Homework 9: Sorting Algorithms in Scala

Stephen Wagstaff
CS 431
November 12, 2018

---

1. Write a function `merge_sort(lst: List[Int]): List[Int]` that takes a list, and return a sorted list in **increasing order**.

```
def merge_sort(initialList: List[Int]): List[Int] = {

  def split: (List[Int], List[Int]) = {
    val cut = initialList.length/2
    ( initialList take cut , initialList drop cut )
  }

  def merge(leftList: List[Int], rightList: List[Int]): List[Int] = {
    (leftList, rightList) match {
      case (Nil, _) => rightList
      case (_, Nil) => leftList
      case ((leftElement::leftRemaining), (rightElement::rightRemaining)) =>
        if (leftElement > rightElement) leftElement::merge(leftRemaining, rightList)
        else rightElement::merge(leftList, rightRemaining)
    }
  }

  initialList match{
    case Nil => Nil
    case x::Nil => x::Nil
    case _ =>
      val (left, right) = split
      merge (merge_sort(left), merge_sort(right))
  }
}
```

2. Write a function `selection_sort(lst: List[Int]): List[Int]` that takes a list and return a sorted list in **increasing order**.

```
def selection_sort(initialList: List[Int]): List[Int] = {
    def select(selectList: List[Int]): List[Int] = {
      selectList match {
        case Nil => Nil
        case (lastElement::Nil) => lastElement::Nil
        case (firstElement::list) =>
          val (testElement::remaining) = select(list)
          if(testElement > firstElement) select(testElement::firstElement::remaining)
          else firstElement::testElement::remaining
      }
    }
    initialList match {
      case Nil => Nil
      case _ => select(initialList)
    }
  }
```

3. Write a function `insertion_sort(lst: List[Int]): List[Int]` that takes a list and return a sorted list in **increasing order**.

```
def insertion_sort(initialList: List[Int]): List[Int] = {
    def insert(element: Int, list: List[Int]): List[Int] ={
      (element, list) match {
```

```
      case (_, Nil) => List(element)
      case (_, head::rest) =>
        if(element > head) element::head::rest
        else head::insert(element, rest)
    }
  }
  def sort(sofar: List[Int], list: List[Int]): List[Int] = {
    list match {
      case Nil => sofar
      case (element::rest) => insert(element, sort(sofar,rest))
    }
  }

  initialList match {
    case Nil => Nil
    case _ => sort(Nil, initialList)
  }
}
```