Algoritmos e Complexidade – 31 de Outubro 2015 – Duração: 90 min

Parte A

1. Considere o seguinte excerto de uma função de *procura binária* num array ordenado de inteiros.

   - 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 e prove as condições de verificação geradas a partir destas anotações.

```
// PRE: n > 0
l = -1; r = n;
while (l+1 != r) {
  m = (l+r)/2;
  // I && m == (l+r)/2
  if (a[m] <= x) l = m;
  else r = m;
}
// POS: -1 <= l < n
```

2. Por vezes há mais do que uma dimensão a considerar no *input* de um algoritmo. Considere a seguinte função que, dado um array com `N` *strings*, todas de comprimento `M`, calcula o índice onde se encontra a que é alfabeticamente menor. O tempo de execução de `minInd` será dado por uma função $T(\texttt{N},\texttt{M})$.

```
int minInd (char *nomes [], int N) {
  int i, r = 0;
  for (i=1; i<N; i++)
    if (strcmp (nomes[i], nomes[r]) < 0)
      r = i;
  return r;
}
```

   Assumindo que `strcmp` executa no melhor caso em tempo constante (1) e no pior caso em tempo linear no tamanho das strings (M), analise assimptoticamente a função `minInd` no melhor e pior caso.

3. Considere a definição de uma versão do algoritmo *quicksort* que particiona o *array* de input em 3 secções que são depois recursivamente ordenadas. A função `part3` executa em *tempo linear*; utiliza como pivots o primeiro e o último elemento do *array*, que são colocados nas posições finais `q` e `s`.

```
void qSort3 (int A[], int p, int r) {
   if (p < r) {
      int q, s;
      part3(A, p, r, &q, &s);
      qSort3(A, p, q-1);
      qSort3(A, q+1, s-1);
      qSort3(A, s+1, r);
} }
```

   Sabendo que o melhor caso para o tempo de execução de `qSort3` ocorre quando as 3 secções criadas por `part3` têm aproximadamente o mesmo tamanho, e o pior caso quando a função de partição cria sempre duas secções vazias, escreva e resolva as recorrência correspondentes a cada um destes casos.

Parte B

Considere a seguinte função que compara os valores (inteiros e não negativos) de dois inteiros representados por arrays de `N` bits. Assume-se que esses bits estão armazenados do menos significativo (no índice `0`) ao mais significativo (no índice `N-1`).

```
int intcmp (int n1[], int n2[], int N) {
  int i;
  for (i=N-1; (i>=0) && (n1[i] == n2[i]); i--);
  if (i<0) return 0;
  else return (n1[i] - n2[i]);
}
```

- Analise a complexidade desta função no caso médio. Para isso determine o número médio de comparações (`n1[i] == n2[i]`) feitas, assumindo que os valores do entrada são dois arrays aleatórios (onde em cada posição existe com igual probabilidade o valor `0` ou `1`).
- A função `strcmp` referida na parte A tem um comportamento semelhante a esta função. A única diferença é que a probabilidade de dois caracteres serem iguais é de $\frac{1}{256}$. Diga por isso qual a complexidade média da função `minInd` apresentada acima.

Formulário:

$$\left|\begin{array}{ll} \sum_{i=1}^{n} 1 & = n \\ \sum_{i=1}^{n} i & = \frac{n\,(n+1)}{2} \\ \sum_{i=1}^{n} i^2 & = \frac{n\,(n+1)\,(2\,n+1)}{6} \\ \sum_{i=0}^{n} a^i & = \frac{a^{n+1}-1}{a-1} \\ \sum_{i=1}^{n} i\,a^{i-1} & = \frac{n\,a^{n+1}-(n+1)\,a^n+1}{(a-1)^2} \end{array}\right|$$

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

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