CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce

343 Davis Hall

alphonce@buffalo.edu

Office hours:

Tuesday 10:00 AM – 12:00 PM*

Wednesday 4:00 PM – 5:00 PM

Friday 11:00 AM – 12:00 PM

OR request appointment via e-mail

**Tuesday adjustments: 11:00 AM – 1:00 PM on 12/6*

ROADMAP

Last time

- exercise 09 solution
- exercise 10
- linear search

Today

- exercise 10 solution
- exercise 11 (?)
- binary search

Coming up

- Q & A session

ANNOUNCEMENTS

OFFICE HOURS

Today's office hours will end at 4:45.

Next week – no regular office hours. E-mail me with questions or to set up an appointment.

We are looking for rooms for the final exam review session – stay tuned!

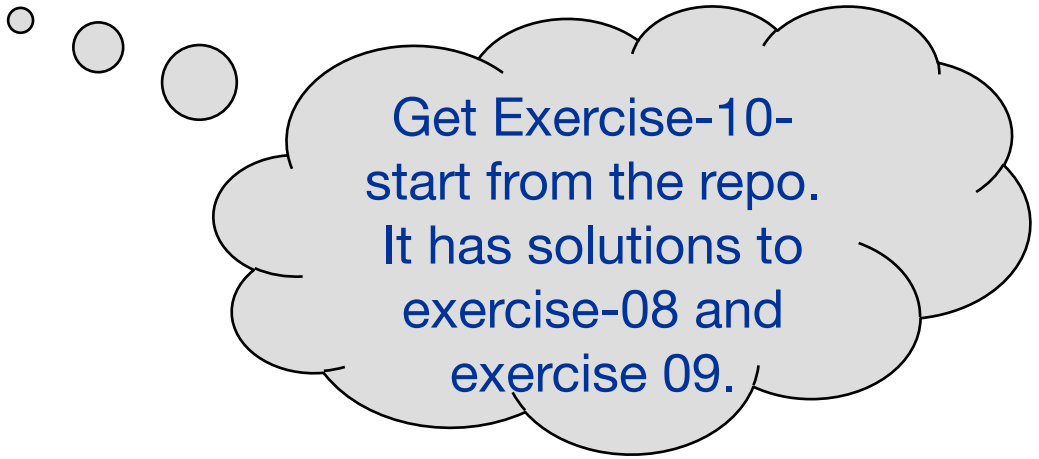
EXERCISE 10
solution

EXERCISE 10

Define a method a method with this header in a class named quiz.Question:

```
public HashSet<HashSet<Point>> partition(ArrayList<ArrayList<String>> board)
```

