Announcements

- MP7 is out, on Tue, 05/03 @ 11:59pm; E/C due on Tue, 04/26 @ 11:59pm
- **Exam 4 topics:**
- **MC:** b-trees, hash tables, huffman trees, kd-trees, priority queue/heaps, disjoint sets, run time.
- **Coding**: lab_hash, lab_heaps and lab_dict.

THE **HEAPs** CODING QUESTION:

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

- Makefile
- MinHeap.cpp/.h MaxHeap.cpp/.h
- main.cpp
- heap.cpp

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

- heap.cpp // function signature will be provided, only implementation is needed

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.

No hand grading!

The Makefile

```
HEAP_Q_NO = heap.cpp
OBJS = main.o MaxHeap.o MinHeap.o
EXENAME = heap.exe

CXX = g++
CXXFLAGS = -std=c++0x -c -g -O0 -Wall -Wextra -pedantic
LINK = g++
LDFLAGS = -std=c++0x

all : $(EXENAME)

$(EXENAME) : $(OBJS)
	$(LINK) $(OBJS) $(LDFLAGS) -o $(EXENAME)

heap_qE_main_$N.o : main.cpp MinHeap.h MinHeap.o MaxHeap.h MaxHeap.o $(HEAP_Q_NO)
	$(CXX) $(CXXFLAGS) main.cpp

MinHeap.o: MinHeap.h MinHeap.cpp
	$(CXX) $(CXXFLAGS) MinHeap.cpp

MaxHeap.o: MaxHeap.h MaxHeap.cpp
	$(CXX) $(CXXFLAGS) MaxHeap.cpp

heap_qE_$N.o: $(HEAP_Q_NO) MinHeap.h MaxHeap.h
	$(CXX) $(CXXFLAGS) $(HEAP_Q_NO)


clean :
	rm -f *.o $(EXENAME)
```

MinHeap.cpp & MinHeap.h

```cpp
#ifndef MINHEAP_H
#define MINHEAP_H

#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>
using namespace std;

class MinHeap
{
public:
    vector<int> elements;

    void heapifyDown(int index);

    void heapifyUp(int index);

    void buildHeap();

    MinHeap(vector<int> vector);

    MinHeap();

    void insert(int newValue);

    int peek();

    int pop();

    void print();

};

#endif //MINHEAP_H
```

main.cpp

```
#include <iostream>
#include "MaxHeap.h"
#include "MinHeap.h"
using namespace std;
#include "heap.cpp"

// Use this main() function to test your code.
// You may change things as you wish here, this is for your own
// use. Please note: This code will not be used for grading.
int main()
{
  BinaryTree t0;
  t0.notCompleteTree(5);
  t0.printTree();
  checkMinHeap(t0);


  BinaryTree t2;
  t2.notCompleteTree(2);
  t2.printTree();
  checkMinHeap(t2);


  BinaryTree t3;
  t3.completeTree(5, 4);
  t3.printTree();
  checkMinHeap(t3);


return 0;
}
```

