Universidade Federal de Mato Grosso do Sul
Faculdade de Computação
Disciplina: AlgProgOO II
Professor: Anderson V. de Araujo

Projeto de AlgProgOO II

Descrição

Desenvolver um projeto em Java que compare os métodos de ordenação (apresentados abaixo) e busca binária **COM** uma busca sequencial em vetor desordenado.

Ou seja: (Algoritmo de Ordenação + Busca Binária) **x** (Vetor desordenado + Busca Sequencial).

Algoritmos:

a) Bubble sort (não otimizado);
b) Bubble sort (otimizado);
c) Insertion sort (não otimizado);
d) Insertion sort (otimizado);
e) Selection sort;
f) Merge sort;
g) Quick sort;
h) Outro algoritmo de ordenação, não apresentado em sala, a sua escolha.

Nas comparações, verificar:

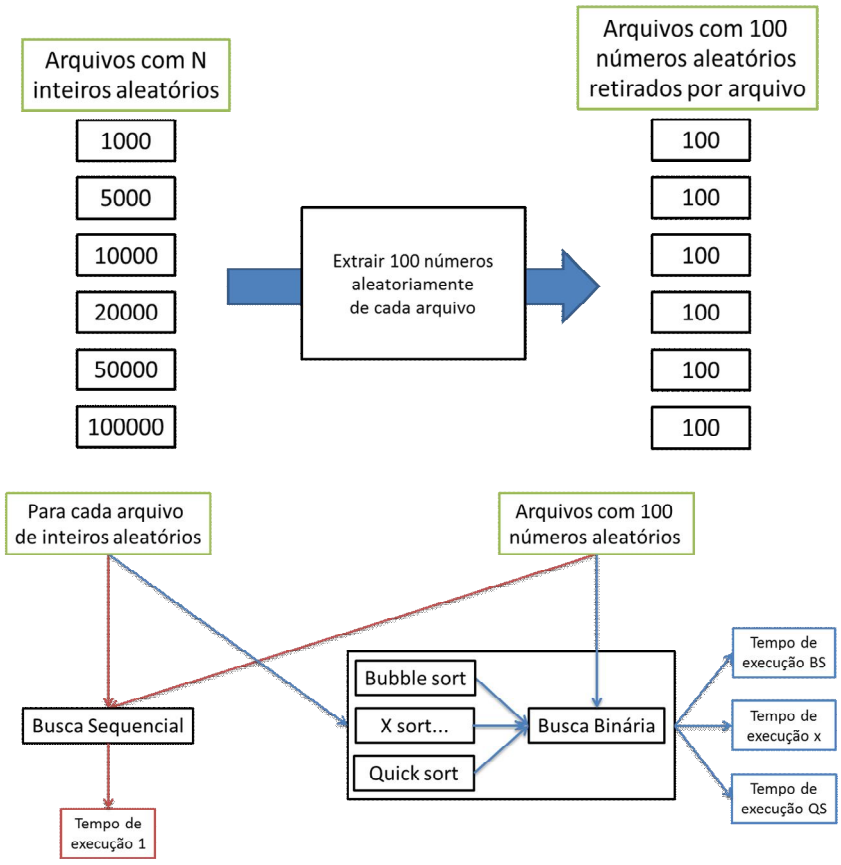
a) O tempo de execução total;
b) Quantidade de comparações efetuadas no algoritmo de ordenação;
c) Quantidade de trocas efetuadas no algoritmo de ordenação.

Fazer a comparação para diferentes tamanhos de vetor:

a) 1000
b) 5000
c) 10000
d) 20000
e) 50000
f) 100000

Para gerar os vetores com os tamanhos apresentados, criar um gerador de arquivos com números inteiros aleatórios (**NÃO** podem existir números repetidos). Cada arquivo vai corresponder a um tamanho de vetor dos seis apresentados anteriormente.

Além dos arquivos de números aleatórios, outro arquivo deve ser criado com 100 números aleatoriamente colhidos de cada uma das massas. Não podem ser os primeiros 100 números do arquivo, pois desta forma a busca sequencial vai ser muito rápida. O seu programa deve ler os arquivos com os números inteiros e executar as buscas.



