1. **Graphs**

   Vertex* findLargestTree(BasicGraph& graph) {
     int largestTreeSize = 0;
     Vertex* largestTreeRoot = NULL;

     for (Vertex* v : graph.getVertexSet()) {
       graph.resetData();

       int treeSize = findLargestTree(v, graph);
       if (treeSize > largestTreeSize) {
         largestTreeRoot = v;
         largestTreeSize = treeSize;
       }
     }

     return largestTreeRoot;
   }

   int findLargestTree(Vertex* v, BasicGraph& graph) {
     if (v == NULL) return 0;
     if (v->visited) return -1;

     v->visited = true;
     int treeSize = 1;
     for (Edge* e : v->edges) {
       int subTreeSize = findLargestTree(e->finish, graph);
       if (subTreeSize < 0) return -1;
       treeSize += subTreeSize;
     }

     return treeSize;
   }
2. Pointers and Linked Lists

struct listnode {
  int val;
  listnode * next;
};

bool contains(listnode* list, listnode* sub) {
  if (sub == NULL) {
    return true;
  } else if (list == NULL) {
    return false;
  }
  else if (list->val == sub->val) {
    return contains(list->next, sub->next);
  } else {
    return contains(list->next, sub);
  }
}

3. Recursion

Set<int> maxSumSubset (treenode* root) {
  if (root == NULL) return Set<int>();
  Set<int> childSet = maxSumSubset(root->left) +
  maxSumSubset(root->middle) +
  maxSumSubset(root->right);
  int childSum = 0;
  for (int i : childSet) {
    childSum += i;
  }
  if (childSum > root->key) {
    return childSet;
  } else {
    Set<int> us;
    us += root->key;
    return us;
  }
}
4. BSTs and Heaps

Diagram after inserting (25,2):

```
  25,2
 /    \
|      |
|      |
17,1   29,0
```

*This one is completed for you.*

Diagram after inserting (17,1):

```
  25,2
 /    \
|      |
|      |
17,1   29,0
```

Diagram after inserting (29,0):

```
  25,2
 /    \
|      |
|      |
17,1   29,0
```

Diagram after inserting (55,1):

```
  25,2
 /    \
|      |
|      |
17,1   29,0
     /    \   |
    |      |   55,1
       |
```

Diagram after inserting (45,7):

```
  25,2
 /    \
|      |
|      |
17,1   29,0
   /    \   |
  |      |   55,1
 /     |
|      |
```

Diagram after inserting (29,3):

```
  25,2
 /    \
|      |
|      |
17,1   29,0
     /    \   |
    |      |   55,1
 /     |
```

Diagram after inserting 25:

```
25
```

This one is completed for you.

Diagram after inserting 37:

```
25
    ▼
     ▼
  37   28
```

Diagram after inserting 28:

```
25
    ▼
     ▼
  37   28
```

Diagram after inserting 12:

```
12
    ▼
     ▼
  25   28
```

Diagram after inserting 30:

```
12
    ▼
     ▼
  25   28
```

Diagram after inserting 3:

```
3
    ▼
     ▼
  25   12
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```

```
25
    ▼
     ▼
  37   30
```
In the table below, indicate in the right-hand column the output produced by the statement in the left-hand column. If the statement produces more than one line of output, indicate the line breaks with slashes as in "x/y/z" to indicate three lines of output with "x" followed by "y" followed by "z".

If the statement does not compile, write "compiler error". If a statement would crash at runtime or cause unpredictable behavior, write "crash".

<table>
<thead>
<tr>
<th>Statement</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>var4-&gt;m3();</td>
<td>B 3 / B 1 / Y 3</td>
</tr>
<tr>
<td>var4-&gt;m1();</td>
<td>B 1</td>
</tr>
<tr>
<td>var4-&gt;m4();</td>
<td>COMPILER ERROR</td>
</tr>
<tr>
<td>var2-&gt;m3();</td>
<td>A 3</td>
</tr>
<tr>
<td>var2-&gt;m1();</td>
<td>P 1 / B 1</td>
</tr>
<tr>
<td>var2-&gt;m4();</td>
<td>A 4 / A 3</td>
</tr>
<tr>
<td>var1-&gt;m4();</td>
<td>COMPILER ERROR</td>
</tr>
<tr>
<td>var1-&gt;m3();</td>
<td>P 3</td>
</tr>
<tr>
<td>var1-&gt;m1();</td>
<td>P 1 / B 1</td>
</tr>
<tr>
<td>var5-&gt;m1();</td>
<td>P 1 / B 1</td>
</tr>
<tr>
<td>var5-&gt;m4();</td>
<td>Y 4</td>
</tr>
<tr>
<td>var5-&gt;m3();</td>
<td>P 3</td>
</tr>
<tr>
<td>((Yeats*) var4)-&gt;m3();</td>
<td>B 3 / B 1 / Y 3</td>
</tr>
<tr>
<td>((Yeats*) var4)-&gt;m4();</td>
<td>Y 4</td>
</tr>
<tr>
<td>((Angelou*) var3)-&gt;m4();</td>
<td>CRASH</td>
</tr>
<tr>
<td>((Byron*) var5)-&gt;m4();</td>
<td>COMPILER ERROR</td>
</tr>
<tr>
<td>((Plath*) var2)-&gt;m3();</td>
<td>A 3</td>
</tr>
<tr>
<td>((Angelou*) var2)-&gt;m3();</td>
<td>A 3</td>
</tr>
</tbody>
</table>
6. Algorithms

(a) \( O(\log n) \)

If \( n \) is even, we divide by two, otherwise we add one to \( n \). Clearly, Binky cannot add one to \( n \) twice in a row. There must therefore be at least as many steps where we divide \( n \) by 2 as there can be steps where we add one to \( n \). As \( n \) gets large, the number of times we have to divide \( n \) by two will be the factor that determines how quickly we approach zero or one. There can be at most \( \log(n) \) of those steps, so the running time is therefore \( O(\log n) \)

(b)

Does this strategy work? YES    NO  (circle)  Briefly explain why or why not:

If we are deleting the last cell in the list, \( \text{ptr} \rightarrow \text{next} \) is NULL. When try to access assign to \( *\text{(ptr)} \) in the next line, the right hand side will dereference NULL and crash.

(c)

<table>
<thead>
<tr>
<th>Worst-case big-O</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( O(n^2) )</td>
</tr>
<tr>
<td>2. ( O(n \log(n)) )</td>
</tr>
<tr>
<td>3. ( O(n^2) )</td>
</tr>
</tbody>
</table>