**APS – (BCC34A) Análise de Algoritmos**
**Prof. Rodrigo Hübner**

**Descrição:**

Realizar a análise empírica dos algoritmos de ordenação e busca listados abaixo, considerando o **tempo de execução** e a **quantidade de instruções** contabilizadas por meio do laço de repetição (ou recorrência) mais interna de cada algoritmo. Cada algoritmo deve ser implementado em uma linguagem de programação de sua preferência. Justifique os métodos para medir o tempo de execução (pesquisar) para a linguagem selecionada. Devem ser gerados gráficos para comparar o tempo de execução e a quantidade de instruções dos algoritmos para diferentes valores de **N.** Façam a inicialização de **N** por meio de três arquivos (com 10.000, 100.000 e 1.000.000 de números separados por espaços simples) contendo números aleatórios de -999 a 999. Na discussão dos resultados, deve-se comparar a análise empírica com a análise matemática (assintótica).

**Algoritmos**:

- Bubble Sort;
- Bubble Sort c/ otimizações;
- Insertion Sort;
- Selection Sort;
- Merge Sort;
- Heap Sort;
- Quick Sort;
- Busca Binária;
- Busca do Vetor Máximo.

**Entrega**:

A APS poderá ser realizada **individualmente ou em duplas**. A entrega será realizada no dia **27 de novembro**. Deve ser entregue um pacote compactado por apenas um aluno do grupo contendo o relatório e os códigos desenvolvidos). **Não é necessário incluir o arquivo txt com os números**.

**Recursos:**

No moodle há materiais descrevendo a realização da análise empírica e exemplo de trabalho que compara diferentes algoritmos de ordenação.

**Exemplo**:

Exemplo com o algoritmo **`maior_elemento`** utilizando a linguagem de programação Python. O algoritmo principal está implementado da linha 3 até 12. A função para carregar o arquivo da linha 14 até 20. A instrução para o carregamento do vetor com 10.000 números acontece na linha 25. A contagem de instruções ocorre no laço principal da função por meio da variável global "`n_inst`" nas linhas 7 e 8. As instruções para contabilizar o tempo de execução ocorrem nas linhas 29 (importa a função "`time`" da biblioteca "`time`"), 30 (carrega o tempo inicial) e 32 (subtrai o tempo final pelo tempo inicial, após a execução da função).

```python
# coding: utf-8

def maior_elemento(v, n):
    maior = v[0]
    for i in range(1, n):
        # contabiliza instruções
        global n_inst
        n_inst += 1
        ##########################
        if v[i] > maior:
            maior = v[i]
    return maior

def carrega_vetor(arquivo):
    '''
    Carrega um vetor de números por meio de um arquivo com o seguinte formato:
    698 153 -649 192 -566 30 -808 630 -736 -369 758 -766 -896 517 514 -982 ...
    '''
    arq = open(arquivo)
    return [int(num) for num in arq.read().split(' ') if num != '']

if __name__ == '__main__':
    n_inst = 0
    # carregamento do vetor
    v1 = carrega_vetor('arquivo_10000.txt')
    # v2 = carrega_vetor('arquivo_100000.txt')
    # v3 = carrega_vetor('arquivo_1000000.txt')
    # calcula tempo de execução somente fa dunção (sem carregamento dos dados)
    from time import time
    tempo_inicial = time()
    maior = maior_elemento(v1, len(v1))
    tempo_final = time() - tempo_inicial
    print "Tempo de execução: %0.3f segundos" % tempo_final
    print "Quantidade de instruções: %d" % n_inst
```