1. All solutions are NOT member functions of heap class.

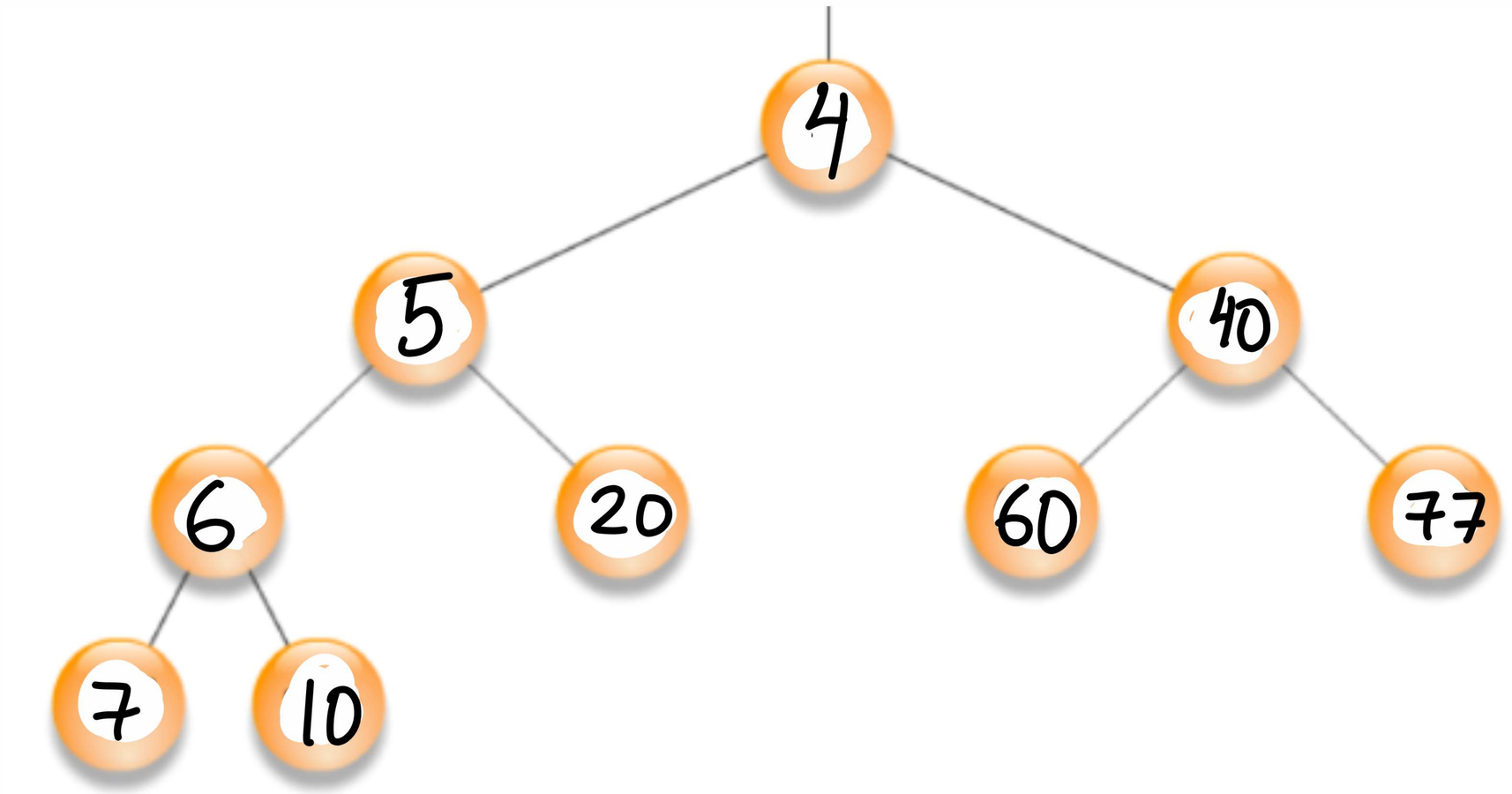
2. We will NOT look at your main.cpp but you might want to use it for your test cases

3. You will need to type "make upload " or just drag and drop the solution file as before.

Your solution in: heap.cpp

```
#include "binaryTree.h"


bool checkMinHeap(BinaryTree t) {
  // Create an array from level order traversal of t:
  int* arr = t.levelData();
  cout << "the tree is complete " << arr[0] << endl;

  bool isMinHeap = arr[0];

  cout << "the tree is a min heap " << isMinHeap << endl;

  return isMinHeap;
}
```

