Announcements

- MP5 is due on Friday, 04/01 @ 11:59pm; MP6 will be released on Friday, 04/01

THE PROGRAMMING QUESTION:

You will be provided a .zip file containing the following files:

- Makefile
- random.h, ...(auxiliary code for our own use)
- main.cpp
- sphere.cpp
- sphere.h
- sphere-p9.cpp
- sphere-p10.cpp

You will need to make changes to ONLY three files and upload them inividually:

- sphere.h // only add 2 function signatures;
- sphere-p9.cpp // add function signature and implementation
- sphere-p10.cpp // add function signature and implementation

The given code compiles initially!

Your code NEEDS to compile, we will give partial credit only if your code compiles.

If your code does not compile - you get 0.

The Makefile

```
# the compiler:
CC = g++

# compiler flags:
CFLAGS  = -g -Wall

default: sample

sample: main.o sphere.o sphere-p9.o sphere-p10.o
	$(CC) $(CFLAGS) -o sample sphere.o sphere-p9.o sphere-p10.o main.o

sphere.o: sphere.cpp sphere.h
	$(CC) $(CFLAGS) -c sphere.cpp

sphere-p9.o: sphere-p9.cpp sphere.h
	$(CC) $(CFLAGS) -c sphere-p9.cpp

sphere-p10.o: sphere-p10.cpp sphere.h
	$(CC) $(CFLAGS) -c sphere-p10.cpp


main.o: main.cpp random.h
	$(CC) $(CFLAGS) -c main.cpp

clean:
	$(RM) sample *.o *~
```

random.h

```
// This is a utility function for our use only.

#include <iostream>
using namespace std;

template <class T>
class print{
 public:
  void operator()(T x, string description = ""){
    cout << "printing " << description << "\t... " << x << endl;
  }
};
```

main.cpp

```
// The main function.
// You can write your test cases here.
// You can see how we use various functions here, too.
#include "random.h"
#include "sphere.h"

int main() {
  print<double> printDouble;
  print<string> printString;

  //test cases for random.h
  printDouble(2.0);
  printString("sample string");

  //test cases for sphere.cpp
  sphere a (4.0, "Sphere A");
  printDouble(a.getRadius(), "a's radius");
  printString(a.getName(), "a's name");

  //test cases for sphere-p9.cpp


  //test cases for sphere-p10.cpp


}
```

sphere.h

```
#include <iostream>
using namespace std;

class sphere {
 public:
   sphere();
   sphere(double r, string n);
   double getRadius();
   string getName();

   //question 9
   double getDiameter();

   //question 10
   void printBunny();
 private:
   double radius;
   string name;
};
```

sphere.cpp

```
#include "sphere.h"

sphere::sphere() {
   radius = 1.0;
   name = "Default Sphere";
}

sphere::sphere(double r, string n) {
   radius = r;
   name = n;
}

double sphere::getRadius(){
   return radius;
}

string sphere::getName(){
   return name;
}
```

Your solution in: sphere-p9.cpp and sphere-p10.cpp

```
#include "sphere.h"

double sphere::getDiameter(){
  return 2*radius;
}

```

```
-:---   sphere-p9.cpp    All L7      (C++/l Abbrev)
```

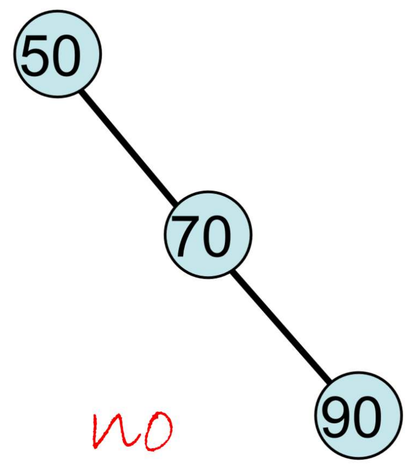
```
#include <iostream>
#include "sphere.h"
using namespace std;

void sphere::printBunny() {
  cout << "printing a bunny...." << endl ;
  cout << "\t  /)/)" << endl;
  cout << "\t ( ..)" << endl;
  cout << "\tc(\")(\")" << endl;
  cout << "     ^^^^^^^^^^^^^^^^" << endl;
}
```

```
-:---   sphere-p10.cpp    All L12     (C++/l Abbrev)
```

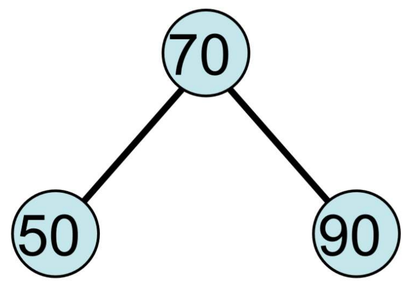
Reminder from last time:

b = 2



Height balanced? no

b = 0



yes

The “height balance” of a tree T is:

$$b = height(T_L) - height(T_R)$$

A tree T is “height balanced” if:

