Universidade Cruzeiro do Sul	***Cursos: Bacharelado em Ciência da Computação*** 2º Semestre/2018	
Nome da Disciplina: Computabilidade e Complexidade de Algoritmos		Série: 5º/6° Período
Prof. Dr. Ivan Carlos Alcântara de Oliveira		

Estudo Prático de Complexidade

Algoritmos de Ordenação

A ordenação de dados é uma tarefa comum em instituições acadêmicas, empresas e atividades gerais. Perguntas interessantes a respeito desse assunto poderiam ser:

- Qual o método mais adequado para um grande conjunto de dados?
- Há alguma configuração dos dados que favorece um melhor desempenho para os algoritmos?
- Qual a influência do hardware na melhoria do tempo de execução dos algoritmos?
- Métodos rápidos em computadores com processadores mais lentos são melhores que métodos menos eficientes em computadores com bons processadores?

Na tentativa de auxiliar a responder a essas perguntas, este trabalho tem o objetivo de realizar dois experimentos sobre métodos de ordenação não recursivos e recursivos e avaliá-los sobre determinadas condições. Os métodos são:

a. Método da Bolha;
b. Método da Seleção Direta;
c. Método Inserção Direta;
d. Método MergeSort;
e. Método HeapSort;
f. Método QuickSort (Recursivo).

No Anexo, podem ser encontrados os algoritmos correspondentes a esses métodos.

Para auxiliá-lo, foi elaborado um programa Java (Ordena_2018.java) que realiza a implementação dos métodos e a geração dos dados para uso nos experimentos. O programa possui um menu de opções, apresentado na Figura 1, que possibilita selecionar a:

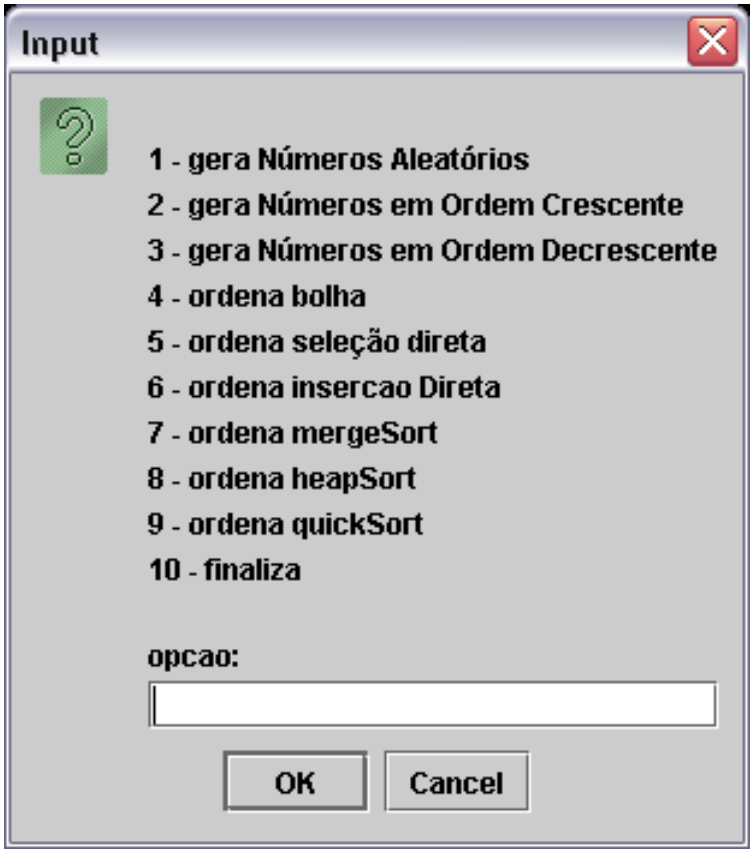
1. Geração aleatória de um conjunto de dados desordenado com um tamanho a ser informado e gravar em um arquivo cujo nome também deve ser informado;
2. Geração aleatória de um conjunto de dados e ordenar em ordem crescente, com um tamanho a ser informado e gravar em um arquivo cujo nome também deve ser informado;
3. Geração aleatória de um conjunto de dados e ordenar em ordem decrescente, com um tamanho a ser informado e gravar em um arquivo cujo nome também deve ser informado;

4. Ordenação pelo método da Bolha;
5. Ordenação pelo método da Seleção Direta;
6. Ordenação pelo método da Inserção Direta;
7. Ordenação pelo método MergeSort;
8. Ordenação pelo método HeapSort;
9. Ordenação pelo método QuickSort;
10. Finaliza.

Se a opção de Ordenação, por algum dos métodos (opção de 4 a 9), for selecionada:

a) É obtido do usuário o nome de um arquivo de dados contendo os dados gerados por meio da opção 1, 2 ou 3 do menu;
b) Os dados do arquivo são lidos para um vetor;
c) É realizada a ordenação pelo método escolhido; e
d) Contabilizado o tempo gasto no intervalo entre a entrada no método e após o término da ordenação.

