INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

# Estruturas de Dados e Algoritmos
# Guias de Aulas Práticas

Doutor José Jasnau Caeiro

Beja
13 de Março de 2016

**Instituto Politécnico de Beja**
**Escola Superior de Tecnologia e Gestão**
**Departamento de Engenharia**

# Estruturas de Dados e Algoritmos

**Guias de Aulas Práticas**

**Elaborado por:**
**Doutor José Jasnau Caeiro**

**Beja**
**13 de Março de 2016**

# Conteúdo

# List of listings

# 1 Guia 1

O aluno deve criar uma pasta com a designação `guia1` no seu repositório de controlo de versões.

O aluno deve programar na linguagem de programação `Java` os algoritmos de ordenação `Insertion-Sort` e `Bubble-Sort`.

## 1.1 Realização de Gráficos

A aplicação **gnuplot** permite a realização de gráficos com grande qualidade. Existem versões disponíveis para várias sistemas operativos.

Instale a versão apropriada para o seu sistema operativo e realize os seguintes gráficos:

1. o gráfico da função $\sin(x)$ para o intervalo entre 0 e $2\pi$;
2. acrescente uma grelha;
3. acrescente um título ao gráfico;
4. acrescente designações ao eixo das abscissas e ao eixo das ordenadas;
5. grave em ficheiro PDF;
6. represente duas funções em simultâneo, por exemplo, as funções $\sin(x)$ e $\cos(x)$.

## 1.2 Sequências de Fibonacci

A sequência de *Fibonacci* é definida pela expressão:

$$F_n = F_{n-1} + F_{n-2},$$

Os valores iniciais são $F_0 = 0$ e $F_1 = 1$.

1. Programe as versões recursivas e iterativa das sequências de Fibonacci;
2. meça os tempos de execução
3. estude o funcionamento da aplicação `gnuplot` para a exibição de gráficos gerados a partir de entradas com ficheiros de dados experimentais;
4. realize o gráfico dos tempos de execução para as duas versões da aplicação que calcula as sequências de *Fibonacci*.

## 1.3 Bubble Sort

Os algoritmos de ordenação Bubble-Sort e Insertion-Sort apresentam uma taxa teórica do crescimento do tempo de execução assintonticamente limitado superiormente de $O(n^2)$.

Programe os algoritmos de ordenação Bubble-Sort e Insertion-Sort. Realize gráficos de tempos de execução previstos teoricamente para cada algoritmo, compare os seus resultados com os resultados experimentais e produza um gráfico único com o valor teóricamente previsto e os valores experimentais.

O aluno deve medir os tempos de execução dos algoritmos.

1. Deve escolher um conjunto de dimensões de tabelas a ordenar;
2. Deve preencher as tabelas com números aleatórios;
3. Deve medir pelo menos 50 vezes o tempo de execução do procedimento de ordenação para cada dimensão da tabela;
4. Deve estimar a média dos tempos de execução para cada dimensão da tabela;
5. Deve estimar o desvio padrão para cada dimensão da tabela.

A escolha das dimensões da tabela deve ser adequada ao computador e garantir tempos de execução suficientemente elevados.

O aluno deve representar graficamente os tempos de execução com a ferramenta `gnuplot`. Deve comparar os tempos de execução experimentais com os tempos de execução previstos teoricamente.

# 2 Guia 2

O aluno deve criar uma pasta com a designação `guia2` no seu repositório de controlo de versões.

O aluno deve programar na linguagem de programação `Java` o algoritmo de ordenação `Merge-Sort`.

## 2.1 Programação do Algoritmo Mergesort

O algoritmo MERGE-SORT é um exemplo do processo de projeto de algoritmos conhecido por *divisão e conquista*. Este algoritmo está representado nas Figuras 2.1 e 2.2. Uma realização com a linguagem de programação `Java` apresenta-se na Figura **??**.

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3     L[1 . . n1 + 1]  e   R[1 . . n2 + 1] novas tabelas
 4  for i = 1 to n1
 5        L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7        R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13        if L[i] ≤ R[j]
14              A[k] = L[i]
15              i = i + 1
16        else A[k] = R[j]
17              j = j + 1
```

Figura 2.1: Algoritmo MERGE-SORT no que se refere à função MERGE.

Compile o código do programa e estude o seu funcionamento realizando as seguintes tarefas:

1. modifique o código de modo a ordenar uma tabela de 100 números reais gerada aleatoriamente;

2. crie uma versão que possa ordenar os objetos de uma classe designada por `Ponto` que representa pontos no espaço tridimensional e cujo termo para comparação seja a distância à origem;

3. crie uma versão que opere com listas;

4. crie uma versão que funcione independentemente do tipo da tabela;

5. experimente criar uma versão noutra linguagem de programação.

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

Figura 2.2: Algoritmo MERGE-SORT.