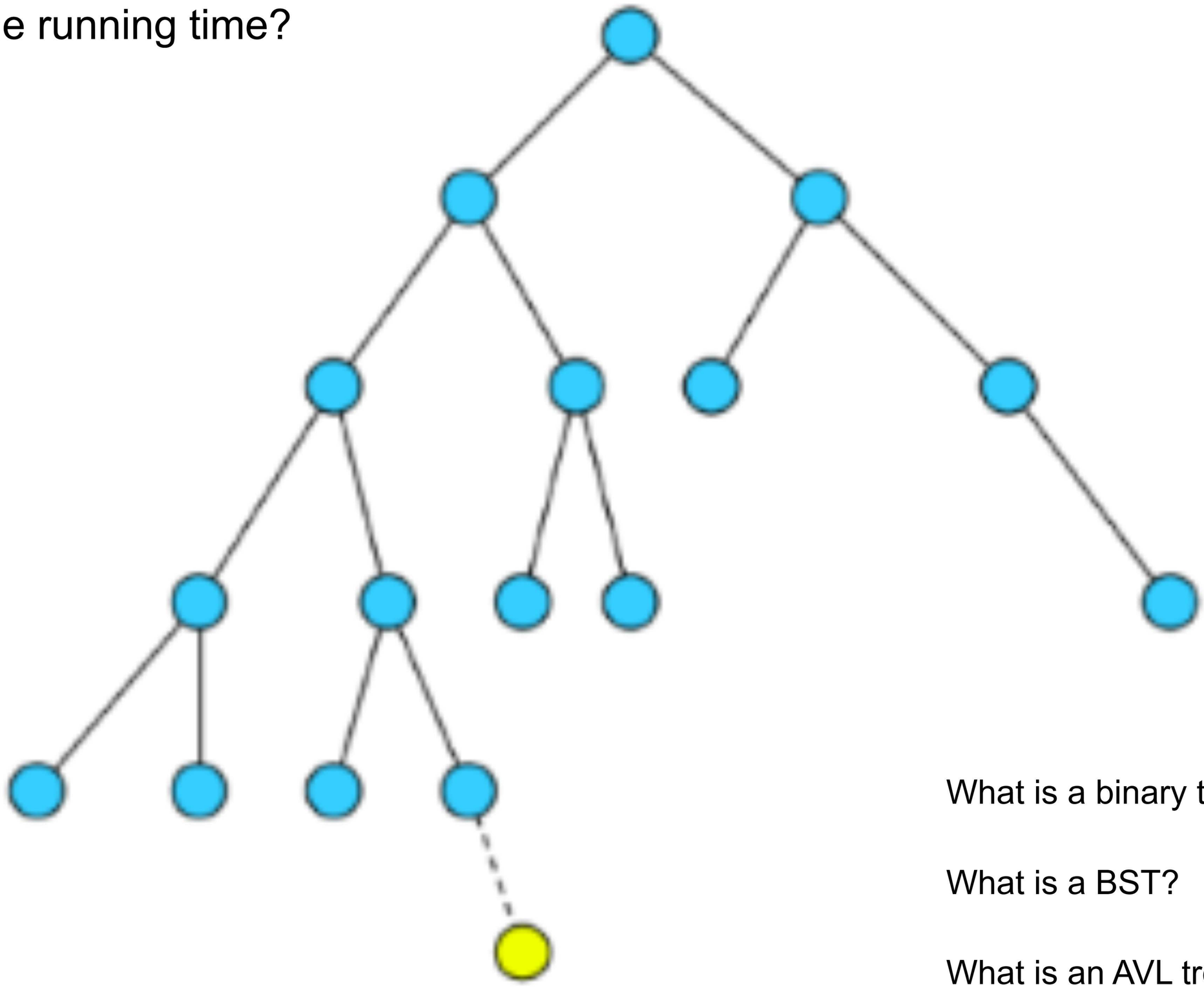
- T = {}
- T = {r, T_L, T_R},
  |b| <= 1, T_L and T_R are both height balanced

We call these trees AVL trees

How many nodes can be out of balance after an **insertion**?

How many rotations will be needed to fix it?

What is the running time?



What is a binary tree?

