Intermediate test of Algorithms 2023-2024

DCC-FCUP, University of Porto
José Proença

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

10th November 2023 – duration: 1h

Number: _____ Name: _____

Recall:

$f(n) = \mathcal{O}(g(n))$ if there exist $n0, c > 0$ such that $\forall n \geq n0 : f(n) \leq c \times g(n)$
$f(n) = \Omega(g(n))$ if there exist $n0, c > 0$ such that $\forall n \geq n0 : f(n) \geq c \times g(n)$
$f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
$\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$
$\sum_{i=1}^{n} i \times x^{i-1} = \frac{n \times x^{n+1} - (n+1) \times x^n + 1}{(x-1)^2}$

Specification (5 values)

Exercise 1. 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 formal specification** (pre and post conditions) of this function).

Considere a definição abaixo de uma função que calcula a menor posição onde 2 arrays diferem; quando os arrays são iguais devolve o comprimento de ambos os arrays. ***Escreva uma especificação formal*** *(pré- e pós-condições) desta função.*

```
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;
}
```

Intermediate test of Algorithms 2023-2024 (continuation)

DCC-FCUP, University of Porto, José Proença, 10th November 2023 – duration: 1h

Number: _____ Name: _____

Correction (8 values)

Exercise 2. Write a **loop invariant** and a **variant**, and **use them to prove** the correctness and termination of the function ***bags*** that computes the number of bags needed when paying at a supermarket. More concretely, given an array 'w' with the weights of a sequence of 'N' products purchased and the capacity 'C' of each bag, the function computes the number of bags needed to carry all 'N' products. For instance, when the weights are `[3,6,2,1,5,7]` and $C = 10$ the function returns 3.

*Escreva um **invariante de ciclo** e um **variante**, e **utilize-os para provar** correção e terminação da função **bags** que calcula o maior número de sacos necessários numas compras de supermercado. Mais concretamente, dado um array 'w' com os pesos de uma sequência de 'N' produtos comprados, e a capacidade 'C' de cada um dos sacos, a função calcula o número de sacos que são necessários para transportar todos os 'N' produtos. Por exemplo, quando os pesos são `[3,6,2,1,5,7]` e $C = 10$ a função retorna 3.*

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

Hint: *The invariant will be larger than most seen in the lectures. Consider extending it with information over* $w[k] \leq C$*, and with other facts such as* $r \leq i$ *and* $t \geq 0$*.*

Ajuda: *O invariante será maior que a maioria vista nas aulas. Considere extendê-lo com informação relativa a* $w[k] \leq C$*, e com outros fatos tais como* $r \leq i$ *e* $t \geq 0$*.*

NOTE: `r*C-t` in the post condition is WRONG. I mentioned this in the test, 45min after the start, and asked to consider `r*C` instead (and gave 1h30 instead of 1h). It should be `r*C` to be correct, or `r*C-C+t` to help (and could be equal to the sum). Invariant has 6 parts:

1. `0<r<=N`
2. `r*C-C+t>=sum_0<=k<iw[k]`
3. `r<=i`
4. `forall_0 <= k < N 0<w[k]<=C` (constant)
5. `i<=N`
6. `t<=C`

Intermediate test of Algorithms 2023-2024 (continuation)

DCC-FCUP, University of Porto, José Proença, 10th November 2023 – duration: 1h

Number: _____ Name: _____

Complexity (7 values)

Exercise 3. Recall the previous ***bags*** function. Analyse the complexity of this function with respect to the number of additions. This should include (1) the identification of the **best and worst cases**, (2) the **total number of additions** in both cases, and (3) the corresponding **asymptotic complexity**.

[*Recorde a função anterior **bags**. Analise a complexidade desta função relativamente ao número de adições. Isto deverá incluir (1) a identificação do **melhor e pior caso**, (2) o **número total de adições** em ambos os casos, e (3) a **complexidade assimptótica** correspondente.*]

Exercise 4. Is $\mathcal{O}(n^3) = \mathcal{O}(3^n)$? Justify your answer.

[*É verdade que $\mathcal{O}(n^3) = \mathcal{O}(3^n)$? Justifique a sua resposta.*]