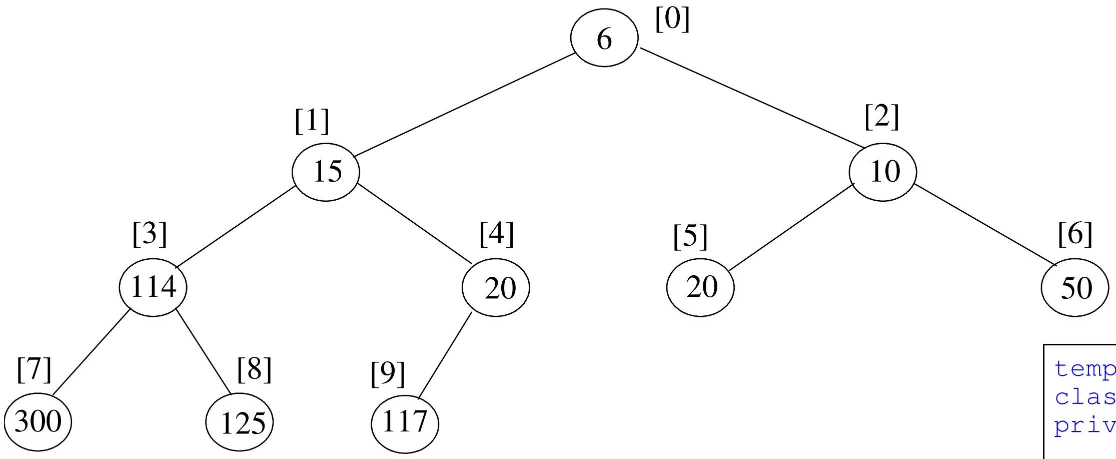
Data Structures (810:052) Lab 6 Name:__________________

<u>Objectives:</u> You will gain experience:

- implementing a min.BinaryHeap data structure
- using a menu-driven tester

Download the following file to your desktop: http://www.cs.uni.edu/~fienup/cs052s10/labs/lab6.zip
Extract this file to the Desktop by right-clicking on lab6.zip icon and selecting Extract All.

Part A: In lecture 11 we discussed implementing a min. heap using an array to store the items, but visualizing the items as a complete binary tree. Below is an example of a heap "viewed" an a complete binary tree. The array indexes are indicated in [ ]'s.



```
template <class T>
class BinaryHeap {
private:
        int maxSize;
        int numItems;
        T * heap;
        . . .
```

We even developed algorithms for insert and siftUp:

Algorithm for insert(T newItem)	**Algorithm for siftUp(int currentPosition)**

heap[numItems] = newItem
siftUp(numItems)
numItems++
	while currentPosition has not reached the root
	    calculate the parentIndex
	    if item at currentPosition < item at parentIndex then
	        exchange the two item
	        update the currentPosition
	    else
	        return since we are done sifting up

The `lab6.zip` file you downloaded and extracted contains a `binaryHeap` folder with a Visual Studio C++ project file: `binaryHeap.sln` inside. Double-click on it to open this project in Visual Studio. Your task is to implement the `insert` and `siftUp` functions which combine to insert a new item into the binary heap. The `main.cpp` file contains a menu-driven test program to test your functions.

After you have implemented and tested you insert and siftUp functions, raise your hand and demonstrate your program.

Part B: In lecture 11 we also discussed implementing the `delMin` operation which returns the root node and eliminates it from the tree by:

- copying the "last leaf item" in the tree (i.e., right most item in the array) to the root,
- "sifting this item down" the tree by repeatedly exchanging it with the smaller of its two children until it is in the correct spot by using a `siftDown(int currentPositon)` function.

We even developed algorithms for `delMin` and `siftDown`:

Algorithm for delMin	**Algorithm for siftDown(int currentPosition)**

temp = heap[0]
numItems--
heap[0] = heap[numItems]
siftDown(0)
return temp
	while true (infinite loop) do
	    if the currentPosition has NO children then
	        return
	
	    if the currentPosition has only a left child then
	        min. child is the left child
	    else if the left child < right child then
	        min. child is the left child
	    else
	        min. child is the right child
	
	    if the item at current Position > min. child then
	        exchange these two items
	        update the currentPosition
	    else
	        return since we are done sifting down
	end while

Your task is to implement the `delMin` and `siftDown` functions which combine to delete and return the smallest item from the binary heap. Use the same project from Part A. The `main.cpp` file contains a menu-driven test program to test your functions.

After you have implemented and tested you `delMin` and `siftDown` functions, raise your hand and demonstrate your program.