```java
import java.lang.Math;
class MergeSort{
    public void Merge(int[] A, int p, int q, int r){
        int n1 = q - p + 1;
        int n2 = r - q;
        int[] L = new int[n1+1];
        int[] R = new int[n2+1];
        for(int i = 0; i < n1; i++) L[i] = A[p + i];
        for(int j = 0; j < n2; j++) R[j] = A[q + j + 1];
        L[n1] = Integer.MAX_VALUE; R[n2] = Integer.MAX_VALUE;
        int i = 0, j = 0;
        for(int k = p; k <= r; k++)
            if(L[i] <= R[j])
                A[k] = L[i++];
            else
                A[k] = R[j++];
    }
    public void run(int[] A, int p, int r){
        if(p < r){
            int q = (int) Math.floor((p+r) / 2.0);
            run(A, p, q); run(A, q+1, r); Merge(A, p, q, r);
        }
    }
}
```

Código do programa 2.1: Algoritmo de ordenação MERGE-SORT.

## 2.2 Taxa de Crescimento do Tempo de Execução

Estude o comportamento do algoritmo MERGE-SORT no que se refere aos tempos de execução realizando as seguintes tarefas:

1. determine um conjunto de 30 dimensões das tabelas a ordenar que resultem numa boa representação gráfica $C_N = \{n_1, n_2, \ldots, n_{30}\}$;
2. para cada dimensão da tabela meça $k = 50$ vezes o seu tempo de execução;
3. calcule a média individual da amostra de tempos de execução para cada valor $n \in C_N$;
4. calcule o desvio padrão correspondente;
5. represente com o programa `gnuplot` os tempos de execução e compare com o valor teórico esperado $O(n \log n)$ realizando os ajustes necessários dos fatores fixos;

6. compare os resultados com os esperados para o algoritmo Insertion-Sort;

7. compare os resultados com os esperados para o algoritmo Bubble-Sort.

A escolha das dimensões da tabela deve ser adequada ao computador e garantir tempos de execução suficientemente elevados.

# 3 Guia 3

O aluno deve criar uma pasta com a designação `guia3` no seu repositório de controlo de versões.

O aluno deve programar na linguagem de programação `Java` o algoritmo de ordenação `Heap-Sort`.

## 3.1 Programação do Algoritmo Heapsort

O algoritmo HEAP-SORT é outro exemplo do processo de projeto de algoritmos conhecido por *divisão e conquista*. Possui a vantagem de realizar a ordenação no espaço de memória da tabela a ordenar.

O pseudo-código correspondente está representado nas Figuras 3.1, 3.2, 3.3, 3.4 e 3.5.

Uma realização com a linguagem de programação `Java` está nas Figuras 3.1 e 3.2.

```
HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for i = A.length downto 2
3      exchange A[1] with A[i]
4      A.heap-size = A.heap-size − 1
5      MAX-HEAPIFY(A, 1)
```

Figura 3.1: Heapsort.

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

Figura 3.2: Heapsort.

Compile o código do programa e estude o seu funcionamento realizando as seguintes tarefas:

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4       largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7       largest = r
 8  if largest ≠ i
 9       exchange A[i] with A[largest]
10       MAX-HEAPIFY(A, largest)
```

Figura 3.3: Heapsort.

```
LEFT(i)
1  return 2i
```

Figura 3.4: Heapsort.

1. modifique o código de modo a ordenar uma tabela de 100 números reais gerada aleatoriamente;

2. crie uma versão que possa ordenar os objetos de uma classe designada por `Ponto` que representa pontos no espaço tridimensional e cujo termo para comparação seja a distância à origem;

3. crie uma versão que opere com listas;

4. crie uma versão que funcione independentemente do tipo da tabela;

5. experimente criar uma versão noutra linguagem de programação.

RIGHT($i$)
1 **return** $2i + 1$

Figura 3.5: Heapsort.

```
1  public class HeapSort {
2      int heapsize = 0;
3      public void Exchange(int[] A, int i, int j){
4          int temp = A[i];
5          A[i] = A[j];
6          A[j] = temp;
7      }
8      public int Left(int i){
9          return 2 * i;
10     }
11     public int Right(int i){
12         return 2 * i + 1;
13     }
14     public HeapSort(int[] A){
15         BuildMaxHeap(A);
16         for(int i=A.length-1; i >= 2; i--){
17             Exchange(A, 1, i);
18             heapsize = heapsize-1;
19             MaxHeapify(A,1);
20         }
21     }
22     public void BuildMaxHeap(int[] A){
23         heapsize = A.length-1;
24         for(int i = (int) Math.floor(A.length/2.0); i >= 1; i--) MaxHeapify(A, i);
25     }
26     public void MaxHeapify(int[] A, int i){
27         int l = Left(i);
28         int r = Right(i);
29         int largest = 0;
30         if(l <= heapsize && A[l] > A[i]) largest = l;
31         else largest = i;
32         if(r <= heapsize && A[r] > A[largest]) largest = r;
33         if(largest != i){
34             Exchange(A, i, largest);
35             MaxHeapify(A, largest);
36         }
37     }
38 }
```

Código do programa 3.1: Algoritmo de ordenação HEAP-SORT.

```java
public class MainHeapSort {
    private static int[] NewCopy(int[] A){
        int[] B = new int[A.length+1];
        for(int k=1; k < B.length; k++) B[k] = A[k-1];
        B[0] = Integer.MIN_VALUE;
        return B;
    }
    public static void PrintArray(int[] A){
        System.out.println();
        for(int i=1; i < A.length; i++) System.out.format("%3d", i );
        System.out.println();
        for(int i=1; i < A.length; i++) System.out.format("%3d", A[i]);
        System.out.println();
    }
    public static void main(String[] args) {
        int[] C = {4,1,3,2,16,9,10,14,8,7};
        int[] A = NewCopy(C);
        PrintArray(A);
        HeapSort hs = new HeapSort(A);
        PrintArray(A);
    }
}
```

Código do programa 3.2: Chamada do lgoritmo de ordenação HEAP-SORT.

## 3.2 Taxa de Crescimento do Tempo de Execução

Estude o comportamento do algoritmo HEAP-SORT no que se refere aos tempos de execução realizando as seguintes tarefas:

1. determine um conjunto de 30 dimensões das tabelas a ordenar que resultem numa boa representação gráfica;
2. meça 50 vezes para cada tamanho da tabela o seu tempo de execução;
3. calcule a média individual para cada tamanho;
4. calcule o desvio padrão individual;
5. represente com o programa `gnuplot` os tempos de execução e compare com o valor teórico esperado $O(n \log n)$;
6. compare com o algoritmo MERGE-SORT.