Algoritmos e Complexidade – 5 de Novembro de 2016 – Duração: 90 min

Parte A

1. Considere o seguinte excerto de uma função de partição para ser usada no algoritmo de *Quick-sort*.

   - Complete a anotação desta função apresentando um invariante `I`, que lhe permita provar (apenas) que o programa é correcto face à pré-condição e pós-condição anotadas no código.
   - Apresente as condições de verificação geradas a partir destas anotações.

```
// PRE: n > 0
i=0; j=n-2; k=0;
while (k<n-1) {
  // I
  if (a[k] <= a[n-1]) {
    b[i] = a[k]; i=i+1;
  } else {
    b[j] = a[k]; j=j-1;
  }
  k=k+1;
}
// POS: j == i-1
```

2. Considere a seguinte definição da função `maxSomaConseq` que calcula a máxima soma de elementos consecutivos de um array.

```
int maxSomaPref (int v[], int N) {
    int i, r, s;
    for (i=r=s=0; i<N; i++) {
       s+=v[i];
       if (s>r) r=s;
    }
    return r;
    }
}
```

```
int maxSomaConseq(int v[], int N) {
    int x,y;
    if (N==0) return 0;
    x = maxSomaPref (v,N)
    y = maxSomaConseq (v+1,N-1);
    if (x > y) return x;
    else return y;
}
```

   (a) Apresente e resolva uma recorrência que traduza o comportamento da função `maxSomaConseq` em termos do **número de acessos ao array**.

   (b) Uma forma de tornar esta função mais eficiente (evitando re-calcular muitas somas) consiste em usar um vector `somas` onde na posição `i` se encontra o resultado de `maxSomaPref (v+i,N-i)`.

   Note ainda que este array pode ser preenchido de uma forma eficiente sem usar a função `maxSomaConseq` (basta preenchê-lo do fim para o início).

   Usando as observações acima, apresente uma alternativa para a definição de `maxSomaConseq` que seja linear no tamanho do array.

Parte B

Considere o algoritmo típico de procura de um elemento numa *árvore binária de procura* nos seguintes cenários:

(i) a árvore é equilibrada (i.e. a sua altura é logarítmica no número de elementos da árvore), e *o elemento procurado ocorre na árvore*, com igual probabilidade em qualquer nó;

(ii) a árvore é equilibrada, e a probabilidade de o elemento procurado ocorrer na árvore é dada por p, e quando ocorre ocorre com igual probabilidade em qualquer nó.

(iii) a árvore é totalmente desequilibrada, degenerando numa lista, e *o elemento procurado ocorre na árvore*, com igual probabilidade em qualquer nó;

Para cada um dos cenários acima, efectue a *análise de caso médio* do tempo de execução do algoritmo, contabilizando o número de comparações efectuadas.

Formulário:

$$\begin{aligned}
\sum_{i=1}^{n} 1 &= n \\
\sum_{i=1}^{n} i &= \frac{n\,(n+1)}{2} \\
\sum_{i=1}^{n} i^2 &= \frac{n\,(n+1)\,(2\,n+1)}{6} \\
\sum_{i=0}^{n} a^i &= \frac{a^{n+1}-1}{a-1} \\
\sum_{i=1}^{n} i\,a^{i-1} &= \frac{n\,a^{n+1}-(n+1)\,a^n+1}{(a-1)^2}
\end{aligned}$$

$$\frac{\{P \wedge c\}\ S_1\ \{Q\} \quad \wedge \quad \{P \wedge \neg c\}\ S_2\ \{Q\}}{\{P\}\ \text{if } c\ S_1 \text{ else } S_2\ \{Q\}}$$

$$\frac{P \Rightarrow I \qquad I \wedge \neg c \Rightarrow Q \qquad \{I \wedge c\}\ S\ \{I\}}{\{P\}\ \text{while } c\ S\ \{Q\}}$$