Algoritmos e Complexidade
Testes Práticos

LEI – 2010/2011

1 Teste 1 – Correcção

1. Considere o seguinte (extracto de) programa que calcula o factorial (da variável `x`)

```
f = 1;
while (x>0) {
    f = f*x; x=x-1;
}
```

   (a) Apresente uma especificação (pré e pós condições) que traduza o enunciado informal apresentado acima.

   (b) Apresente um invariante para a prova de correcção (parcial) do programa acima.

2. Considere o seguinte (extracto de) programa que calcula o factorial (da variável `x`)

```
f = 1; i=0;
while (i<x) {
    i=i+1;f = f*i;
}
```

   (a) Apresente uma especificação (pré e pós condições) que traduza o enunciado informal apresentado acima.

   (b) Apresente um invariante para a prova de correcção (parcial) do programa acima.

3. O algoritmo seguinte recebe como argumento o valor N e pretende calcular o valor 2^N.

```
I := 0; S := 1;
WHILE (I < N) DO
BEGIN I:=I+1; S:=S*2
END
```

(a) Escreva uma especificação de correcção total que seja satisfeita por este algoritmo. Note que isto implica que a pré-condição deve implicar que o ciclo termine.

(b) Explique a seguinte regra que utilizou nas aulas TP de AeC.

$$\frac{P \Rightarrow I \quad \{\, I \wedge c \,\}\, S \,\{\, I \,\} \quad (I \wedge \neg c) \Rightarrow Q}{\{\, P \,\}\ \texttt{while}\ c\ S\ \{\, Q \,\}} \text{ (while)}$$

4. O algoritmo seguinte recebe como argumentos os valores X e Y e pretende calcular o valor $RES = Y * X$.

```
RES := 0;
WHILE (Y>0) DO
BEGIN RES := RES + X;
      Y = Y-1
END
```

(a) Escreva uma especificação de correcção total que seja satisfeita por este algoritmo. Note que isto implica que a pré-condição deve implicar que o ciclo termine.

(b) Explique **as diferenças** entre as duas regras representadas em baixo.

$$\frac{\{\, I \wedge c \,\}\, S \,\{\, I \,\}}{\{\, I \,\}\ \texttt{while}\ c\ S\ \{\, I \wedge \neg c \,\}} \text{ (while-1)}$$

$$\frac{[\, I \wedge c \wedge E = n \,]\, S\, [\, I \wedge E < n \,] \quad (I \wedge \neg c) \Rightarrow E \geq 0}{[\, I \,]\ \texttt{while}\ c\ S\ [\, I \wedge \neg c \,]} \text{ (while-1)}$$

5. Considere o seguinte algoritmo que calcula o factorial de um número:

```
RES := 1;
WHILE (X>0) DO
 BEGIN
  RES := RES * X;
  X := X-1;
 END
```

(a) Escreva pré e pós-condições que exprimam a especificação para o fragmento de código apresentado.

(b) Apresente um *invariante* e um *variante* para o ciclo.

(c) Demonstre a preservação do invariante.

6. Considere o seguinte algoritmo que calcula o factorial de um número:

```
RES := 1;
Y := 0;
WHILE (Y<X) DO
 BEGIN
  Y := Y+1;
  RES := RES * Y;
END
```

(a) Escreva pré e pós-condições que exprimam a especificação para o fragmento de código apresentado.

(b) Apresente um *invariante* e um *variante* para o ciclo.

(c) Demonstre a preservação do invariante.

7. Considere a seguinte implementação da função `void reverse (int v[], int n)` que inverte um vector.

```
char *reverse (int v[], int n) {
int x;
if (n>1) { x = v[0]; v[0] = v[n-1]; v[n-1] = x;
           reverse (v+1,n-2);
         }
}
```

Faça a análise assimptótica do tempo de execução desta função (em função do tamanho do array argumento). Para justificar a sua resposta apresente uma relação de recorrência que defina o tempo de execução desta função.

8. Considere o seguinte algoritmo de multiplicação de dois números inteiros.

```
RES := 0;
WHILE (Y>0) DO
 BEGIN
  RES := RES + X;
  Y-1;
 END
```

(a) Escreva pré e pós-condições que exprimam a especificação para o fragmento de código apresentado.

(b) Apresente um *invariante* e um *variante* para o ciclo.

(c) Demonstre a preservação do invariante.

9. Considere o seguinte algoritmo de exponenciação de dois números inteiros.

```
RES := 1;
WHILE (Y>0) DO
 BEGIN
  RES := RES * X;
  Y-1;
 END
```

(a) Escreva pré e pós-condições que exprimam a especificação para o fragmento de código apresentado.

