Programming Exercise 1

1. Write function `nsols` that receives three integers `a`, `b`, `c` and determines the number of solutions of the equation

```
def nsols (a : Int, b: Int, c: Int) : Int = {
  if (a == 0)
    0
  else {
    if ((b * b - 4 * a * c) == 0)
        1
      else
          2
  }
}
```

2. Write recursive function `min` that uses tail recursion to find the least value in the given list of integers (assume that the given list is non-empty).

```
def min (l : List[Int]) : Int = {
    def _min (l: List[Int], m: Int) : Int = {
      l match {
        case Nil => m
        case h::t => {
            if (h < m) _min(t, h) else _min(t, m)
        }
      }
    }

    l match {
      case Nil => 0
      case h::t => _min(t, h)
```

```
    }
}
```

3. Write recursive function `desc` that receives a positive integer `n`, and returns the list of numbers from `n` to 1 in descending order.

```
def desc (n : Int) : List[Int] = {
  if (n > 0) n::desc(n - 1)
  else Nil
}
```

4. Write recursive function `get_elem` that receives a list of integers `l`, and a non-negative integer `i`, and generate a list containing the integer at the `i`-th index in if exists; otherwise, empty list.

```
def get_elem (l : List[Int], i: Int) : List[Int] = {
    l match {
        case Nil => Nil
        case h::t => if (i == 0) h::Nil else get_elem(t, i
    }
}
```

5. Write recursive function `merge_list` that receives two ascending list of integers and generates the merged ascending list.

```
def merge_list (l1 : List[Int], l2: List[Int]) : List[Int]
    (l1, l2) match {
        case (Nil, Nil) => Nil
        case (Nil, h::t) => h::merge_list(Nil, t)
        case (h::t, Nil) => h::merge_list(t, Nil)
        case (h1::t1, h2::t2) => {
            if (h1 < h2) h1::merge_list(t1,l2)
            else h2::merge_list(l1, t2)
        }
```

```
    }
}
```

6. Write recursive function `bin_search` that receives a monotonic function `p : Int => Boolean`, and two integers `low` and `high` which indicate the upper bound and lower bound of the search, and then find the first least `x` in between low and high such that `p(x)` is true.

```
def bin_search (p : Int => Boolean, low : Int, high : Int)
  val mid : Int = (low + high) / 2

  if (low == high) high
  else {
    if (p(mid) == false) bin_search(p, mid + 1, high)
    else bin_search(p, low, mid)
  }
}
```