```scala
package partitions

import scala.annotation.tailrec

def take(numbers: List[Int], target: Int): Option[List[Int]]  =  (numbers) match {
  case _ if target == 0 =>Some(List.empty[Int]) //Success base case
  case Nil => None //failure base case
  case h :: tail => //recursive case
    take(tail, target - h)  match { //see if h works
      case Some(l) => Some(h :: l) //h might work since there's still elements in the list
      case None => take(tail, target) //went through all the elements and h doesn't work, try the next element
    }
}
def partition2(numbers: List[Int]): Option[Partition]         = {
  val weight = numbers.sum
  if (weight == 0) {
    throw new IllegalArgumentException()
  }
  if (weight % 2 == 0) {
    val l1 = take(numbers, weight / 2)
    l1 match {
      case Some(ls) => Some(List(ls, numbers.diff(ls)))
      case None => None
    }
  } else {
    None
  }

}
def partition(numbers: List[Int], k: Int): Option[Partition]  = {

```

```
32   def rec(numbers: List[Int],targets:List[Int], part:
   Partition): Option[Partition] = numbers match {
33     case Nil =>//If the list is empty
34       if (targets.forall(_ == 0)) Some(part) //if all
   targets are 0, success
35       else None //if any target is not 0, failure
36     case h :: tail =>
37       @tailrec
38       def rec_targets(i: Int): Option[Partition] = { //
   try to put h into the ith list of the partition and see
    if it works
39         if (i >= k) None //iterate through elements of
   target and part
40         else {
41           val new_targets = targets.updated(i, targets(
   i) - h) //new targets update element i
42           val new_part = part.updated(i, h :: part(i))
   //new partition with head of numbers to prepended to
   the ith list
43           rec(tail, new_targets, new_part) match { //
   recurse down branch
44             case Some(part) => Some(part)    // found a
    valid partition down this branch
45             case None            => rec_targets(i + 1)
   // try next subset
46           }
47         }
48       }
49
50       rec_targets(0)
51   }
52
53   val weight = numbers.sum
54   if (weight == 0 || k == 0) {
55     throw new IllegalArgumentException()
56   }
57   if (weight % k != 0) {
58     None
```

```scala
  } else {
    val part = List.fill(k)(List.empty[Int])
    val targets = List.fill(k)(weight / k)

    //It might be a good idea to map the targets to the elements of the partition in some way
    // since their directly related to each other and will always be at the same index but I'm running out of time
    rec(numbers, targets, part)
  }

}

def partition0(numbers: List[Int], k: Int): Option[Partition] = ???

```