0 Problemas e Algoritmos

1. Considere o problema de contar num conjunto de números inteiros quantos pares de números somam zero.

   (a) Indique uma instância do problema.

   (b) Conceba um algoritmo para resolver o problema. Comece por exprimir o algoritmo em linguagem natural e depois em pseudo-código.

   (c) Explique porque o seu algoritmo resolve corretamente o problema.

   (d) Compare a sua solução com a apresentada abaixo em termos de eficiência.

   (e) Desenvolva um algoritmo mais eficiente que resolva o problema se os elementos do conjunto forem dados numa sequência ordenada de forma crescente.

   (f) Será que se consegue aproveitar a ideia anterior para obter um algoritmo mais eficiente para o problema original?

```
input: v vector de inteiros
output: número de pares de elementos de v
        que somam 0
Algorithm twoSum(v){
   n:=|v|
   cnt:=0
   for i:=0 to n-1 do
      for j:=i+1 to n-1 do
         if v[i]+v[j]=0
            cnt:=cnt + 1
   return cnt
}
```

2. Considere o problema de determinar uma aproximação da raiz quadrada de um número positivo.

   (a) Indique uma instância do problema.

   (b) Considerando que *epsilon* é uma constante entre 0 e 1, compare os três algoritmos apresentados mais à frente relativamente à correção.

   Pode recorrer ao seguinte resultado (corolário de um teorema provado por Newton sobre as raízes de um polinómio, ou seja, os valores para os quais o polinómio é zero):

   > Seja $p(x)$ um polinómio. Se x_n é uma aproximação de uma raiz de p então $x_{n+1} = x_n - \frac{p(x_n)}{p'(x_n)}$ é uma melhor aproximação dessa raiz de p.

Neste caso, para calcular $\sqrt{a}$ o polinómio que pretendemos aproximar a raíz é $p(x) = x^2 - a$. A sua derivada é $p'(x) = 2x$, o que resulta na expressão

$$x_{n+1} = x_n - \frac{x^2 - a}{2x}$$

(c) Experimentalmente compare a eficiência do *bissectionSquareRoot* e *NewtonMethod*. Pode focar a sua análise apenas no número de iterações que são feitas em cada caso.

```
input: a número positivo
output: aproximação da raiz quadrada de a
Algorithm exhaustiveSquareRoot(a){
   step:=epsilon * epsilon
   x:=0.0
   while |x*x - a| > epsilon and x<=a do
      x:=x+step
   return x
}
```

```
input: a número positivo
output: aproximação da raiz quadrada de a
Algorithm bissectionSquareRoot(a){
   low:=0.0
   high:=max(1.0,a)
   x:=(high+low)/2
   while |x*x - a| > epsilon do
     if x*x < a
        low:=x
     else
        high:=x
     x:=(high+low)/2
   return x
}
```

```
input: a número positivo
output: aproximação da raiz quadrada de a
Algorithm NewtonMethod(a){
   x:=a
   while |x*x - a| > epsilon do
      x:=x - ((x*x - a)/(2*x)))
   return x
}
```

(d) Considere a implementação em Java de um outro algoritmo também baseado no Método de Newton. Formule com rigor as pós-condições dos dois algoritmos. Compare-os, por exemplo, no caso em que $a = 0.25$ e $a = 2500$.

```
//@requires a>0
public static double sqrt(double a){
   double x = a;
   while (Math.abs(x - a/x) > EPSILON) {
      x -= (x*x - a) / (2.0*x);
   return x;
}
```

3. Considere o problema de gerar uma permutação aleatória dos números entre 1 e n.

(a) Indique uma instância do problema.

(b) Conceba um algoritmo para o problema. Comece por exprimir o algoritmo em linguagem natural e depois em pseudo-código.

(c) Formule e análise a correção do seu algoritmo.

(d) Indique como poderia, experimentalmente, analisar a correção do seu algoritmo.

4. Considere o algoritmo descrito abaixo para gerar uma permutação aleatória dos números entre 1 e n , proposto por Fisher & Yates em 1963:

   1 Escreva os números de 1 a n numa sequência.

   2 Escolha aleatoriamente um número k entre um e o número de números que ainda permanecem por riscar (inclusivé).

   3 Começando a contar a partir do lado esquerdo, risque o k-ésimo número ainda não riscado, e escreva-o noutro sítio.

   4 Repita a partir do passo 2 até todos os números estarem riscados.

   5 A sequência de números escritos no passo 3 é uma permutação dos números originais.

   (a) Exprima o algoritmo em pseudo-código.

   (b) Analise a correcção deste algoritmo.

5. Considere o algoritmo descrito abaixo popularizado por D.Knuth:

```
input: v vector de tamanho maior ou igual a 1
output: v baralhado de forma aleatória
Algorithm shuffle(v){
   n = |v|
   for i:=n-1 downto 1 do
      j:=random integer with 0 <= j <= i
      exchange v[j] and v[i]
   return v
}
```

   (a) Implemente este algoritmo em Java.

   (b) Indique como poderia usar este algoritmo para resolver o problema anterior.

   (c) Compare essa solução com a anterior.

   (d) Compare essa solução com a seguinte:

```
input: n>=1
output: vector com permutação aleatória
        dos números entre 1 e n
Algorithm perm(n){
   create array v of length n
   for i:=0 to n-1 do
      j:=random integer with 0 <= j <= i
      if j != i
        v[i]:=v[j]
      v[j]:=i+1
   return v
}
```

6. Considere o problema de selecionar um elemento aleatório de uma sequência que é lida por exemplo do *standard input* (e portanto tem um tamanho desconhecido inicialmente). Conceba um algoritmo que resolva o problema e que gaste sempre a mesma quantidade de espaço de memória.