(b) Apresente um *invariante* e um *variante* para o ciclo.

(c) Demonstre a preservação do invariante.

2 Teste 2 – Complexidade

1. Considere a seguinte implementação da função `char * strcat (char *s1, char *s2)` que concatena a string `s2` a `s1`.

```
char *strcat (char s1[], char s2[]) {
int i=0, j=0;
while (s1[i] != '\0') i++;
while (s2[j] != '\0') { s1 [i] = s2 [j]; i++; j++;}
s1 [i] = '\0';
return s1;
}
```

Faça a análise assimptótica do tempo de execução desta função (em função dos tamanhos das **duas** strings argumento).

2. Considere a seguinte implementação da função `int strlen (char *)` que calcula o comprimento de uma string.

```
int strlen (char s[]) {
  int i=0
  while (s[i] != '\0') i++;
  return i;
}
```

Considere ainda a seguinte definição de uma função que calcula a mais longa de duas strings:

```
char *longest (char s1[], char s2[]) {
    if (strlen (s1) > strlen s2) return s1;
    else return s2;
}
```

Faça a análise assimptótica do tempo de execução desta função (em função dos tamanhos das **duas** strings argumento). Apresente uma definição alternativa cujo tempo de execução seja proporcional ao tamanho da menor string.

3. O algoritmo seguinte recebe como argumento o valor N e pretende calcular o valor 2^N.

```
I := 0; S := 1;
WHILE (I < N) DO
BEGIN I:=I+1; S:=S*2
END
```

Assumindo que L é o número de bits necessário para representar N e que:

- A multiplicação executa em tempo constante.
- A soma/subtracção executam em tempo constante.

(a) Escreva uma equação que descreva o tempo de execução $T(L)$ **em funcão do número de bits** L **necessários para representar** N.

(b) Indique a quais das seguintes classes podemos dizer que o algoritmo pertence: $\Omega(1), \Theta(1), O(1), \Omega(L), \Theta(L), O(L), \Omega(L^2), \Theta(L^2), O(L^2), \Omega(2^L), \Theta(2^L), O(2^L)$.

4. O algoritmo seguinte recebe como argumento o valor N e pretende calcular o valor 2^N.

```
I := SIZE(N)-1; S := 1;
WHILE (I >= 0) DO
BEGIN S:=S*S;
      IF BIT(I,N) THEN S=S*2; ENDIF
      I:= I-1;
END
```

Assumindo que L é o número de bits necessário para representar N e que:

- A função `SIZE(N)` retorna o número de bits do número N, e executa em tempo linear em L.
- A função `BIT(I,N)` retorna o valor do bit I do número N, e executa em tempo constante.

- A multiplicação executa em tempo constante.
- A soma/subtracção executam em tempo constante.

(a) Escreva uma equação que descreva o tempo de execução $T(L)$ **em funcão do número de bits L necessários para representar N.**

(b) Indique a quais das seguintes classes podemos dizer que o algoritmo pertence: $\Omega(1), \Theta(1), O(1), \Omega(L), \Theta(L), O(L), \Omega(L^2), \Theta(L^2), O(L^2), \Omega(2^L), \Theta(2^L), O(2^L)$.

5. Considere a seguinte implementação da função `mtmul` que multiplica duas matrizes triangulares:

```
void mtmul (float m1[][], float m2[][], float m3[][], int N) {
 int i, j, k;
 for (i=0; i<N; i++)
  for (j=0; j<=i; j++) {
   m3[i][j] = 0;
   for (k=0; k<=i; k++)
    m3[i][j] += m1[i][k] * m2[k][j];
   }
}
```

(a) Faça a análise assimptótica do tempo de execução no melhor e pior caso deste programa em função do tamanho das matrizes N (relembre que $\sum_{i=1}^{n} i = \frac{n*(n+1)}{2}$ e $\sum_{i=1}^{n} i^2 = \frac{n*(n+1)*(2n+1)}{6}$).

6. Considere a seguinte implementação da função `mtmul` que multiplica duas matrizes triangulares:

```
void mtmul (float m1[][], float m2[][], float m3[][], int N) {
 int i, j, k;
 for (i=0; i<N; i++)
  for (j=i; j<N; j++) {
   m3[i][j] = 0;
   for (k=i; k<N; k++)
    m3[i][j] += m1[i][k] * m2[k][j];
   }
}
```

(a) Faça a análise assimptótica do tempo de execução no melhor e pior caso deste programa em função do tamanho das matrizes N (relembre que $\sum_{i=1}^{n} i = \frac{n*(n+1)}{2}$ e $\sum_{i=1}^{n} i^2 = \frac{n*(n+1)*(2n+1)}{6}$).