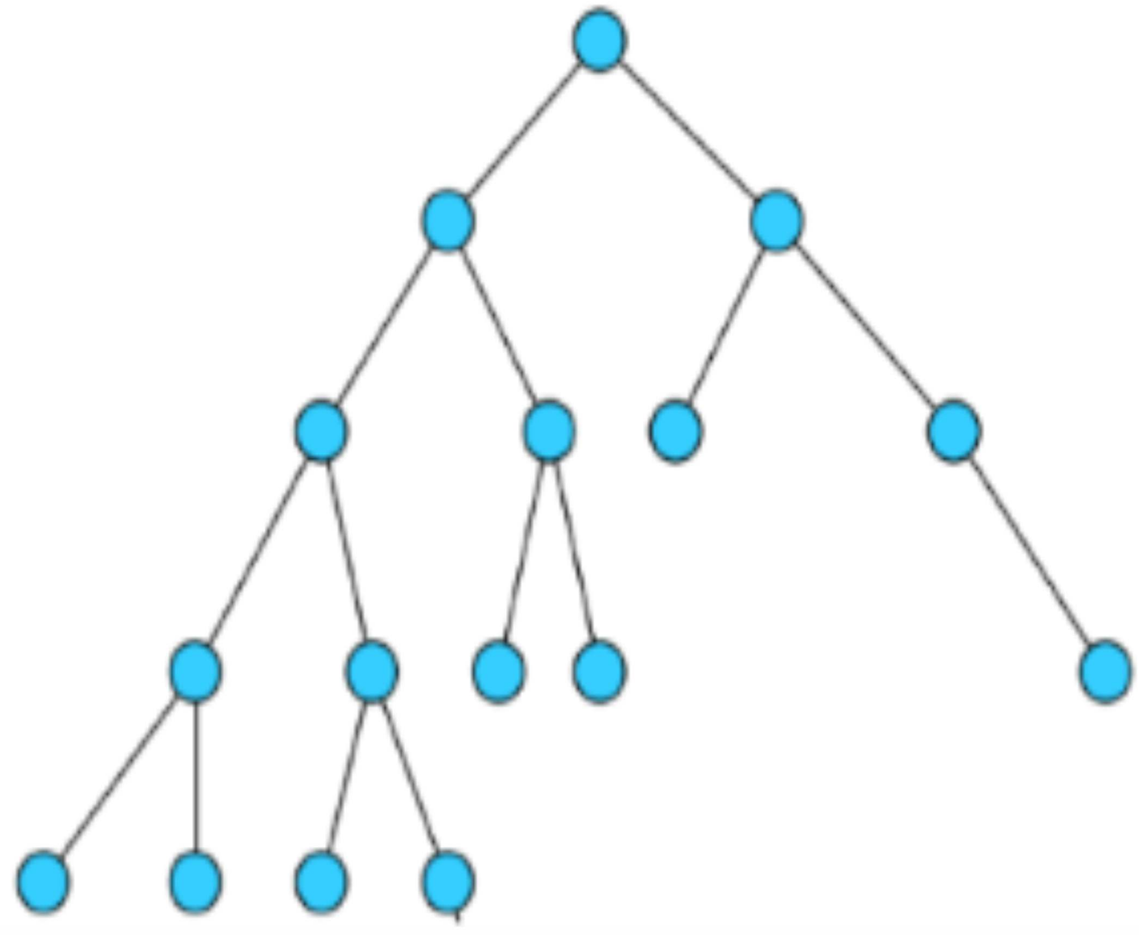
What is a BST?

What is an AVL tree?

How many nodes can be out of balance after one **deletion**?

How many rotations will be needed to fix it?

What is the running time?