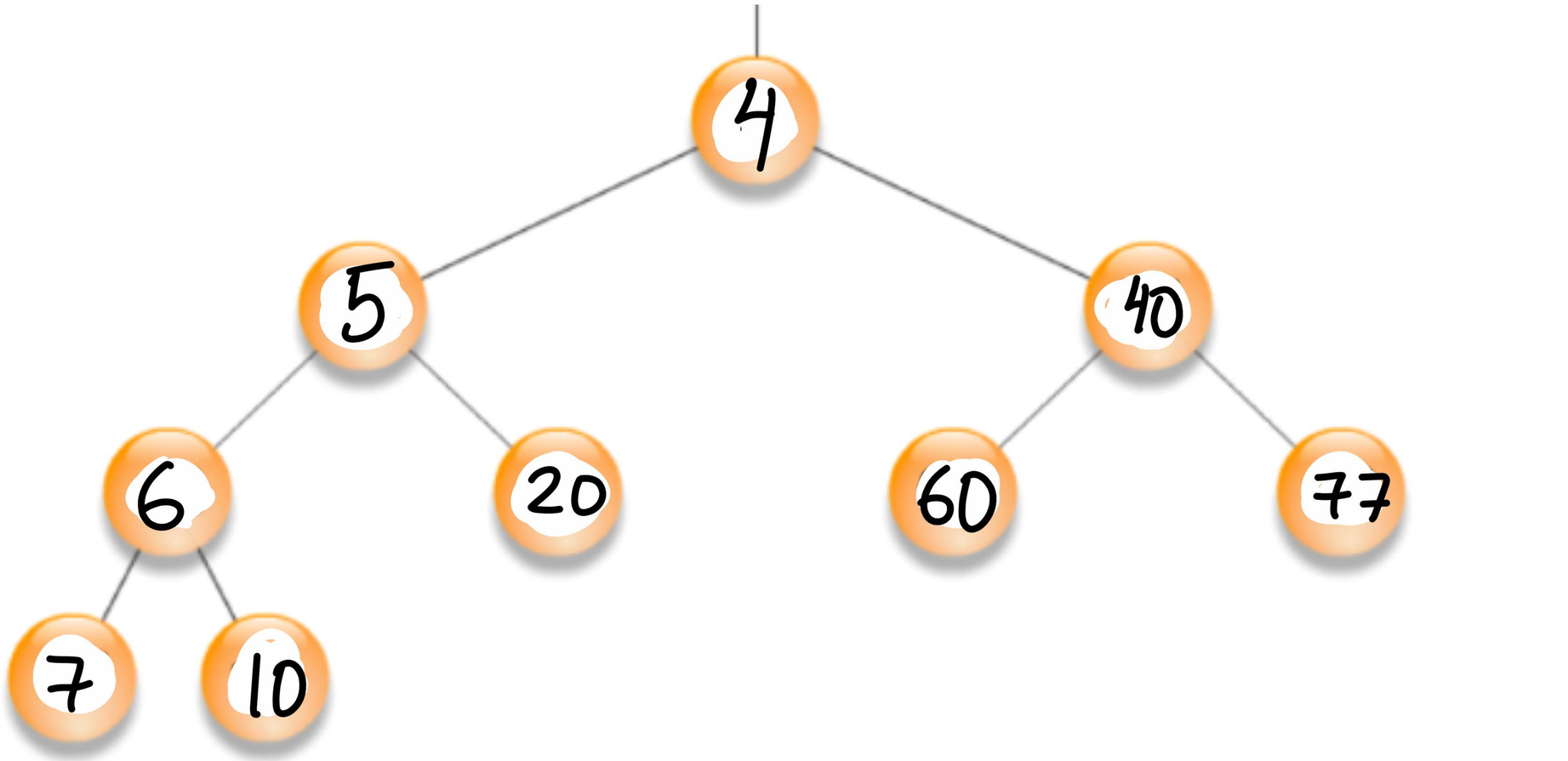
Problem: Given a binary tree, check if it is a heap.

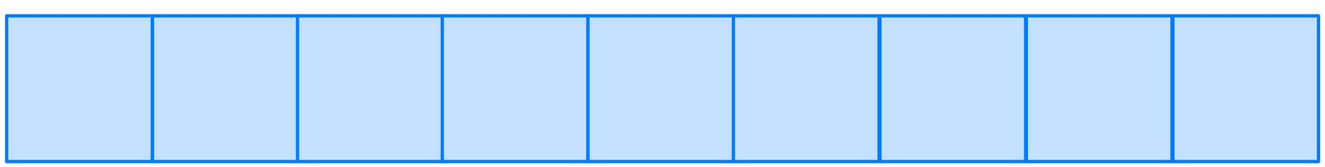



```
private:
   struct treeNode {
      treeNode(int d){da
      int data;
      treeNode * left;
      treeNode * right;
   };
   treeNode * root;
   int size;
```

Problem: Given a binary tree, check if it is a heap.

Part 1: Given the level order traversal in an array format, check if it is a heap.