Define the method so that it returns a HashSet<HashSet<Point>> that is a partition of the given board into disjoint and covering sets of contiguous and matching points from the board.



```
public HashSet<HashSet<Point>> partition(ArrayList<ArrayList<String>> board) {

}
```

```
public HashSet<HashSet<Point>> partition(ArrayList<ArrayList<String>> board) {
  HashSet<HashSet<Point>> partition = new HashSet<HashSet<Point>>();









  return partition;
}
```

```
public HashSet<HashSet<Point>> partition(ArrayList<ArrayList<String>> board) {
  HashSet<HashSet<Point>> partition = new HashSet<HashSet<Point>>();
  if (board != null) {









  }
  return partition;
}
```

```
public HashSet<HashSet<Point>> partition(ArrayList<ArrayList<String>> board) {
  HashSet<HashSet<Point>> partition = new HashSet<HashSet<Point>>();
  if (board != null) {
    ArrayList<Point> pointsToCheck = allPoints(board);









  }
  return partition;
}
```

```
public HashSet<HashSet<Point>> partition(ArrayList<ArrayList<String>> board) {
  HashSet<HashSet<Point>> partition = new HashSet<HashSet<Point>>();
  if (board != null) {
    ArrayList<Point> pointsToCheck = allPoints(board);









  }
  return partition;
}

private ArrayList<Point> allPoints(ArrayList<ArrayList<String>> board) {
  ArrayList<Point> listOfPoints = new ArrayList<Point>();
  for (int row=0; row<board.size(); row=row+1) {
    for (int col=0; col<board.get(0).size(); col=col+1) {
      listOfPoints.add(new Point(row, col));
    }
  }
  return listOfPoints;
}
```

```
public HashSet<HashSet<Point>> partition(ArrayList<ArrayList<String>> board) {
  HashSet<HashSet<Point>> partition = new HashSet<HashSet<Point>>();
  if (board != null) {
    ArrayList<Point> pointsToCheck = allPoints(board);
    while (! pointsToCheck.isEmpty()) {
      Point p = pointsToCheck.get(0);
      HashSet<Point> region = maximalMatchedRegion(p, board);
      pointsToCheck.removeAll(region);
      partition.add(region);
    }
  }
  return partition;
}

private ArrayList<Point> allPoints(ArrayList<ArrayList<String>> board) {
  ArrayList<Point> listOfPoints = new ArrayList<Point>();
  for (int row=0; row<board.size(); row=row+1) {
    for (int col=0; col<board.get(0).size(); col=col+1) {
      listOfPoints.add(new Point(row, col));
    }
  }
  return listOfPoints;
}
```

```
public HashSet<HashSet<Point>> partition(ArrayList<ArrayList<String>> board) {
  HashSet<HashSet<Point>> partition = new HashSet<HashSet<Point>>();
  if (board != null) {
    ArrayList<Point> pointsToCheck = allPoints(board);
    while (! pointsToCheck.isEmpty()) {
      Point p = pointsToCheck.get(0);
      HashSet<Point> region = maximalMatchedRegion(p, board);
      pointsToCheck.removeAll(region);
      partition.add(region);
    }
  }
  return partition;
}

private ArrayList<Point> allPoints(ArrayList<ArrayList<String>> board) {
  ArrayList<Point> listOfPoints = new ArrayList<Point>();
  for (int row=0; row<board.size(); row=row+1) {
    for (int col=0; col<board.get(0).size(); col=col+1) {
      listOfPoints.add(new Point(row, col));
    }
  }
  return listOfPoints;
}
```

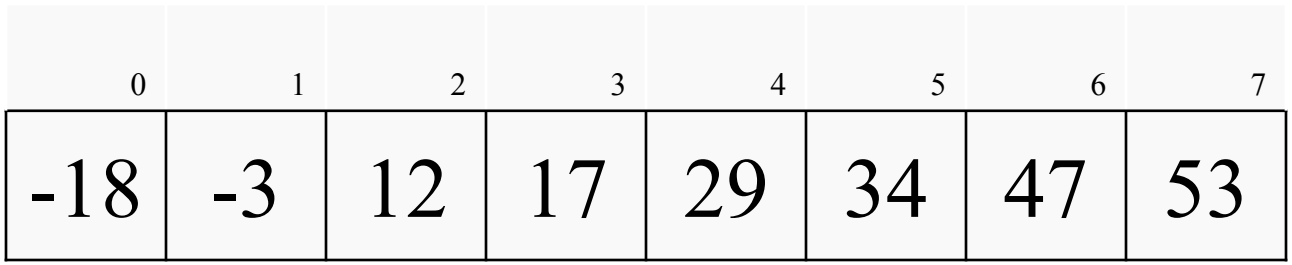
BINARY SEARCH

BINARY SEARCH

(data must be sorted:
ordered from smallest to largest)

Visualization

Let's find 53 using binary search.
Initial interval is [0,8)

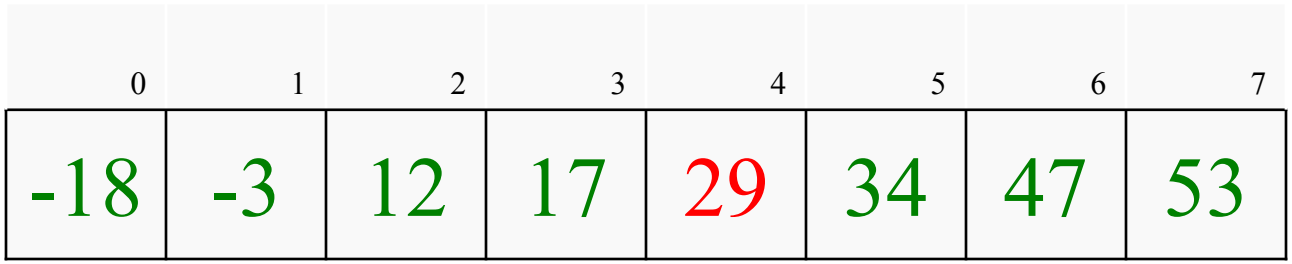


Visualization

Compare the target to the value at the mid-point index of the interval.
For interval [0,8), midpoint index is (0+8)/2 = 4.

