Trabalho Prático 1 - TP1 - Algoritmos e Classificação de Dados (ACD)

Projeto prático de *programação*, *experimentação*, *análise* e *apresentação* de resultados de desempenho de algoritmos de ordenação. O trabalho poderá ser realizado em duplas (excepcionalmente, trios) e com peso de 18% na média do componente de ACD. O TP1 envolverá os ALGORITMOS de CLASSIFICAÇÃO ou ORDENAÇÃO estudados no componente. De forma resumida, o trabalho TP1 envolverá a implementação em Java de diferentes algoritmos de ordenação (a saber: *bubblesort*, *insertionsort*, *selectionsort*, *shellsort*, *heapsort*, *mergesort* e *quicksort*; além dos algoritmos de tempo linear: *countingsort*, *radixsort* e *bucketsort*), o planejamento e a execução de experimentos visando a comparação de desempenho (tempo de execução) destes algoritmos, relato técnico (na forma de um relatório ou artigo) dos experimentos e seus resultados (apresentação das médias dos tempos de execução de cada algoritmo e respectivos gráficos para comparação entre os métodos de ordenação), seguidos da análise dos resultados, os relacionando, também, com a complexidade (notação assintótica ***O***) de cada algoritmo, conforme análises realizadas nas aulas, e com aspectos específicos da implementação de cada método de ordenação. O trabalho TP1 será computado como atividade EaD, com carga horária vinculada de 5 horas. O trabalho deverá ser apresentado pelo(s) discente(s) para o docente e, preferencialmente, também para a turma (conforme cronograma do componente).

Passo-a-Passo/Detalhes do Trabalho

1. Os métodos de ordenação ***bubblesort***, ***insertionsort***, ***selectionsort***, ***shellsort***, ***heapsort***, ***mergesort*** e ***quicksort***; além dos métodos *countingsort*, *radixsort* e *bucketsort*, precisam estar devidamente implementados na linguagem Java. Assim, para o bom desenvolvimento do trabalho, é necessário que cada grupo tenha a implementação destes métodos devidamente realizada e testada. Assim, para cada algoritmo de ordenação, a orientação é que os mesmos sejam implementados no Java como métodos estáticos (de classe) e que recebam *arrays* (vetores) de valores que sejam de tipos comparáveis (*Comparable*) - desta forma, espera-se o uso da interface *Comparable* da API Java. Exemplo: Classe **Ordenadores** com métodos *static* para cada algoritmo, conforme exemplo abaixo:

```
public class Ordenadores{
        public static <T extends Comparable<T>> void quicksort(T[ ] a, int lo, int hi)
        /* Cada método deve ter o seu conjunto adequado de parâmetros, mas sempre
        recebendo um array de Comparable como sendo os elementos a serem ordenados.
        Um método pode ter métodos auxiliares, conforme estudado em nossas aulas*/
        {...}
        public static <T extends Comparable<T>> void heapsort(T[ ] a, int n)
        /* */
        {...}
        //... demais métodos de ordenação (e métodos auxiliares, quando existirem)
}
```

É necessário que existam alguns vetores de teste de correção (de poucos elementos: <= 10) dos métodos e que estes possam ser executados facilmente para demonstração da “correção” dos métodos. Sabe-se que não é possível gerar casos de teste que possam garantir a corretude de um dado algoritmo, mas espera-se aqui que existam vetores de poucos elementos que possam demonstrar certa confiabilidade de que a implementação dos métodos está correta. Assim, sugere-se que sejam preparados pelo menos 4 vetores de teste de correção (de tamanhos diferentes - par e ímpar), a saber: i) com elementos em ordem crescente; ii) com elementos em ordem decrescente; iii) com elementos gerados aleatoriamente, e iv) com vários elementos repetidos no vetor.

Chama-se a atenção que os algoritmos *countingsort*, *radixsort* e *bucketsort* precisarão ser tratados de forma específica e separada, já que os mesmos pressupõem informações extras sobre os elementos do vetor a ser ordenado e, exatamente por este motivo, atingem tempos de execução de ordem linear. Estes três algoritmos deverão ser experimentados e analisados separadamente dos demais métodos. Fica a cargo de

cada grupo o planejamento, organização, execução e relato dos experimentos realizados com estes métodos de ordenação de tempo linear.

2. Preparar vetores de entrada para serem utilizados nos testes de ordenação dos métodos. Os vetores de teste terão algum tipo de variação quanto a algumas de suas características, quais sejam: i) **ordenação dos elementos** - elementos em ordem crescente, em ordem decrescente e em ordem aleatória; ii) **tamanho do vetor** - embora esteja sujeito a testes por parte de cada grupo, a fim de identificar o menor e o maior tamanho de vetor a serem utilizados, sugere-se que sejam gerados vetores de tamanhos na ordem de 100, 1000, 10000, 100000 e 1000000 de elementos (podendo chegar a valores maiores, dependendo de testes realizados pelos grupos). Sobre os vetores com elementos em ordem crescente e decrescente, os mesmos podem ser testados somente com um tamanho de vetor (sugere-se um tamanho intermediário entre aqueles utilizados no experimento). Já os vetores com elementos gerados de forma aleatória, chama-se a atenção que os mesmos, em seus diferentes *tamanhos*, devem ser aplicados de forma coerente, isto é, o mesmo vetor de determinado tamanho deve ser aplicado para cada um dos métodos de ordenação em avaliação.

