```
  val v = myMax()
  while(i < s.length) {
    if(s(i) == v)
      freq = freq + 1
    i = i + 1
  }
  freq
}
```

hopcamp1 O(n)

7. Kerro lyhyesti puolitushaun idea sanallisesti (eli antamatta ohjelmakoodia).

   Oletetaan, että käytetään puolitushakua vastaamaan kysymykseen "sisältääkö alla oleva taulukko lukua 63?". Mitä taulukon alkioita puolitushaku käsittelee eli vertaa lukuun 63?

```
Array(2,3,4,5,21,29,31,34,35,35,37,500,1000)
```

8. Tarkastellaan alla olevaa funktiota `f`. Kerro sanallisesti, millaisen arvon `f` laskee. Anna suorituksen kutsupino kun funktiota `f` kutsutaan argumentin `l` ollessa `List(1,2,6)`.

```
def f(l: List[Int]): Double = {
  def inner(r: List[Int], s: Int, n: Int): (Int, Int) = {
    if(r.isEmpty) (s, n)
    else inner(r.tail, s + r.head, n + 1)
  }
  val (s, n) = inner(l, 0, 0)
  if(n == 0) 0.0
  else s.toDouble / n.toDouble
}
```

9. Tarkastellaan oppimateriaalissa esitettyjä linkitettyjen listojen luokkia. Onko alla oleva metodi `count` häntärekursiivisessa muodossa? Jos ei ole, kerro miksei ole ja anna funktionaalisella tyylillä toteutettu vastaava häntärekursiivinen versio.

```
abstract class LinkedList[A] {
  def isEmpty: Boolean
  def head: A
  def tail: LinkedList[A]
  def count(p: A => Boolean): Int = {
    this match {
      case Nil() => 0
      case Cons(h, t) =>
        if(p(h)) 1 + t.count(p)
        else t.count(p)
    }
  }
}
case class Nil[A]() extends LinkedList[A] {
  def isEmpty = true
  def head = throw new java.util.NoSuchElementException("head of empty list")
  def tail = throw new java.util.NoSuchElementException("tail of empty list")
}
case class Cons[A](val head: A, val tail: LinkedList[A]) extends
    LinkedList[A] {
  def isEmpty = false
}
```