Programação II

Exercícios 2
Complexidade assintótica

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Tecnologias da Informação

2020/2021

1. Ordene as seguintes funções por taxa de crescimento assintótico: $4n \log n$, 2^{10}, $2^{\log_{10} n}$, $3n + 100 \log n$, 4^n, $n^2 + 10n$.

2. Num dado computador, uma operação demora um milisegundo a ser executada. Considere um programa que realiza um número de operações dado pela função $f(n) = n^5$, para um qualquer parâmetro n. Qual o maior valor de n para o qual o programa é capaz de terminar os seus cálculos se o tempo disponível for (a) um ano; (b) uma década; (c) 15 mil milhões de anos (a idade do universo)? Nota: um ano tem aproximadamente 3.15×10^7 segundos.

3. Repita o exercício anterior para as seguintes funções: $f(n) = 10n$, $f(n) = 1000n$, $f(n) = n^2$ e $f(n) = 2^n$.

4. Supponha que o tempo de execução de um algoritmo em inputs de tamanho 1000, 2000, 3000 e 4000 é de 5 segundos, 20 segundos, 45 segundos, and 80 segundos, respetivamente. Estime quanto tempo levará para resolver um problema de tamanho 5000. A taxa de crescimento assintótico do algoritmo é linear, log-linear, quadrática, cúbica ou exponencial?

5. Apresente uma caracterização $\mathcal{O}$ do tempo de execução de cada uma das funções abaixo, em termos do input `n`.

(a)
```
def a (n):
    m = 0
    for i in range (1, 10 * n):
        for j in range (1, n):
            m += 1
    return m
```

(b)
```
def b (n):
    x = 0
    for a in range (0, 2021):
        x += a * n
    return x
```

(c)
```
def c (n):
    b = n * n
    while b > n:
        if b % 2 == 0:
            b -= 1
        else:
            b -= 2
    return b
```

(d)
```
def d (n, v):
    soma = 0
    for i in range(0, n):
        for j in range (1, 4):
            soma += v[i]
    return soma
```

(e)
```
def e (n):
    x = n * n * n
    while x > 1:
        x /= 2
    return x
```

(f)
```
def f (n):
    soma = n * n
    while soma % 2 == 0:
        soma -= 1
    return soma
```

(g)
```
def g (n):
    c = 0
    for i in range (0, n):
        for j in range (i, n):
            c += 1
    return c
```

(h) analise a complexidade em termos dos tamanhos das listas `l1` e `l2`.

```
def h (l1, l2):
    soma = 0
    for x in l1:
        if x % 2 == 0:
            soma += 1
        else:
            for y in l2:
                soma += y
    return soma
```

6. Analise a complexidade assintótica da função `diferenca`.

```
def diferenca (l1, l2):
    """Devolve uma lista que contém os elementos de
    l1 que não estão em l2

    Args:
        l1 (list): lista
        l2 (list): lista

    Returns:
        list: lista l1 \ l2
    """
    resultado = []
    for x in l1:
        if x not in l2:
            resultado.append(x)
    return resultado
```

7. Analise a complexidade assintótica da função `unicos`.

```
def unicos(lista) :
    """Recebe uma lista ordenada e devolve uma outra
    lista contendo exatamente uma ocorrência de cada
    elemento da lista original

    Args:
        lista (list): lista original de valores

    Returns:
        list: lista resultado

    Requires: a lista original está ordenada
    Ensures: a lista resultado está ordenada
    """
    return [] if not lista else (
        [lista[i] for i in range(len(lista)-1)
                  if lista[i] != lista[i+1]]
        + [lista[-1]])
```

8. Analise a complexidade assintótica da função `inverter`.

```
def inverter(lista):
    """Inverte a ordem dos elementos de uma lista

    Args:
        lista (list): lista original

    Ensures: a lista é alterada, invertendo a ordem
    dos seus elementos
    """
    for i in range(len(lista)//2):
        lista[i], lista[-1-i] = lista[-1-i], lista[i]
```

9. A função `minimo` devolve o valor mínimo de uma lista.

```
def minimo (lista):
    copia = list(lista)
    copia.sort()
    return copia[0]
```

   (a) Analise a complexidade assintótica da solução dada.

   (b) Proponha uma solução linear.

10. As funções `sem_repetidos1` e `sem_repetidos2` verificam se uma lista *não* tem repetidos.

```
def sem_repetidos1 (l):
    for i in range (len(l)):
        if l[i] in l[(i+1):]:
            return False
    return True

def sem_repetidos2 (l):
    copia = list(l)
    copia.sort()
    for i in range (len(l) - 1):
        if copia[i] == copia[i+1]:
            return False
    return True
```

    (a) Analise a complexidade assintótica de cada uma das soluções.

    (b) Proponha uma solução linear. Sugestão: converta a lista num conjunto.

11. A seguinte função calcula as médias dos prefixos de uma lista.

```
def media_prefixos (l):
    """
    Requires: uma lista de números
    Ensures: devolve uma lista m onde m[i] é a média
    dos elementos l[0] ,... , l[i-1]
    """
    m = []
    for i in range(len(l)):
        soma = 0.0
        for j in range(i + 1):
            soma += l[j]
        m.append(soma/(i + 1))
    return m
```

    (a) Verifique que a função tem um tempo de execução quadrático.

    (b) Apresente uma solução com tempo linear. Sugestão: calcule incrementalmente a média de cada um dos prefixos através da seguinte fórmula.

$$\begin{cases} m[0] = l[0] \\ m[k+1] = (m[k] \cdot (k+1) + l[k+1])/(k+2) \end{cases}$$

12. A sequência de Fibonacci descreve o crescimento de uma população de coelhos em condições ideais. Os primeiros números da sequência são 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377. O n-ésimo termo da sucessão é dado pela seguinte fórmula de recorrência:

$$f(n) = \begin{cases} 1 & \text{se } n = 1 \text{ ou } n = 2 \\ f(n-1) + f(n-2) & \text{caso contrário} \end{cases}$$

(a) Converta a fórmula de recorrência numa função Python recursiva. Analise a complexidade da solução.

(b) O n-ésimo termo da sucessão de Fibonacci também pode ser dado pela seguinte função.

```
def fib(n):
  if n <= 1:
    return 1
  a = 1
  b = 1
  for i in range(1, n):
    resultado = a + b
    a = b
    b = resultado
  return resultado
```

Analise a complexidade desta solução e compare com a função da alínea anterior.