Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
**Algoritmos e Estruturas de Dados**
Semestre de Verão 2021/22
Primeira série de problemas

---

**Observações:**

- Data de entrega: **11 de Abril de 2022**.
- No contexto desta série, o triplo *(v, l, r)* representa o *subarray* do *array* $v$, compreendido entre os índices $l$ e $r$ inclusive.
- Valorizam-se soluções mais eficientes.

# 1 Algoritmos Elementares

1. Realize a função

   ```
   fun upperBound(a: IntArray, l: Int, r: Int, element: Int): Int
   ```

   que dado um *sub-array (a,l,r)* ordenado de modo crescente, retorna o último índice `i` tal que `array[i]<=element`. No caso do *sub-array* ser um *sub-array* vazio retorna -1. Note que se todos os números no *sub-array* forem maiores do que `element` retorna-se -1.

2. Realize a função

   ```
   fun countIncreasingSubArrays(v: IntArray): Int
   ```

   que, dado um *array* `v` de inteiros, retorna o número de *sub-arrays* estritamente crescentes presentes em `v` com dimensão maior ou igual a 2. Por exemplo, no caso de $v = \{1, 2, 4, 4, 5\}$ este método deverá retornar 4, visto que existem os seguintes sub-arrays estritamente crescentes: $\{1, 2\}$,$\{1, 2, 4\}$, $\{2, 4\}$ e $\{4, 5\}$.

3. Realize a função

   ```
   fun countEquals(points1: Array<Point>, points2: Array<Point>,
                   cmp:(p1:Point, p2:Point)-> Int): Int
   ```

   que dados dois *arrays* `points1` e `points2`, ordenados de modo estritamente crescente segundo a função de comparação `cmp`, retorna o número de pontos que ocorrem simultaneamente em ambos os *arrays*. No caso de não existirem elementos em algum dos *arrays*, o método retorna 0. O tipo `Point` está definido da seguinte forma: `data class Point(var x:Int, var y:Int)`

4. Realize a função

   ```
   fun mostLonely(a: IntArray): Int
   ```

   que dado o *array* `a` de inteiros positivos, retorna o inteiro que se encontra mais **isolado**. O inteiro mais isolado num *array* é o inteiro que tem a maior distância `k`, sendo `k`, a menor das diferenças, em valor absoluto, entre ele e cada um dos restantes. Por exemplo, considere `a`= $\{10, 16, 1, 17, 5\}$. O elemento mais isolado neste caso é o elemento 10, porque $k_1 = |1 - 5| = 4$, $k_5 = |5 - 1| = 4$; $k_{10} = |10 - 5| = 5$, $k_{16} = |16 - 17| = 1$ e $k_{17} = |17 - 16| = 1$. Assuma que o *array* tem pelo menos dimensão 1, e no caso de ser 1, deve retornar o único elemento que existe. Em caso de empate, retorna o menor dos elementos empatados que se encontra no *array*.

# 2 Análise de desempenho

1. Considere as seguintes funções:

```
fun method1(n: Int) :Int {
  var s = 0
  for ( i in 1 . . n ) {
   s += i
  }
  return s
}
```

```
fun method2(n : Int) : Int {
   var s = 0
   var i = 1
   while( i <= n ){
       s += i
       i = i*2
   }
   return s;
}
```

   Qual a complexidade destas 2 funções em função de `n`? Justique.

2. Considere o algoritmo `xpto`, que recebe como parâmetro dois inteiros $n$ e $m$.

```
fun xpto(n: Int, m: Int): Int {
    return if (n / m == 0) 0 else 1 + xpto(n / m, m)
}
```

   2.1. Considerando que $m = 2$, indique, justificando, a complexidade de `xpto` em função de $n$.

   2.2. Indique, justificando, a complexidade de `xpto`.

# 3 Problema: Filtrar entre palavras

Pretende-se desenvolver uma aplicação que permita juntar de forma ordenada os dados provenientes de vários ficheiros, compreendidos entre duas palavras introduzidas pelo utilizador, produzindo um novo ficheiro de texto ordenado de modo crescente. Os ficheiros originais encontram-se ordenados de modo crescente e contêm uma palavra por linha.

O problema é descrito por:

- Um conjunto $F = \{f_1, ..., f_n\}$ de $n$ ficheiros de texto, em que $n > 0$;
- Duas palavras $word1$ e $word2$ em que $word1$ é lexicograficamente inferior a $word2$;
- Cada linha de um ficheiro $f_n$ tem uma palavra;
- Os ficheiros $f_n$ estão ordenados lexicograficamente, de modo crescente;
- O número total de palavras é $m$.

O objetivo da aplicação a desenvolver é a produção de um novo ficheiro de texto, ordenado de modo crescente, que contenha as palavras dos ficheiros pertencentes ao conjunto $F$, sem repetições e que estejam compreendidas entre as palavras $word1$ e $word2$. O ficheiro produzido deverá também conter uma palavra por linha.

### Parâmetros de execução

A aplicação a desenvolver terá de suportar os seguintes parâmetros:

- `kotlin -Xmx32m filterFiles word1 word2 outputFile sourceFiles ...`

  Esta opção corresponde à produção de um ficheiro de texto designado por `outputFile` que resulta da junção ordenada, sem repetições, das palavras presentes nos ficheiros de texto descritos em `sourceFiles` e compreendidas entre as palavras `word1` e `word2`. A opção `-Xmx32m` estabelece para a JVM um *heap size* máximo de 32 MB.

  Um exemplo de execução é:

  `kotlin -Xmx32 filterFiles anabela joaquim  output.txt f1.txt f2.txt f3.txt`

**Implementação**

1. **Primeira Implementação**: realize esta aplicação, sem utilizar as estruturas presentes em `kotlin.collections` ou `java.util`.
2. **Segunda Implementação**: realize esta aplicação, utilizando as estruturas presentes em `kotlin.collections` ou `java.util` que considere necessárias.

**Valorizam-se soluções** com complexidade temporal $O(m\ log_2\ n)$.

**Avaliação Experimental**

Realize uma avaliação experimental do(s) algoritmo(s) desenvolvido(s) para a resolução deste problema, comparando ambas as implementações. Como exemplo, poderá utilizar os ficheiros que se encontram disponíveis juntamente com este enunciado. Apresente os resultados graficamente, utilizando uma escala adequada.

ISEL, 22 de março de 2022.