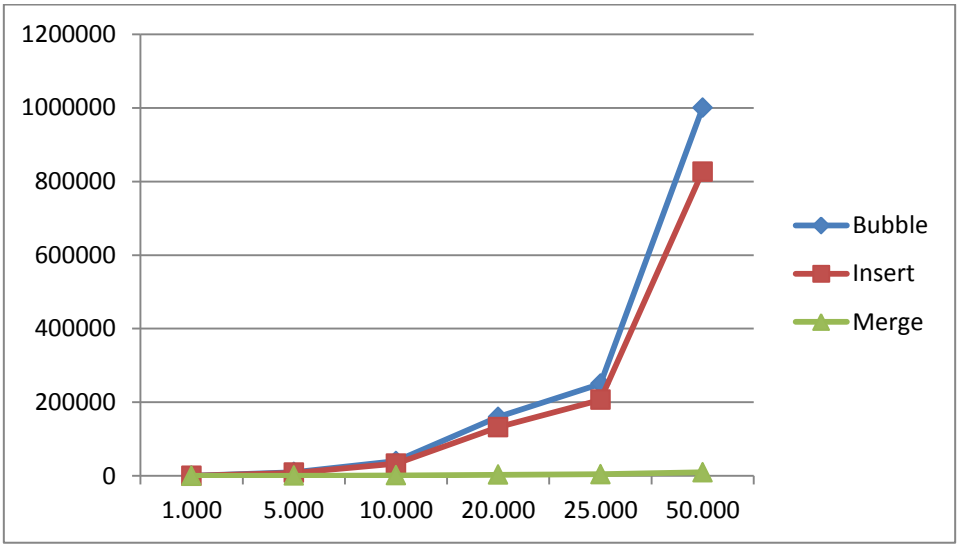
Universidade do Estado de Santa Catarina
Ciência da Computação
Complexidade de Algoritmos
Exercício Prático I

Exercício 1: Utilize o código fonte disponibilizado na página (sort.c) para realizar uma comparação de tempo de execução dos seguintes algoritmos de ordenação: **BUBBLE SORT**, **INSERTION SORT**, **MERGE SORT** e **QUICK SORT**. O algoritmo **QUICK SORT** deverá ser implementado de duas formas: uma que utiliza o **primeiro elemento como pivô**, e outra que utiliza um **elemento aleatório como pivô**.

Os algoritmos devem realizar a ordenação de uma estrutura de dados que representa um intervalo de tempo. Para ordenar um intervalo, considere que o inicio terá maior peso na ordenação, seguido pelo fim, com menor peso. Para os intervalos {13 – 15}, {12 – 17}, {12 – 20}, {10 – 15}, por exemplo, o resultado final deveria ser: {10 – 15}, {12 – 17}, {12 – 20}, {13 – 15}.

Faça a medição de tempo para cada algoritmo indicado, utilizando diferentes valores de N (tamanho do vetor), com pelo menos 15 testes de execução para obter o tempo médio. Faça um **gráfico** para cada um dos três tipos de entrada que são gerados pelo programa disponibilizado: vetor em ordem crescente, vetor em ordem decrescente e vetor com valores aleatórios. Os gráficos devem ilustrar a diferença do crescimento de tempo de execução de cada algoritmo para cada tipo de entrada, em relação ao número de elementos N (usar os valores de N no eixo X do gráfico, e os valores de tempo médio de execução no eixo Y do gráfico).

Exemplo de resultado para entradas aleatórias



Exercício 2: Adapte o algoritmo do **MERGE SORT** e do **QUICK SORT** de modo que eles exibam as suas chamadas recursivas conforme o exemplo abaixo (observe a endentação):

[77 , 33 , 55 , 22 , 66]

MERGE SORT

```
MergeSort( 0, 4 )
   MergeSort( 0, 2 )
      MergeSort( 0, 1 )
         MergeSort( 0, 0 )
         MergeSort( 1, 1 )
      33 77
      MergeSort( 2, 2 )
   33 55 77
   MergeSort( 3, 4 )
      MergeSort( 3, 3 )
      MergeSort( 4, 4 )
   22 66
22 33 55 66 77
```

QUICK SORT

```
QuickSort( 0, 4 )
pivo = 4 -> 77
66 33 55 22 77
   QuickSort( 0, 3 )
   pivo = 3 -> 66
   22 33 55 66 77
      QuickSort( 0, 2 )
      pivo = 0 -> 22
      22 33 55 66 77
         QuickSort( 0, -1 )
         QuickSort( 1, 2 )
         pivo = 1 -> 33
         22 33 55 66 77
            QuickSort( 1, 0 )
            QuickSort( 2, 2 )
      QuickSort( 4, 3 )
   QuickSort( 5, 4 )
```