RESUMINDO:

1. Gerar um arquivo [Arquivo Vetor] para cada tamanho com números inteiros aleatórios;
2. Gerar um arquivo [Arquivo de Busca] para cada tamanho com 100 números inteiros retirados aleatoriamente de cada [Arquivo Vetor];
3. Executar a busca sequencial sobre o [Arquivo Vetor] para cada elemento do [Arquivo de Busca] e armazenar o tempo total de busca;
4. Executar os algoritmos de ordenação para cada [Arquivo Vetor], armazenando o tempo total de execução, o número de comparações e o número de trocas;
5. Executar a busca binária sobre o vetor ordenado para cada algoritmo de ordenação e cada elemento do [Arquivo de Busca];
6. Gerar o relatório com os dados obtidos durante a ordenação **E** dados da comparação **entre** ordenar o vetor (usando os algoritmos) e realizar a busca binária e **NÃO** ordenar e fazer busca sequencial.

Relatório

Relatório em PDF apresentando:

- Os resultados das comparações entre os algoritmos de ordenação e as duas abordagens de busca. Use gráficos e tabelas para a comparação;
- Uma descrição do funcionamento do algoritmo extra (h);
- A complexidade Big Oh (O) de todos os algoritmos utilizados – inclusive as buscas;
- Um ranking geral dos algoritmos e justificativas das posições de cada abordagem.

Não é necessário entregar o relatório impresso. O relatório deve **obrigatoriamente** ter entre 5 e 10 páginas.

Entrega

Entregar o projeto do eclipse (ou outra IDE) zipado com todos os códigos desenvolvidos, o PDF do relatório **e todos** os arquivos gerados para os testes. O projeto deve ser desenvolvido **em dupla ou individualmente**. No caso da dupla, a nota vai ser a mesma para ambos os alunos. Basta que apenas um da dupla faça a entrega no Moodle. A data de entrega será definida no Moodle.

Avaliação

A avaliação do trabalho vai ser realizada de acordo com a implementação e o relatório entregue. No total a nota do trabalho é no máximo 10 (verificar o peso do trabalho na nota final no SISCAD).

Se tiver algum código copiado de outro trabalho é zero para os dois trabalhos.

Quaisquer dúvidas no desenvolvimento do trabalho devem ser enviadas para o e-mail **andvicoso@facom.ufms.br** ou pessoalmente.

Bom trabalho!

Dicas e Observações Importantes

1. Tem que comparar a busca dos 100 elementos dos arquivos de busca através das duas maneiras:
   - Buscar os 100 elementos sequencialmente e somar todos os tempos no [Arquivo Vetor] --> TempoBuscaLinear1001000, TempoBuscaLinear1005000, TempoBuscaLinear10010000, ...;
   - Para cada algoritmo de ordenação: Somar o tempo de ordenar o vetor com o algoritmo + buscar os 100 elementos usando busca binária --> TempoOrdBuscaBin1001000, TempoOrdBuscaBin1005000, TempoOrdBuscaBin10010000, ...

Com isso, comparar esses valores finais. Algumas das conclusões interessantes são: vale a pena ordenar e fazer a busca binária? Se sim, usando qual algoritmo? Para quais tamanhos de vetores não vale a pena ordenar? Qual algoritmo de ordenação fez menos trocas? Menos comparações? Qual foi mais rápido? Para que tamanho de vetor? Explique as respostas para estas perguntas no relatório.

2. Para gerar os arquivos de busca (com 100 elementos) não pode pegar os 100 primeiros elementos do arquivo dos vetores, pois neste caso a busca linear vai ser muito mais rápida. Pois vai encontrar os elementos logo no começo do vetor desordenado.

3. Além disso, seria melhor para os resultados se não existissem números repetidos nos vetores grandes. Pois para estes casos, a busca linear também vai ser melhor, ela sempre vai pegar o inteiro mais próximo na busca pelo vetor.

4. Devem ser contadas apenas as comparações e trocas para os algoritmos de busca, e não para a busca binária e a linear.

5. O Mergesort não chama o método *swap* (troca) diretamente, mas faz uma cópia para o vetor temporário e depois volta para o vetor original. Considerar cada vez que ele faz isso uma troca. Ou seja, toda vez que copiar de volta o elemento para o vetor original, conta como uma troca.

6. Uma alteração que pode deixar o merge sort mais rápido é colocar a criação do vetor auxiliar fora do método merge, por exemplo, dentro da chamada do método search. Com isso, não fica criando o vetor auxiliar toda hora, já que ele vai ser sempre do mesmo tamanho. Exemplo:

```
class MergeSort extends AbstractSortingAlgorithm{
    int[] tempMergArr;
```

```
    public void sort(int[] v) {
        tempMergArr = new int[v.length];
        sort(v, 0, v.length - 1);
    }
...
```