Algoritmos e Complexidade – 8 de Novembro de 2014 – Duração: 90 min

Parte A

1. Considere o seguinte programa parcialmente anotado:

```
// 0 <= a <= b && forall (a <= i <= b) v[i] = v_i
j=a; r=0;
while (j<=b) {
  r = r+v[j]; j=j+1;
}
// (forall (a <= i <= b) v[i] = v_i) && r = sum (a <= i <= b) v_i
```

   (a) Descreva informalmente (i.e., sucintamente e por palavras) a especificação deste programa.

   (b) Complete a anotação do programa (fornecendo o invariante do ciclo) e apresente as condições de verificação correspondentes à prova da **correcção parcial** do programa em causa.

2. Considere o problema de determinar, num array de inteiros o valor da máxima soma de elementos consecutivos. Por exemplo no array `[1,-5,2,-1,4,-3,1,-7,3]` esse valor corresponde a `5` (a soma `2-1+4`.

   Efectue a análise do comportamento assimptótico da função `A(N)`, que representa o número de acessos ao array `v` efectuados pela função `maxSoma` apresentada em baixo. Assuma para isso que a invocação da função `soma (v,a,b)` corresponde a fazer `(b-a+1)` acessos o array. No verso da folha encontram-se alguns somatórios que poderão ser úteis.

```
int soma (int v[], int a, int b){
  int r = 0, i;
  for (i=a; i<=b; i++) r=r+v[i];
  return r;
}
```

```
int maxSoma (int v[], int N){
  int i, j, r=0, m;
  for (i=0; i<N; i++)
    for (j=0; j<=i; j++){
      m = soma(v,j,i);
      if (m>r) r = m;
  }
  return r;
}
```

3. Considere a seguinte definição recursiva da função `maxSoma`:

```
int maxSomaR (int v[], int N){
    int r=0, m1, m2, i;
    if (N>0) {
      m1 = 0; m2=0;
      for (i=0;i<N;i++){
        m2 = m2+v[i];
        if (m2>m1) m1=m2;
      }
      m2 = maxSomaR (v+1,N-1);
      if (m1>m2) r = m1; else r = m2;
    }
    return r;
}
```

   De forma a analizar o comportamento desta função: (1) apresente uma relação de recorrência que traduza a complexidade desta função em termos do número de acessos ao array `v`; (2) apresente a correspondente árvore de recursão, e (3) apresente uma solução para a referida recorrência.

4. Na definição iterativa da função `maxSoma` apresentada na alínea 2, o ciclo mais interior faz sucessivas invocações da função `soma` que repetem muitos cálculos (e consequentemente acessos ao array). Apresente uma definição alternativa que evita as invocações da função `soma`.

Parte B

Considere a seguinte função que calcula o complemento para dois de um número inteiro armazenado num array de *booleanos*.

```
void complemento (char b[], int N){
  int i = N-1;
  while ((i>0) && !b[i])
    i--;
  i--;
  while (i>=0) {
    b[i] = !b[i]; i--;
  }
}
```

De forma a analizar a complexidade desta função em termos do número de elementos do array que são alterados (i.e., o número de iterações do segundo ciclo):

1. Identifique o pior caso, apresentando a complexidade (i.e, o número de elementos do array que são alterados) da função nesse caso.
2. Calcule a complexidade da função no caso médio. Considere para isso que o valor do input é perfeitamente aleatório, i.e., que a probabilidade de cada posição do array ser um `0` ou `1` é 0.5.

Formulário

- $\sum_{i=1}^{n} 1 = n$
- $\sum_{i=1}^{n} i = \frac{n\,(n+1)}{2}$
- $\sum_{i=1}^{n} i^2 = \frac{n\,(n+1)\,(2\,n+1)}{6}$
- $\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$
- $\sum_{i=1}^{n} i\,a^{i-1} = \frac{n\,a^{n+1}-(n+1)\,a^n+1}{(a-1)^2}$
- $\sum_{i=a}^{b} f(i) = \left(\sum_{i=1}^{b} f(i)\right) - \left(\sum_{i=1}^{a-1} f(i)\right)$
- $$\frac{P \Longrightarrow I \qquad I \wedge \neg c \Longrightarrow Q \qquad \{I \wedge c\} S \{I\}}{\{P\}\text{while } c\ S\{Q\}}$$