Figura 1. Menu de Opções do Programa Java do Trabalho Prático.



Fonte: Captura de Tela do Programa.

Os seguintes conjuntos de dados devem ser gerados antes da realização dos experimentos:

a) Conjunto aleatório, contendo: 50, 100, 250, 500, 1000, 5000, 10000, 20000, 40000, 90000, 120000, 400000, 600000, 800000, 1000000 elementos.
b) Conjunto em ordem crescente, contendo: 50, 100, 250, 500, 1000, 5000, 10000, 20000, 40000, 90000, 120000, 400000, 600000, 800000, 1000000 elementos.
c) Conjunto em ordem decrescente, contendo: 50, 100, 250, 500, 1000, 5000, 10000, 20000, 40000, 90000, 120000, 400000, 600000, 800000, 1000000 elementos.

1º Experimento

i. Em um único computador, realizar os testes de todos os métodos para as massas de dados geradas previamente (ou seja, os conjuntos de dados gerados de forma aleatória, em ordem crescente e em ordem decrescente) e obter o tempo de execução para cada conjunto de dados e método de ordenação;
ii. Montar uma planilha no Excel que ilustre os resultados dos tempos obtidos em uma tabela e os respectivos gráficos. São três gráficos:
    a. Tempo de cada método x Conjuntos Aleatórios.
    b. Tempo de cada método x Conjuntos em Ordem Crescente.
    c. Tempo de cada método x Conjuntos em Ordem Decrescente.

Ao montar os gráficos no Excel, interligue os pontos como se fosse uma função contínua.

2º Experimento

i. Em dois computadores com configurações (processadores) bem diferentes, ou seja, um mais rápido e outro mais lento (por exemplo, um Pentium);
ii. Testar o método da Bolha, da Seleção Direta e da Inserção Direta, utilizando SOMENTE as massas de dados dos conjuntos ALEATÓRIOS no computador mais rápido e obter o tempo de execução;
iii. Testar o método Quicksort, MergeSort e HeapSort, utilizando SOMENTE as massas de dados dos conjuntos ALEATÓRIOS no computador mais lento e obter o tempo de execução;
iv. Montar uma tabela que ilustre os resultados dos tempos obtidos e os respectivos gráficos no Excel. Os gráficos a serem montados são;
    a. Computador Rápido: Tempo obtido com os métodos x Conjuntos ALEATÓRIOS.
    b. Computador Lento: Tempo obtido com os métodos x Conjuntos ALEATÓRIOS.

Ao final dos experimentos elaborar um relatório contendo os itens:

1) Capa: com o nome, RGM dos integrantes e Campus.
2) Sumário.

3) Introdução: contendo uma introdução ao assunto análise de algoritmos e notação assintótica, além de descrever o conteúdo das próximas seções.
4) Tabela de O-grande para os algoritmos de ordenação (melhor caso, caso médio e pior caso): Seleção Direta, Bolha, Inserção Direta, QuickSort, MergeSort e HeapSort.
5) Análise do 1o Experimento
    a) Situação de teste: descrição do teste realizado, informando o hardware, o sistema operacional e, se for o caso, programas que compartilharam a execução do teste.
    b) Tabela dos tempos para os métodos (conjunto aleatório, em ordem crescente e decrescente).
    c) Gráficos dos resultados.
    d) Análise crítica/discussão dos resultados: discutir os resultados obtidos explicando/comentando/comparando os algoritmos, os tempos obtidos para os conjuntos (aleatório, em ordem crescente e em ordem decrescente) e os seus gráficos.
6) Análise do 2o Experimento
    a) Situação de teste: descrição do teste realizado, informando o hardware, o sistema Operacional e, se for o caso, programas que compartilharam a execução do teste.
    b) Tabela com os resultados dos tempos dos métodos para os conjuntos ALEATÓRIOS utilizados.
    c) Gráfico dos resultados.
    d) Análise crítica dos resultados: discutir os resultados obtidos explicando/comentando/comparando os algoritmos, o hardware utilizado, a situação do teste e os seus gráficos.

7) Considerações Finais: deve conter um resumo do experimento, síntese das análises críticas e uma conclusão envolvendo todos os métodos de ordenação.

Este trabalho deve ser realizado em grupo de 2 a 6 pessoas.

Data de entrega do relatório: 25 do mês de novembro até as 23h59min.

Valor Máximo do Trabalho: 2.0 pontos na A2.

A entrega do relatório deve ser na forma digital em área específica do Blackboard.