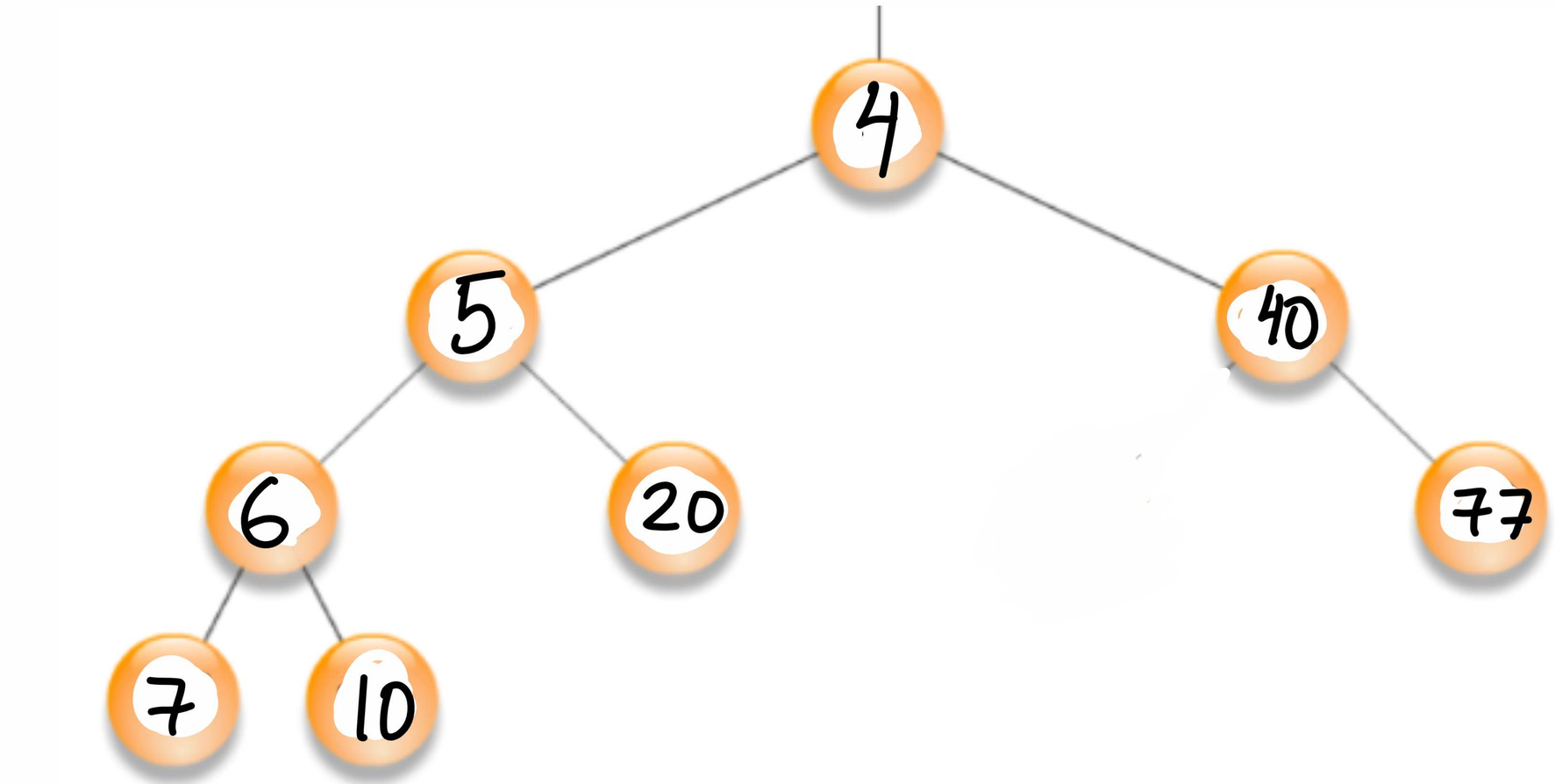


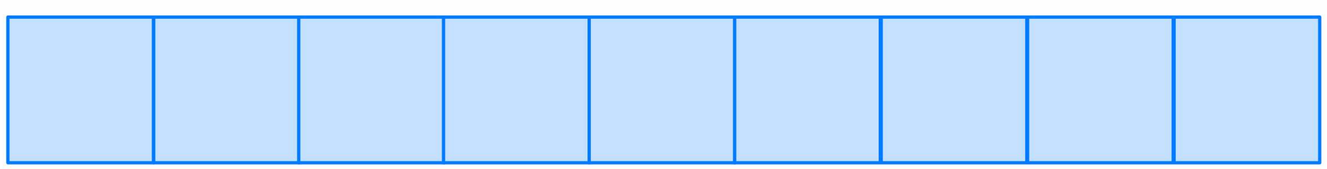


```
private:
   struct treeNode {
      treeNode(int d){da
      int data;
      treeNode * left;
      treeNode * right;
   };
   treeNode * root;
   int size;
```

Problem: Given a binary tree, check if it is a heap.

Part 2: Check if a given binary tree is a complete tree. Implement it as a member function of BinaryTree class that returns level order traversal.





```
private:
   struct treeNode {
     treeNode(int d){da
     int data;
     treeNode * left;
     treeNode * right;
   };
   treeNode * root;
   int size;
```

Fix Level Order Traversal

```
// Create an array from level traversal.
// The element at 0 is a bool for whether the tree is complete or not
int * BinaryTree::levelData() {
  int* arr = new int[size+1]; //new
  treeNode** nodes = new treeNode*[size];

  if (root != NULL) {
    nodes[0] = root;
    arr[1] = root->data; //new

    int seeNull = false; // new
    bool complete = true; // new
    int cEntry = 1;
    int cExit = 0;
    while (cExit < size) {
      if (nodes[cExit]->left != NULL){
        nodes[cEntry++] = nodes[cExit]->left;
        if(seeNull) complete = false; //new
      }
      else seeNull = true; //new

      if (nodes[cExit]->right != NULL) {
        nodes[cEntry++] = nodes[cExit]->right;
        if(seeNull) complete = false; //new
      }
      else seeNull = true; //new

      arr[cExit + 1] = nodes[cExit]->data; //new
      cExit++;
    }
    arr[0] = (int)complete; //new
  }
  delete[] nodes;
  return arr;
}
```