# Final test of Algorithms 2023-2024

DCC-FCUP, University of Porto
José Proença

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

**15th December 2023 – duration: 2h**

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 $f(n) = \Omega(g(n))$

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

## Recursive functions (5 values)

**Exercise 1.** The function `putInBag` receives 3 inputs: (1) an array `p` with a sequence of weights of products that will be put in bags, (2) the number `N` of products, and (3) the capacity `C` of each bag (the maximum weight supported). 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 `{3,6,2,1,5,7,20}` and `C==12` the function should return 4, corresponding to the 1st, 3rd, 4th, and 5th products.

**1.1. Identify** the worst and best cases with respect to the number of array accesses.

**1.2. Write down and solve a recurrence relation** for each case that characterizes the number of **array accesses** made by this function.

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

*A função `putInBag` recebe 3 valores: (1) um array `p` com uma sequência de pesos de produtos que serão postos em sacos, (2) o número `N` de produtos, e (3) a capacidade `C` de cada saco (o peso máximo suportado). Esta função calcula o número máximo de produtos que podem ser colocados num único saco de capacidade `C`. Por exemplo, se os pesos dos produtos forem `{3,6,2,1,5,7,20}` e `C==12` a função devolve 4, correspondente a comprar o 1º, 3º, 4o e 5º produtos.*
*(1) Identifique o melhor e pior caso relativamente ao número de acessos ao array.*
*(2) Escreva e resolva a relação de recorrência para cada um dos casos que caracterize o número de acessos ao array feito por esta função.*

# Final test of Algorithms 2023-2024 (continuation)

**DCC-FCUP, University of Porto, José Proença, 15th December 2023 – duration: 2h**

Number: _____ Name: _____

## Average-case analysis (5 values)

**Exercise 2.** Consider the function **f** below, where a is an array with $0$ and $1$ values and N is the size of a. Assume that the function g(a,N) runs in $\Theta(2^N)$.

**2.1. Identify** the best and worst cases.

**2.2. Compute the average time** taken by **f**, and explain the assumptions that you make.

```
int f(int a[], int N) {
  int b = 0;
  for (int i = 0; i < N; i++)
    if (a[i] == 1) b++;
  if (b == 1) return g(a,N);
  else return 0;
}
```

*Considere a função **f** acima, onde a é um array com valores $0$ e $1$ e N é o tamanho de a. Assuma que a função g(a,N) corre em $\Theta(2^N)$.*
*(1) Identifique o melhor e pior casos.*
*(2) Faça a análise de caso médio de **f** e explique as assumções que fizer.*

# Final test of Algorithms 2023-2024 (continuation)

**DCC-FCUP, University of Porto, José Proença, 15th December 2023 – duration: 2h**

Number: _____ Name: _____

## Amortised analysis (5 values)

**Exercise 3.** Consider an implementation of a Hashtable using open addressing and dynamic arrays. Consider also that, when the load factor is not more than 50%, insertions and lookups of execute in constant time (1).

When the load factor reaches 50% the size of the array is doubled, and this duplication has a cost equal to the size of the array (before duplication). **Show that the amortised cost** of the insertion operator is still constant.

*Considere que se implementa uma tabela de Hash com tratamento de colisões por open addressing e usando arrays dinâmicos. Considere ainda que as inserções e consultas de uma chave executam em tempo constante (1), desde que não haja realocação do array. Esta assumpção só é válida quando o factor de carga da tabela não é superior a 50%. Por isso, quando esse factor atinge esse valor o array é duplicado e essa duplicação tem um custo adicional igual ao tamanho do array (antes da duplicação).* ***Mostre que o custo amortizado*** *da inserção é constante.*

# Final test of Algorithms 2023-2024 (continuation)

**DCC-FCUP, University of Porto, José Proença, 15th December 2023 – duration: 2h**

Number: _____ Name: _____

## Data Structures (5 values)

**Exercise 4.** Recall the definition of **min-heap** implemented as an array h, where given a node at index $n$, the indices of its parent, left, and right node are $(n-1)/2$, $2n+1$, and $2n+2$, respectively.

```
void bubbleUp(i,h,N) {
  // 1. implement "bubbleUp"
}

void bubbleDown(i,h,N) {
  // You can use "bubbleDown" without defining it.
}

// helper function that swaps a[i] by a[j]
void swap(int i, int j, int a[]) {
  int tmp = h[i];
  h[i]=h[j]; h[j]=tmp;
}

int decrease(x,y,h[],N) {
  // 2. implement "decrease"
}
```

**4.1. Implement the function** bubbleUp, described in the lectures. Recall that this function, given a tree that obeys the min-heap invariant except possibly between a given node n and its parent, performs successive swaps until the min-heap invariant is restored.

**4.2. Implement a function** `int decrease(int x, int y, int h[], int N)` that, given a min-heap h with N elements and a value x, substitutes a **single occurrence of x by a strictly smaller value y**. You can use the functions bubbleDown, bubbleUp, and swap, and do not need to aim at best possible performance. This function returns 1 if the value x is found, and 0 otherwise.

**4.3. Identify** the **best** and **worst** case of the decrease function and **compute** its complexity. If needed, you can assume that the complexity of bubbleDown, regarding the size of the subtree, is constant in the best case and logarithmic in the worst case.

*Relembre a definição de min-heap implementada como um array h, onde dado um nodo num índice n, os índices do seu pai, filho esquerdo, e filho direito são dados por $(n-1)/2$, $2n+1$, e $2n+2$, respetivamente.*
*(1) **Implemente uma função** bubbleUp, descrita nas aulas. Recorde que esta função, dada uma árvore que obedece ao invariante das min-heap com a possível exceção num dado nodo n relativamente ao seu pai, aplica trocas sucessivas até restaurar o invariante das min-heap.*
*(2) **Implemente uma função** `int decrease(int x, int y, int h[], int N)` que, dada uma min-heap h com N valores e um valor x, substitui uma única ocorrência de x por um valor y estritamente menor. Pode usar as funções bubbleUp, bubbleDown, e swap, e não precisa de produzir uma função com máxima performance. O valor retornado é 1 quando o valor é encontrado, e 0 caso contrário.*
*(3) **Identifique** o melhor e pior caso da função decrease e **calcule** a sua complexidade.*