target

53

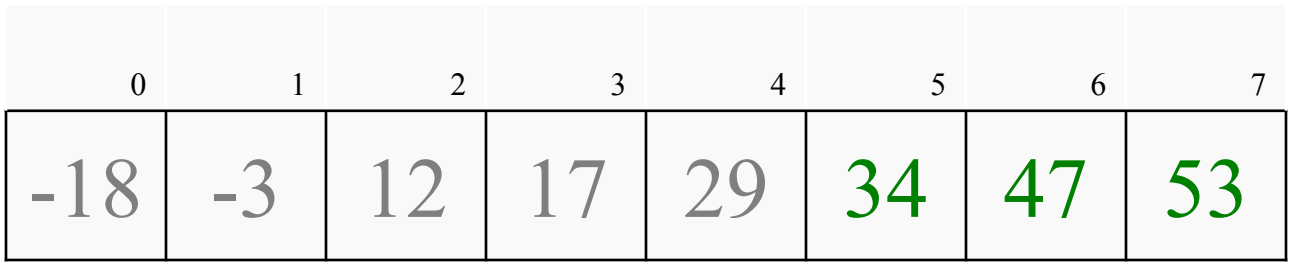


Visualization

Since 53 > 29, focus attention on interval to the right of 29, [5,8).

target

53



Visualization

Compare the target to the value at the mid-point index of the interval.
For interval [5,8), midpoint index is (5+8)/2 = 6.

target

53

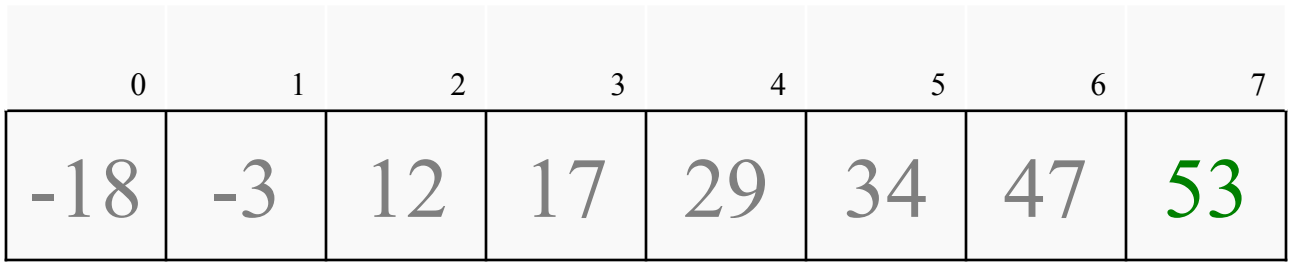
0	1	2	3	4	5	6	7
-18	-3	12	17	29	34	47	53

Visualization

Since 53 > 47, focus attention on interval to the right of 47, [7,8).