3. Preparar código auxiliar para permitir a devida medição e o devido registro dos tempos de execução de cada algoritmo. A medição que deve ser feita corresponde somente ao processo de ordenação em si. Assim, sugere-se que sejam preparados os devidos trechos de código que vão marcar o tempo de início de execução, disparar a execução de determinado método de ordenação e o registro do tempo de fim de execução (depois que o método termina). Sugere-se o seguinte esquema para este passo:

```
Timer t_ini = Time(); // Marca o tempo de início. Timer: registro do tempo. Time(): retorna o tempo.
sort(V,...);  // Invoca o método de ordenação. “sort” representa o nome do método de ordenação.
Timer t_fim = Time();  // Marca o tempo de fim.
register(“sort”, t_ini, t_fim);  // Método de registro do tempo de execução do algoritmo “sort”.
```

Para a medição dos tempos, avalie o uso dos seguintes métodos: ***currentTimeMillis()*** e ***nanoTime()*** - ambos da classe **System** da API java. A escolha do método vai depender da precisão necessária - o primeiro possibilita medidas *milissegundos*, enquanto o segundo em *nanosegundos*.

4. Preparar o código de execução e registro de tempos para que sejam executados um determinado número de vezes, sempre considerando condições o mais semelhantes possíveis em cada execução - por exemplo, sempre fornecer o mesmo vetor de entrada, sempre marcando o tempo nos momentos adequados, etc. Como em qualquer experimento que envolva a execução de código, podem haver variáveis totalmente fora do contexto do experimento que influenciam o mesmo - por exemplo, o sistema operacional pode executar algum processo em paralelo juntamente com determinado algoritmo de ordenação, influenciando o tempo de execução registrado para o algoritmo. Assim, sugere-se que sejam tomadas todas as medidas possíveis para deixar o computador onde os experimentos serão executados destinado unicamente para esta tarefa. Entretanto, procurando evitar distorções de alguma execução anômala, cada método de ordenação, em cada configuração a ser testada (considerando as diferentes ordenações dos elementos do vetor, os diferentes tamanhos do vetor de entrada, etc.), deverá ser executada pelo menos **10** vezes (**30** seria um número mais adequado estatisticamente falando). O tempo de execução associado a cada método e configuração será a média destas execuções. Na realidade, os tempos de todas as execuções de cada método/configuração poderão ser considerados nos gráficos se forem utilizados os chamados *boxplots* (diagramas de caixa) para a apresentação dos resultados (na realidade outros tipos de gráficos - como de dispersão (*scatter plot*) e de barras (*bar plot*) também poderão ser utilizados na apresentação dos resultados). Não esqueça: *Registrar os tempos de todas as execuções de todas as configurações de cada método de ordenação*.

5. Relatar em um documento técnico, na forma de relatório ou mesmo artigo científico, o experimento realizado. Descrever devidamente o experimento no relato, incluindo:

- A descrição do escopo do experimento.
- Uma breve descrição descrição de cada algoritmo de ordenação em análise;
- Metodologia de planejamento e execução do experimento
- Apresentação dos resultados obtidos e Análise dos mesmos.
- Conclusões

Consulte o professor sobre como relatar o experimento em um documento científico. Uma boa estratégia é compartilhar o documento que o grupo for editar para que as dúvidas possam ser apresentadas e discutidas com maior eficiência.

6. Preparar a apresentação do trabalho perante o docente e a turma, considerando um tempo de apresentação em torno de 10 minutos. Durante a apresentação, descrever rapidamente a metodologia empregada no experimento, seguindo para a apresentação dos resultados e da respectiva análise e, por fim, das conclusões. A apresentação em si poderá acontecer em uma de nossas aulas. Porém, devido aos vários ajustes que foram necessários ao plano de ensino da componente, devido ao cancelamento de aulas que ocorreram, poderemos buscar alternativas como apresentação em horário extraclasse ou mesmo geração de vídeo de apresentação. Discutiremos as alternativas no decorrer das próximas aulas.

Sobre o escopo do experimento, o mesmo pode ser descrito, resumidamente, usando o seguinte template[1]:

Analyze <Object(s) of study>
for the purpose of <Purpose>
with respect to their <Quality focus>
from the point of view of the <Perspective>
in the context of <Context>.

Assim, no contexto descrito para este trabalho, podemos descrever o escopo do experimento da seguinte forma:

Analisar **<os métodos de ordenação de dados>**
para o propósito de **<avaliação>**
com respeito ao **<tempo de execução de cada método>**
do ponto de vista do(a) **<programador>**
no contexto de **<estudos do componente de Algoritmos e Classificação de Dados>**.

Observações gerais:

O docente da disciplina fica à disposição dos grupos para discutir dúvidas sobre o trabalho e sobre a metodologia para a realização dos experimentos. Procurem consultar o professor para discutir seu trabalho. Variações poderão ser testadas pelos grupos em seus experimentos. Por exemplo, pode ser elencado um determinado tamanho de vetor de entrada e serem testados algoritmos de ordenação adaptados para lidar especificamente com dados de tipo primitivo (*int*, por exemplo) e realizar experimento para analisar se o uso de um tipo mais genérico para os dados de entrada (Comparable, neste caso) tem alguma influência sobre o tempo de execução dos métodos. Outro exemplo seria verificar qual o desempenho de algum algoritmo modificado em comparação aos métodos originais, ou mesmo em relação a um método em específico - poderia ser avaliado o desempenho do método que agrupa a *ordenação por intercalação* e a *ordenação por inserção* (uma questão relacionada apareceu em nossa avaliação) em comparação aos métodos originais. Também haveria a possibilidade de propor outras adaptações nos métodos existentes e se fazer a avaliação desta com os métodos originais (a criatividade de cada grupo é o limite neste aspecto).

[1] Apresentado no livro "*Experimentation in Software Engineering*", de Wohlin et al. PDF do livro disponível na internet.