```
//ALUNO: JOÃO VICTOR DA SILVA PRADO - PROVA LPF

typealias fInt = (Int)->Int
typealias fFloat = (Float)->Float
typealias fBoolean = (Int) -> Boolean

fun main() {
  println("1A)")
  print("Com recursividade: ")
  println(raizesRec(-5,5){x->x*x-4})
  print("Com biblioteca de coleções Kotlin: ")
  println(raizesCol(-5,5){x->x*x-4==0})

  println("B)")
  val f : fFloat = {x:Float -> x}
  println("Com recursividade: ${somatorioRec(f,1f,10f,2f)}")
  println("Com biblioteca de coleções Kotlin:
${somatorioCol(f,1f,10f,2f)}")

  println("\n2)SEGUNDA QUESTÃO COMENTADA NO CÓDIGO")

  println("\n3)")
  val a:Arvore = Arvore(11, Arvore(5, Arvore(4, Arvore(3, null, null),
null), Arvore(7, Arvore(6, null, null), null)), Arvore(16, Arvore(12,
null, null), Arvore(17, null, null)))
  println(checaArvoreB(a))
}


//1)

//A) COM RECURSIVIDADE
fun raizesRec(i:Int, f:Int, g:fInt):Int{
  val inicio = if(i>f) f else i
  val fim = if(i<f) f else i
  val tam = (fim-inicio)+1
  val arrayAux = IntArray(tam, {inicio + it});
  return fAux(arrayAux, g)
}

fun fAux(array: IntArray, f:fInt, nRaizes:Int=0, pos:Int=0):Int =
  if(pos==array.size)
    nRaizes
  else if(f(array[pos])!=0)
    fAux(array, f, nRaizes,pos+1)
  else
    fAux(array, f, nRaizes+1,pos+1)


//A) COM A BIBLIOTECA DE COLEÇÕES KOTLIN
fun raizesCol(ini:Int, fim:Int, f:fBoolean):Int{
  fun fAux(i: Int, f: Int): List<Int> {
    val inicio = if(i>f) f else i
    val final = if(i<f) f else i
    val tam = final-inicio +1
    if (inicio >= final) return listOf()
```

```
    val arrayAux = IntArray(tam,{inicio + it})
    return arrayAux.toList()
  }

  fun qtdR(l:List<Int>, f:fBoolean):Int{
    if (l.size==0) return 0
    val mList: MutableList<Int> = mutableListOf()
    l.forEach{
      if(f(it) == true) mList.add(1)
    }
    return mList.size
  }
  return qtdR(fAux(ini, fim),f)
}


//B) COM RECURSIVIDADE
fun somatorioRec(f:fFloat, inicio:Float, fim:Float, inc:Float):Float =
  if(inicio>  fim)
    0f
  else
    f(inicio) + somatorioRec(f,inicio+inc,fim,inc)


//B) COM A BIBLIOTECA DE COLEÇÕES KOTLIN
fun somatorioCol(f:fFloat,inicio:Float,fim:Float,inc:Float):Float{
// Na função passada na main: inicio = 1, fim = 10, inc = 2. Ou seja,
o esperado é :
// f(1)+f(3)+f(5)+f(7)+f(9). Aqui temos 5 elementos. Para chegar a
esse número somarei as extremidades que formam o intervalo e dividirei
pelo incremento para chegar nessa quantidade de termos. Porém, esse
número não estará inteiro. Para arredondá-lo pegarei o resto e
dividirei pelo próprio incremento, para saber o valor da casa decimal
do número obtido. Se o valor de 'dec' for maior que 0.6 significa que
arredondaremos o número para cima, caso contrario arredondaremos para
baixo.

  val dec= ((inicio+fim)%inc)/inc
  val qtd = if(dec>0.6)(1-dec)+((inicio+fim)/inc) else
((inicio+fim)/inc) - dec
  val qtdFinal = qtd.toInt()

  //Criarei uma lista com tamanho necessário e transformarei ela com
um "map" para se adequar ao somatório da questão. No fim faço o
somatório de todos os valores da função
  val l = (1..qtdFinal)
  val l2 = l.map{x-> f(inicio+((x-1)*(inc)))}
  val soma = l2.fold(0f){soma:Float, x-> x+soma}
  return soma
}

//REPOSTA DA SEGUNDA QUESTÃO:
// A Avaliação lazy faz parte da abordagem de programação assincrona.
Nela os componentes são calculados apenas quando fossem requisitados,
e suas funções retornam dados parcialmente calculados. Isso resulta em
um tempo menor de inicialização da aplicação. E se você estiver
trabalhando com uma pagina web com muitos dados ela demorará menos a
```

carregar.

// Já em aplicações Eager cada componente deve ser iniciado assim que a aplicação é inicializada. Só que nem sempre o usuario vai necessitar de todos componentes a principio. Além disso faz com que a aplicação demore mais para carregar

// Corrotinas são um exemplo de programação assincrona e representam operações que esperam por algo na maior parte do tempo.
// Ex: escrita e leitura a um banco de dados
// Corrotinas podem suspender uma operação a qualquer momento e essas funçõessão denominadas Suspending Functions.

// Exemplo de aplicação lazy:
// 1. Como já foi citado em aula "sites de noticias": o site carrega apenas as principais noticias da pagina principal e, a medida que o usuario busca outras noticias, o restante vai sendo carregado.
// 2.Jogos: não é necessario carregar todos os niveis de uma vez ja que demoraria muito
// e o usuario é obrigado a seguir uma sequencia pre-estabelecida pelos devs

//3)
```
data class Arvore(val info:Int, val esq:Arvore?, val dir:Arvore?)

fun checaArvoreB(a:Arvore?):Boolean{
  if(a!=null){
    if(a.esq != null && a.esq.info > a.info) return false
    if(a.dir != null && a.dir.info < a.info) return false
    if(!checaArvoreB(a.esq) || !checaArvoreB(a.dir)) return false
  }
  return true
}
```