Algoritmos e Complexidade — Universidade do Minho
Teste 3 de Novembro de 2022

Número: _____ Nome: _____

1. Especificação (5 valores)

Considere a seguinte função que determina o primeiro índice onde dois arrays diferem. Se os arrays forem iguais, a função devolve `N` (o tamanho dos arrays). Apresente uma especificação para este problema.

(Consider the definition of a function which computes the lowest position where two arrays differ; when the arrays are equal it returns the length of the arrays. Provide a specification (pre and post conditions) of this function).

```
int difInd (int a[], int b[], int N) {
     // Pre: ?
     int i=0;
     i = 0;
     while (i<N && a[i] == b[i])
          i=i+1;
     // Pos: ?
     return i;
}
```

2. Correção (5 valores)

Indique qual o invariante e variante que permitem verificar a correção total da seguinte função que, dado um array com os pesos de `N` produtos que se pretende comprar num supermercado, e a capacidade `C` dos sacos desse supermercado, determina o número de sacos necessários para transportar todos os produtos. Por exemplo, se os pesos dos produtos forem `{3,6,2,1,5,7}` e `C == 10` a função devolve `3`.

(Write down a loop invariant and a variant that can be used to prove the correctness and termination of the function `sacos` that, given an array with the weights of `N` products and the

capacity `C` of a bag, computes the number of bags necessary to carry all `N` products. For instance, if the weights are `{3,6,2,1,5,7}` and `C == 10` the function should return `3`.)

```
int sacos(int p[], int N, int C) {
     // Pre: N > 0 && C > 0 && forall_{0 <= k < N} 0 < p[k] <= C
     int r = 1, t = p[0], i = 1, j = 0;
     while (i < N) {
          // Inv: ?
          // Var: ?
          if (t + p[i] > C) {
               j = i; t = 0; r = r + 1;
          }
          t = t + p[i]; i = i + 1;
     }
     return r;
     // Pos: 0 < r <= N && r*C >= sum_{0 <= k < N} p[k]
}
```

3. Complexidade algoritmos iterativos (5 valores)

Analise a complexidade da função `sacos` (conte o número de operações aritméticas), identificando o pior e melhor caso.

(Calculate the number of arithmetic operations performed by the above function `sacos`, identifying the worst and best cases)

4. Complexidade algoritmos recursivos (5 valores)

Considere agora a seguinte função que calcula o maior número de produtos que se pode comprar no supermercado com um único saco de capacidade C. Por exemplo, se os pesos dos produtos forem `{3,6,2,1,5,7}` e `C == 12` a função devolve 4, correspondente a comprar o 1º, 3º, 4º e 5º produtos.

(The function `ensaca` receives an array with the weights of each product. This function computes the maximum number of products that can be carried with a single bag of capacity C.

For instance, if the weights of the products are given by `{3,6,2,1,5,7}` and `C == 12` the function should return `4`, corresponding to the1st, 3rd, 4th, and 5th products.)

```
int ensaca(int p[], int N, int C) {
     int r,t;
     if (N == 0) return 0;
     r = ensaca(p+1,N-1,C);
     if (p[0] <= C) {
          t = 1 + ensaca(p+1,N-1,C-p[0]);
          if (t > r) r = t;
     }
     return r;
}
```

Analise a complexidade da função `ensaca` (conte o número de acessos ao array), identificando o pior e melhor caso.

(Write down and solve a recurrence relation that characterizes the number of array accesses made by this function, identifying the worst and best cases)