Anexo
Algoritmos dos Métodos de Ordenação

a) Método da Bolha Original

```
procedimento bolha (inteiro conjunto[ ],  inteiro qtde)
Entrada: - qtde: número de elementos do conjunto;
               - conjunto: vetor com os dados desordenado.
Saída: - conjunto: vetor ordenado.
início
  para (i ← 0; i < qtde – 1; i++) faça
      para (j ← 0; j < qtde – i-1; j++) faça
         Se ( conjunto [j] > conjunto [j+1] ) então
            // permuta conjunto [j]  com conjunto [j+1]
            aux ← conjunto [j];
            conjunto [j] ← conjunto [j+1];
            conjunto [j+1] ← aux;
fim.
```

b) Método da Seleção Direta

```
procedimento selecaoDireta (inteiro conjunto[ ],  inteiro qtde)
Entrada: - qtde: número de elementos do conjunto;
               - conjunto: vetor com os dados desordenado.
Saída: - conjunto: vetor ordenado.
início
  inteiro i, indConjunto, j, maior;
  para (i ← n-1; i > 0; i--) faça
     maior ← conjunto[0];
     indConjunto = 0;
     para (j ← 1; j <= i; j++) faça
        se ( conjunto[j] > maior) então
            maior ← conjunto[j];
            indConjunto ← j;
     conjunto[ indConjunto ] ← conjunto [ i ];
     conjunto[ i ] ← maior;
fim.
```

b) Método Inserção Direta

```
Método InsertionSort (A, tamanho = n)
// A = vetor de 0 a n – 1, n >=1
início
para j ← 1 até n-1 faça
  chave ← A[j];
  i ← j – 1;
  // Insere A[j] na seq. Ord. A[0..j-1]
  enquanto i >= 0 e A[i] > chave faça
    A[i+1] ← A[i];
    i ← i - 1;
  A[i+1] ← chave;
fim
```

d) Método MergeSort

```
procedimento intercala(inteiro p, inteiro q, inteiro r, inteiro conjunto[ ] )
Entrada: - p, q, r valores inteiros;
               - conjunto: vetor com os dados.
Saída: - conjunto: vetor modificado.
início
      inteiro i, j, k,  w [ r-p ];
      i ← p; j ← q; k ← 0;
```

```
      enquanto (i < q e j < r) faça
         se (conjunto[i] <= conjunto[j])  w[k++] ← conjunto[i++];
         senão  w[k++] ← conjunto[j++];
      fim-enquanto
      enquanto (i < q)  faça
           w[k++] ← conjunto[i++];
       fim-enquanto
      para(i ← p; i < j; ++i)
          conjunto[i] ← w[i-p];
      fim-para
fim.
```

procedimento *MergeSort*(inteiro conjunto[ ], inteiro qtde)
Entrada: - qtde: número de elementos do conjunto;
- conjunto: vetor com os dados desordenado.
Saída: - conjunto: vetor ordenado.

```
início
   inteiro p, r;
   inteiro b ← 1;
   enquanto (b < qtde)  faça
       p ← 0;
      enquanto (p + b < qtde) faça
          r ← p + 2*b;
          se (r > qtde) r ← qtde;
          fim-se
          intercala (p, p+b, r, conjunto);
          p ← p + 2*b;
      fim-enquanto;
      b ← 2*b;
   fim-enquanto;
fim.
```

e) Método HeapSort
procedimento *peneira*(inteiro p, inteiro m, inteiro conjunto[ ])
Entrada: - p, u: valores inteiros;
- conjunto: vetor.
Saída: - conjunto: vetor.

```
início
   inteiro x ← conjunto[p];
   enquanto (2*p <= m) faça
       inteiro f ← 2*p;
       se (f < m e conjunto[f] < conjunto[f+1])  ++f;
       se (x >= conjunto[f]) break;
       conjunto[p] ← conjunto[f];
       p ← f
   fim-enquanto
  conjunto[p] ← x;
fim.
```

procedimento *heapSort*(inteiro conjunto[ ], inteiro qtde)
Entrada: - qtde: número de elementos do conjunto;
- conjunto: vetor com os dados desordenado.
Saída: - conjunto: vetor ordenado.

```
início
   inteiro p, m, x;
   para (p ←  0; p < qtde; p++) faça
       conjunto [p+1] ← conjunto[p];
   fim-para
   para (p ← qtde/2; p >= 1; --p)
      peneira (p, qtde, conjunto);
   para (m ← qtde; m >= 2; --m)
      x ← conjunto[1]; conjunto[1] ← conjunto[m]; conjunto[m] ← x;
      peneira (1, m-1, conjunto);
```

```
   para (p ←  1; p <= qtde; p++) faça
       conjunto [p-1] ← conjunto[p];
   fim-para
fim.
```

f) Método Quicksort

```
void quickSort( int conjunto[ ], int início, int fim){
       int pivo, trab, i, j;
       i ← inicio;
       j ← fim;
       pivo ← conjunto [ (i + j) / 2 ];
       do{
         while (conjunto [i] < pivo) i++;
         while (conjunto [j] > pivo) j--;
         if (i <= j) {
         trab ← conjunto[i];
         conjunto[i] ← conjunto[j];
         conjunto[j] ← trab;
         i ← i + 1;
         j ← j - 1;
       }
       } while ( i <= j );
       if (inicio < j) quickSort( conjunto, inicio, j);
       if (fim > i ) quickSort ( conjunto, i, fim);
}
```