target

53



Visualization

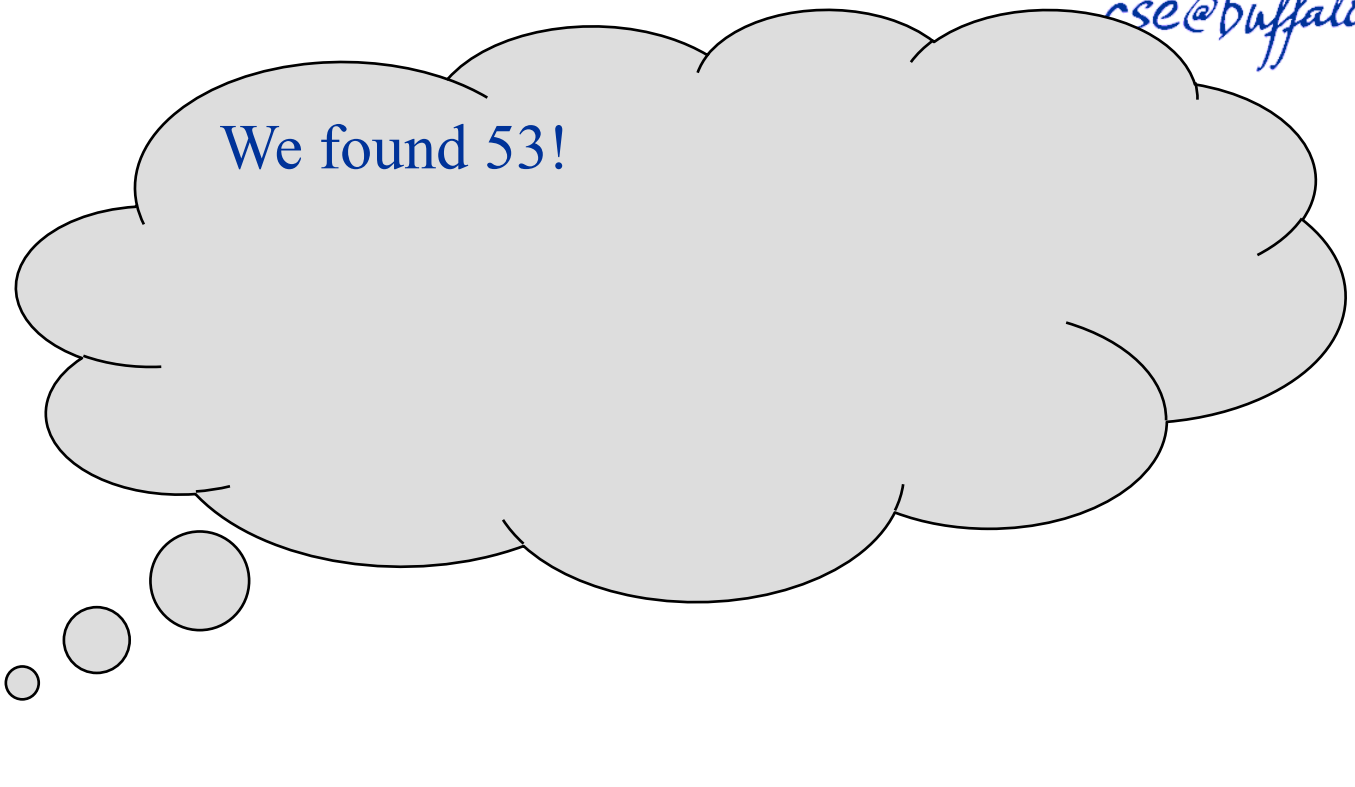
Compare the target to the value at the mid-point index of the interval.
For interval [7,8), midpoint index is (7+8)/2 = 7.

target

53

0	1	2	3	4	5	6	7
-18	-3	12	17	29	34	47	53

Visualization



0	1	2	3	4	5	6	7
-18	-3	12	17	29	34	47	53

```
public boolean isMemberOf(int target, List<Integer> list)
{
    int left = 0;
    int right = list.size();
    while ( left != right) {
        int mid = (left + right) / 2;
        if (list.get(mid) == target) { return true; }
        if (target < list.get(mid))  { right = mid; }
        if (list.get(mid) < target)  { left = mid + 1; }
    }
    return false;
}
```

EXERCISE 11

EXERCISE 11

There will be no submission of this exercise.

Exclude points from matches if they are not in a 3-in-a-row or 3-in-a-column.

Exclude matches if they are not of minimum size 3.

```
private HashSet<Point> trimmedRegion(HashSet<Point> in) {
  HashSet<Point> out = new HashSet<Point>();
  for (Point p : in) {
    if (inRow(p,in) || inCol(p,in)) {
      out.add(p);
    }
  }
  return out;
}

private boolean inCol(Point p, HashSet<Point> in) {
  return ( in.contains(new Point(p.x,p.y-2)) && in.contains(new Point(p.x,p.y-1)))
 ||      ( in.contains(new Point(p.x,p.y-1)) && in.contains(new Point(p.x,p.y+1)))
 ||      ( in.contains(new Point(p.x,p.y+1)) && in.contains(new Point(p.x,p.y+2)));
}


private boolean inRow(Point p, HashSet<Point> in) {
  return ( in.contains(new Point(p.x-2,p.y)) && in.contains(new Point(p.x-1,p.y)))
 ||      ( in.contains(new Point(p.x-1,p.y)) && in.contains(new Point(p.x+1,p.y)))
 ||      ( in.contains(new Point(p.x+1,p.y)) && in.contains(new Point(p.x+2,p.y)));
}
```