar o número de inversões de um vetor de números inteiros.

**Exercício 5:** Repita o exercício anterior, mas agora use o *merge sort* como base (sem considerar a combinação do *merge sort* com o *insertion sort*).

**Exercício 6:** Até o momento, consideramos apenas o caso de ordenar números inteiros. Modifique os três métodos de ordenação (*insertion*, *selection* e *merge*) para ordenar *structs* do tipo definido abaixo

```
struct person {
	char primeiroNome[50];
	char ultimoNome[50];
	char cpf[11];
	int peso;
	int idade;
	double altura;
};
```

Sua função deve receber como argumento um ponteiro para uma função capaz de comparar *structs*. Leia esse material sobre ponteiros para funções (https://www.geeksforgeeks.org/function-pointer-in-c/). Tal função de comparação deve retornar: 0, se os elementos comparados forem iguais; 1, se o primeiro elemento for maior que o segundo; e -1, se o primeiro elemento for menor que o segundo. Teste seu método utilizando diferentes funções de comparação, uma para cada critério de comparação diferente. Exemplos: como ordenar as pessoas pelo primeiro nome? E pelo último nome? E pelo primeiro + último nome?

**Exercício 7:** (Para casa) Repita o exercício anterior para o caso de ordenação indireta, ou seja, ao invés de um arranjo de *structs*, sua função deve ordenar um arranjo de ponteiros para *structs*.

**Exercício 8:** (Para casa) Revise o conceito de ordenação estável (e.g., https://pt.wikipedia.org/wiki/Ordenação_estável ou https://www.geeksforgeeks.org/stability-in-sorting-algorithms/). Use os resultados do Exercícios 6 e 7 para, por meio de testes, fixar o conceito de estabilidade em ordenação.

**Exercício 9**: (Desafio) Leia sobre a função *qsort* do C (https://www.tutorialspoint.com/c_standard_library/c_function_qsort.htm). Faça uma implementação do *mergeSort* (estendendo a do Exercício 1) que funcione de forma genérica, assim como a *qsort*.

Obs: Em geral, a aritmética de ponteiros para *void* não funciona da mesma forma em diferentes compiladores. Sugestão, considere a versão do *gcc* do laboratório.