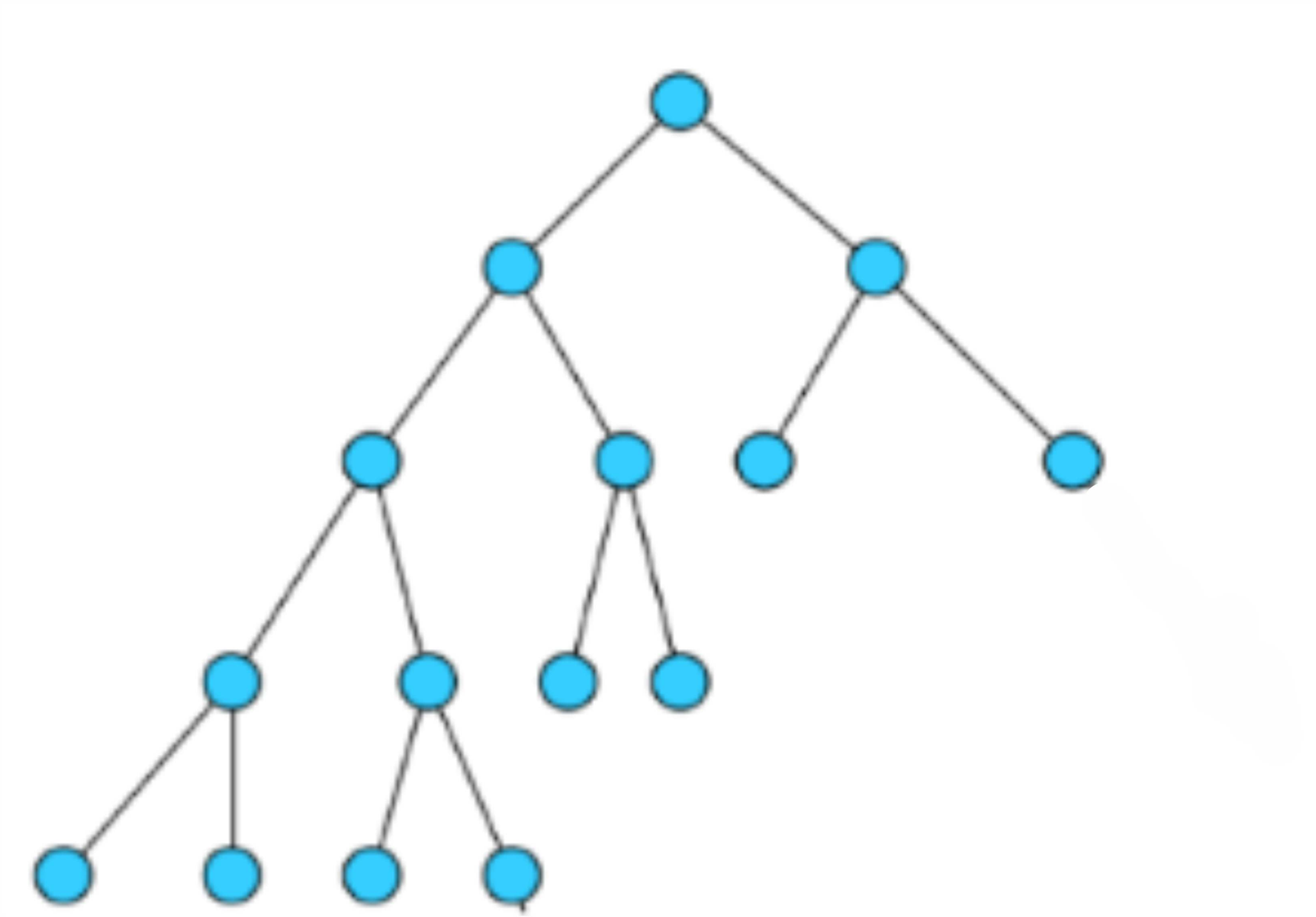


How to know which rotation needs to be applied?

How to perform a needed rotation? http://www.qmatica.com/DataStructures/Trees/AVL/AVLTree.html

How many possible rotations are there?

What is the running time?

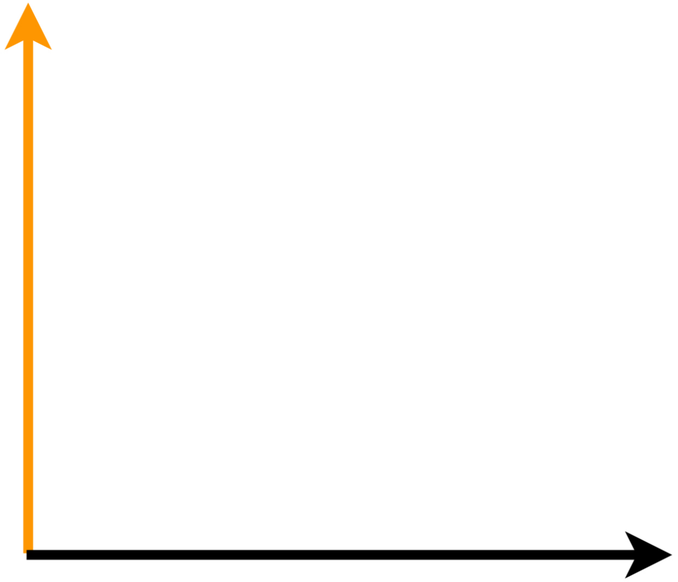


AVL tree analysis:

Since running times for Insert, Remove and Find are O(h), we'll argue that h = O(log n).

- Defn of big-O:

- Draw two pictures to help us in our reasoning:



- Putting an upper bound on the height for a tree of n nodes is the same as putting a lower bound on the number of nodes in a tree of height h.