Precept Outline
- Review of Lectures 13 and 14:
  - Hash Tables
  - K-d Trees

Relevant Book Sections
- Book chapters: 3.4

A. Review: Hash Tables + K-d Trees

Your preceptor will briefly review key points of this week's lectures.

B. Hash Tables

Part 1: Linear probing vs. separate chaining

Consider the following sequence of (integer) insertions into an empty hash table of capacity 11, where the hash function `h(i)` returns `i % 11`:

```
15, 90, 53, 100, 34, 65, 20
```

Suppose that the symbol table is implemented as a *linear-probing* hash table. Which positions of the `keys[]` array contain null values after all 7 insertions?

Suppose we perform the same sequence of insertions with the same hash function, but use a *separate-chaining* table instead. What linked list sizes are present in the table after all 7 insertions?

Part 2: Designing hash methods

Suppose that you are implementing a symbol table as a hash table, where the keys are the following integers:

`5, 20, 40, 55, 64`

You decide to use the hash function `h(i) = i % c`, where `c` is the capacity of the hash table. What is the smallest value of `c` you can pick such that there are no collisions?

Since you have mastered hashing integers, you decided you wanted a bigger challenge: strings. Consider the following two Java methods that hash a string of lowercase letters `s`:

```
public int hashMethod1(String s) {
  int hash = 0;
  for (int i = 0; i < s.length(); i++)
    hash = hash + (s.charAt(i) - 'a') % 13;
  return hash;
}
```

```
public int hashMethod2(String s) {
  int hash = 0;
  for (int i = 0; i < s.length(); i++)
    hash = (hash * 13) + (s.charAt(i) - 'a');
  return hash;
}
```

For each of them, find a pair of distinct strings of length 5 containing only lowercase letters that hash to the same value.

C. K-d Trees

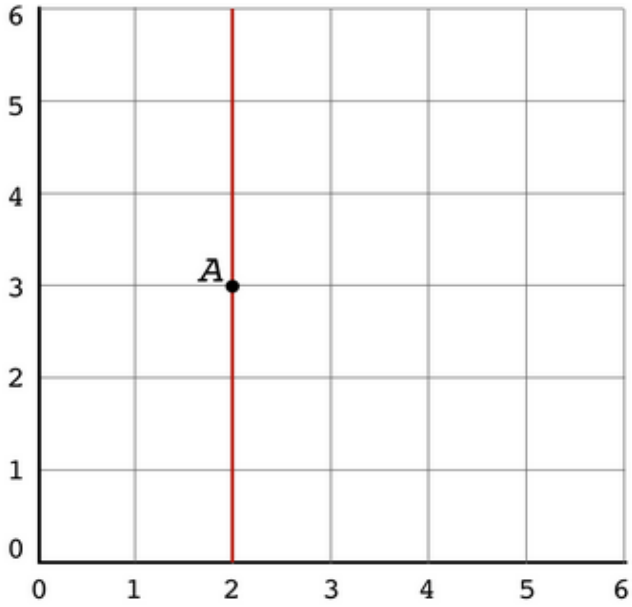
Draw the 2d-tree that results from inserting the following points:

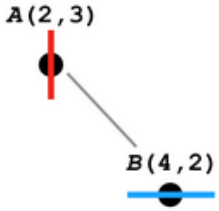
A	B	C	D	E	F	G
$(2,3)$	$(4,2)$	$(4,5)$	$(3,3)$	$(1,5)$	$(4,4)$	$(1,1)$

Additionally, draw each point on the grid, as well as the vertical or horizontal line that runs through the point and partitions the plane or a subregion thereof.

Recall: We take the convention that while inserting, we move left if the coordinate of the inserted point is less than the coordinate of the current node; and go right if it is greater than **or equal**.

Use the images below to draw your grid/tree.





Determine each point's bounding box and fill them into the table below.

A	$[-\infty,\infty] \times [-\infty,\infty]$
B	
C	
D	
E	
F	
G	

Number the (non-null) nodes in the sequence they are visited by a *range query* with the rectangle shown below. Which subtrees are pruned? (Some null subtrees may be pruned, and some may not be.)

Remember. The range search algorithm recursively searches in both the left and right subtrees unless the bounding box of the *current* node does not intersect the query rectangle. If both do, our convention is to visit the left one first.