Task

- Imagine that the following condition didn't exist in `BinarySearch` (lines 45-46).

```
if (beginIndex > endIndex)
  return -1;
```

- Give a list of integers containing more than one key, sorted in ascending order, that would make `BinarySearch` throw a different kind of exception or error. Explain why this would be thrown.

Solution

Take for example

MyList

Index	0	1
Key	7	9

An out of bounds error will occur when searching for a number that both doesn't exist in the list and is smaller than the first index key.

Ex.
search(MyList , 5);

1) Walking through the search, my request becomes search(MyList,5,0,1)
2) beginIndex ≠endIndex as 0 ≠ 1
3) middleIndex is an Int variable so when calling upon the method MathUtils.getMiddleIndex we are returned 0 as $\frac{1}{2} = 0$ ←main reason we end up with a negative index later on
4) When we call the .comparTo function we end receiving a -1, 0 or 1 depending on if they key we are searching for (5 in this case) is less than, equal to, or greater than the key value at middleIndex (i.e index 0, key value of 7 at this moment) 5 < 7 so diff = -1
5) because diff < 0 and we are searching recursively, the search request turns into "search(MyList, 5, 0, -1)"
6) obviously because there is no negative indexes when we reach the part where we key.compareTo(list.get(middleIndex)), there will be an error when trying to access data from an index of -1 because that is outside of the bounds of MyList.

TLDR: Searching for a number that both doesn't exist in the list and is smaller than the first index key will result in an out of bounds error. Why? because middleIndex is an int so when finding the middle of a list of 2 elemets,5 elements , 11 element list (2x +1 from previous?) you'll end up dividing 1 by 2 and getting 0, which then is followed by subtracting 1 giving you a negative index.