Análise de Algoritmos

1. Uma pequeno sapo quer chegar ao outro lado da estrada. O sapo está atualmente localizado na posição X e quer chegar a uma posição maior ou igual a Y. O pequeno sapo sempre salta uma distância fixa, D. Contar o número mínimo de saltos que o sapo pequeno deve executar para atingir seu alvo.

   Escreva uma função: int solução (int X, int Y, int D); que, dado três inteiros X, Y e D, retorna o número mínimo de saltos da posição X para uma posição igual ou maior do que Y.

   Por exemplo, dado: X = 10, Y = 85 e D = 30, a função deve retornar 3, porque o sapo será posicionado da seguinte maneira:

   Após o primeiro salto, na posição 10 + 30 = 40; após o segundo salto, na posição 10 + 30 + 30 = 70; após o terceiro salto, na posição 10 + 30 + 30 + 30 = 100

   a) Desenvolva um algoritmo com complexidade de tempo esperada $\mathcal{O}(E)$, onde E é o número pulos dados pelo sapo.

Algorithm 1 sapo(X, Y, D)

$steps \leftarrow 0$
while $X < Yx$ **do**
  $steps \leftarrow steps + 1$
  $X \leftarrow X + D$
end while
return steps

   b) Desenvolva um algoritmo com complexidade de tempo esperada $\mathcal{O}(1)$.

Algorithm 2 sapo(X, Y, D)

if (Y-X) **mod** D == 0 **then**
  return (Y-X)/D
else
  return (Y-X)/D + 1
end if

2. Dado um vetor A consistindo de N inteiros distintos. O vetor contém inteiros no intervalo $[1 \ldots (N+1)]$, o que significa que falta exatamente um elemento.

   Seu objetivo é encontrar esse elemento que falta.

   Por exemplo, dado vetor A tal que:

   - $A[0] = 2$
   - $A[1] = 3$
   - $A[2] = 1$
   - $A[3] = 5$

   O seu algoritmo deve retornar 4, pois é o elemento ausente.

   Escreva um algoritmo para as seguintes suposições:

   - N é um número inteiro dentro do intervalo [0..100.000];
   - Os elementos de A são todos distintos;
   - Cada elemento do vetor A é um inteiro dentro do intervalo [1..(N + 1)].

a) Desenvolva um algoritmo com complexidade de tempo esperada $\mathcal{O}(N)$ e complexidade de espaço extra esperada $\mathcal{O}(N)$.

Algorithm 3 PermMissingElement(A,N)

```
Seja um vetor B[0 ... N+1]          ▷ Quantidade linear de memória extra utilizada
for i ← 1 to N+1 do                  ▷ Inicialize o vetor B com zero
    B[i] ← 0
end for
for i ← 0 to N-1 do                  ▷ Marque com 1 os elementos presentes no vetor
    B[A[i]] ← 1
end for
for i ← 1 to N+1 do
    if B[i] == 0 then
        return i
    end if
end for
```

b) Desenvolva um algoritmo com complexidade de tempo esperada $\mathcal{O}(N)$ e complexidade de espaço extra esperada $\mathcal{O}(1)$.

Algorithm 4 PermMissingElement(A,N)

```
soma ← 0
for i ← 1 to N+1 do                  ▷ Some todos os números entre 1 e N+1
    soma ← soma + i
end for
for i ← 0 to N-1 do                  ▷ Subtraia dos elementos presentes no vetor A
    soma ← soma − A[i]
end for
return soma
```

3. Dado um vetor não vazia A consistindo de N inteiros. O vetor A representa os números em uma fita.

   Para um inteiro P, tal que $0 < P < N$, divide esta fita em duas partes não vazias: $A[0], A[1], ..., A[P-1]$ e $A[P], A[P+1], ..., A[N-1]$.

   A diferença entre as duas partes é o valor de:

$$|(A[0] + A[1] + ... + A[P-1]) - (A[P] + A[P+1] + .. + A[N-1])|$$

   Em outras palavras, é a diferença absoluta entre a soma da primeira parte e a soma da segunda parte.

   Por exemplo, considere o vetor A tal que:

   - $A[0] = 3$
   - $A[1] = 1$
   - $A[2] = 2$
   - $A[3] = 4$
   - $A[4] = 3$

   Podemos dividir esta fita em quatro lugares:

   - P = 1, diferença $= |3 - 10| = 7$
   - P = 2, diferença $= |4 - 9| = 5$

- P = 3, diferença = $|6 - 7| = 1$
- P = 4, diferença = $|10 - 3| = 7$

Escreva um algoritmo que, dada um vetor não vazia A de N inteiros, retorna a diferença mínima que pode ser alcançada.

No exemplo anterior, o algoritmo deve retornar 1, conforme explicado acima.

Escreva um algoritmo eficiente para as seguintes suposições:

- N é um número inteiro dentro do intervalo $[2 \ldots 100.000]$;
- Cada elemento do vetor A é um inteiro dentro do intervalo $[-1.000 \ldots 1.000]$.

a) Desenvolva um algoritmo com complexidade de tempo esperada $\mathcal{O}(N^2)$

Algorithm 5 Equilibrium(A,N)

```
min_dif ← 100.000.000                         ▷ maior soma possível
for P ← 1 to N − 1 do
    esq ← 0
    dir ← 0
    for i ← 0 to N − 1 do
        if i < P then
            esq ← esq + A[i]
        else
            dir ← dir + A[i]
        end if
    end for
    dif ← |esq − dir|
    min_dif ← min(min_dif, dif)
end for
return min_dif
```

b) Desenvolva um algoritmo com complexidade de tempo esperada $\mathcal{O}(N)$

Algorithm 6 Equilibrium(A,N)

```
min_dif ← 100.000.000                         ▷ maior soma possível
esq ← 0
dir ← 0
for i ← 0 to N − 1 do
    dir ← dir + A[i]
end for
for i ← 0 to N − 2 do
    esq ← esq + A[i]
    dir ← dir − A[i]
    dif ← |esq − dir|
    min_dif ← min(min_dif, dif)
end for
return min_dif
```