7. Considere a seguinte implementação da função `void reverse (int v[], int n)` que inverte um vector.

```
void reverse (int v[], int n) {
int x;
if (n>1) { x = v[0]; v[0] = v[n-1]; v[n-1] = x;
           reverse (v+1,n-2);
         }
}
```

   (a) Escreva a equação de recorrência associada.

   (b) Faça a análise assimptótica do tempo de execução no melhor e pior caso desta função (em função do tamanho do array argumento).

8. Considere a seguinte implementação recursiva da função `void split (int pivot, int v[], int n)` que parte um vector em dois sub-vectores com base na comparação dos elementos com o `pivot`.

```
int split (int pivot, int v[], int n) {
  if (n==0) return 0;
  if (v[0]>pivot) {
    swap (v, 0, n-1);
    return split(pivot, v, n-1);
    }
  else return 1 + split(pivot, v+1, n-1);
}
```

   (a) Escreva a equação de recorrência associada (a função *swap* executa em tempo constante).

   (b) Faça a análise assimptótica do tempo de execução no melhor e no pior caso desta função (em função do tamanho do array argumento).

3 Teste 3 – Estruturas de Dados

1. Considere as seguintes definições para implementar uma tabela de hash com open addressing.

```
#define Hsize ...

typedef void *Info;
typedef char KeyType [9];
```

```
typedef struct pair {
    int deleted;
    int empty;
    KeyType key;
    Info    info;
} Cell;

typedef struct {
   int size; // tamanho do vector
   int used; // numero de celulas usadas
   Cell *table;
} THash;
```

Considere ainda que existe definida uma função de inserção `void addKey (THash *, KeyType, Info)`.

Defina uma função `reHash (THash *source, THash *dest);` que copia para `dest` as chaves de `source` (não apagadas), libertando o espaço ocupado por `source`. Assuma que `dest` já se encontra correctamente inicializado com a tabela vazia.

2. Considere as seguintes definições para implementar uma tabela de hash com chainning.

```
#define Hsize ...

typedef void *Info;
typedef char KeyType [9];

typedef struct pair {
    KeyType key;
    Info    info;
    struct pair *next;
} *Cell;

typedef struct {
   int size; // tamanho do vector
   Cell *table;
} THash;
```

Considere ainda que existe definida uma função de inserção `void addKey (THash *, KeyType, Info)`.

Defina uma função `reHash (THash *source, THash *dest);` que copia para `dest` as chaves de `source`, libertando o espaço ocupado por `source`. Assuma que `dest` já se encontra correctamente inicializado com a tabela vazia.

3. Represente graficamente a evolução de uma árvore AVL quando são introduzidos os seguintes números: 50, 40, 30, 90, 60, 70, 80, 100.

4. Represente graficamente a evolução de uma árvore AVL quando são introduzidos os seguintes números: 80, 70, 60, 100, 30, 40, 50, 90.

5. Considere os seguintes tipos de dados para duas representação alternativas para grafos dirigidos e pesados:

```
#define VMAX 10
typedef struct {
  int nvert;
  int mat[VMAX][VMAX];
} MAdj;
typedef struct nodo {
  int dest;
  int peso;
  struct nodo *prox;
} Nodo;
typedef struct {
  int nvert;
  Nodo *ladj[VMAX];
} LAdj;
```

   (a) Codifique a função `ladj2madj` que converte um grafo representado numa *listas de adjacência* para a representação numa *matriz de adjacência*.

   (b) Codifique a função `grauEnt` que calcula o grau de entrada de todos os vértices de um grafo representado numa *lista de adjacência*.

6. Considere os seguintes tipos de dados para duas representação alternativas para grafos dirigidos e pesados:

```
#define VMAX 10
#define AMAX 80
typedef struct nodo {
  int dest;
  int peso;
  struct nodo *prox;
} Nodo;
typedef struct {
  int nvert;
  Nodo *ladj[VMAX];
} LAdj;
```

```
typedef struct {
  int nvert;
  int vadj[VMAX];
  int narest;
  int dest[AMAX];
  int peso[AMAX];
} VAdj;
```

(a) Codifique a função `ladj2vadj` que converte um grafo representado numa *lista de adjacência* para a representação num *vector de adjacência*.

(b) Codifique a função `grauEnt` que calcula o grau de entrada de todos os vértices de um grafo representado com um *vector de adjacência*.