Question 1 – Open-addressed hashing (15 points)

In the questions below, you are given a partly-filled hash table and a sequence of operations. Draw (in the 3rd column) the hash table that results after performing all of the operations in the sequence in the order given. In addition, indicate (in the 4th column) the load factor of the resulting table after all of the operations are performed. For each operation, only the key is provided. You may ignore the corresponding values for this problem and insert only the keys into the hash table. Use open addressing and simple linear probing to resolve collisions. Use the value \texttt{TS} to indicate a tombstone. Tombstone values should be excluded from the load factor computation.

For each operation, use the following hash function:

```c
int hash_fn(char* key) {
    int hash_val = key[0] - 'a';
    return hash_val % 10;
}
```

This function simply returns the alphabet position of the first letter of the key, starting with 0, mod 10. In other words, keys starting with ‘a’ hash to index 0, keys starting with ‘b’ hash to index 1, and so forth. You may assume all lower-case keys. (5 points each)

<table>
<thead>
<tr>
<th>1.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dog</td>
<td>octopus</td>
<td></td>
<td>gorilla</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert(&quot;ibis&quot;);</td>
<td>insert(&quot;snake&quot;);</td>
<td>insert(&quot;narwhal&quot;);</td>
<td>remove(&quot;octopus&quot;);</td>
<td>insert(&quot;turtle&quot;);</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Load factor:
Question 1 – Open-addressed hashing (continued)

2.  

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>kale</td>
<td>lettuce</td>
<td>carrot</td>
<td></td>
<td></td>
<td>zucchini</td>
<td>garlic</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
insert("peas");
insert("radish");
remove("garlic");
insert("onion");
insert("yam");
remove("carrot");
```  

3.  

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>apple</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
insert("grape");
insert("tomato");
insert("pear");
remove("apple");
```  

Load factor:
Question 2 – Array representation of heaps (20 points)

In each question below, you are given a representation of a heap as an array. Each of these arrays may or may not exhibit the minimizing heap property. For each array, use the formulas we talked about in class for finding a node’s parent and children in an array representation of a heap to determine whether or not the array does in fact exhibit the minimizing heap property. If the array does in fact exhibit the minimizing heap property, check the “exhibits minimizing heap property” box for that array. Otherwise, circle the array element corresponding to the highest node in the tree where the minimizing heap property is violated as well as the child of that element that violates the heap property. For example, if you find that the element at index j violates the minimizing heap property with its parent at index i, you should circle both elements i and j. (4 points each)

1. 

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>18</td>
<td>22</td>
<td>26</td>
<td>32</td>
<td>36</td>
</tr>
</tbody>
</table>

Exhibits minimizing heap property

2. 

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>64</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

Exhibits minimizing heap property
Question 2 – Array representation of heaps (continued)

3.

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>18</td>
<td>14</td>
<td>32</td>
<td>22</td>
<td>20</td>
<td>16</td>
<td>18</td>
<td>30</td>
</tr>
</tbody>
</table>

Exhibits minimizing heap property

4.

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 32| 64| 36| 96| 72| 48| 42| 100|

Exhibits minimizing heap property

5.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>64</td>
<td>8</td>
<td>24</td>
<td>48</td>
<td>72</td>
</tr>
</tbody>
</table>

Exhibits minimizing heap property
Question 3 – Graph searches (25 points)

For each question below, perform the specified graph search on the given graph, listing out the nodes *in the order first visited by the search* in the spaces provided. Each node should be listed at most one time, though some nodes may not be listed at all, and some of the provided spaces may be empty.

1. Perform a *breadth-first search* on the following graph starting with node A. (10 points)

![Graph diagram]

**Nodes visited:**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Run *Dijkstra’s algorithm* on the graph below, starting with node A. Include in the listing of visited nodes the minimum total cost of the path from A to each visited node. (15 points)

Nodes visited, with minimum total cost from A:

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 4 – Graph representations (20 points)

For each question below, draw the requested representation of the following graph:

1. Draw an adjacency list representation of the graph above. (10 points)
2. Draw an *adjacency matrix* representation of the graph above. The rows of the matrix should correspond to the *tails* of edges (i.e. the vertex the edge is leaving), and the columns should correspond to the *heads* of edges (i.e. the vertex the edge is entering). (10 points)
Question 5 – Heap operations (25 points)

For each question below, draw the heap that results from performing the specified operation on the given heap. *Make sure to include any adjustments to ensure that the heap property is maintained.* For each question, the heap is a minimizing heap. (5 points each)

1. Add the value 16 to the heap below:

```
        8
       / \  
      10  12
     /  \
    20  16
```

2. Add the value 6 to the heap below:

```
        2
       / \  
      8   16
     /     /  \
    10   28  18  20
    /     /     /     \
   32   32   32   32
```
Question 5 – Heap operations (continued)

3. Add the value 14 to the heap below:

```
  16
   /\   \\
  18   20
   /\   /\
  22 24 26 28
  /\  /\  /\  /\  \
 30 32 34 36 38
```

4. Remove the minimum value from the heap below:

```
  8
 /\   /
10 16
 /\  /\  /\  /\  \
12 20 18
```

5. Remove the minimum value from the heap below:

```
  6
 /\   /
12   8
 /\  /\  /\
20 26 28
```

Question 6 – AVL Trees (25 points)

For each question below, draw the AVL tree that results from performing the specified operation on the given AVL tree, including all replacement and rebalancing. (5 points each)

1. Remove the value 16 from the AVL tree below:

   22
   /   
  15   48
 /     /
8     24
 /     /  
4     20  28
 /     /   /
12    28  36  40
 /     /     /
4      36    44  52
     /       /     /
    20     44    52  60

2. Remove the value 10 to the AVL tree below:

   8
   / 
  6  10
 /  
4
Question 6 – AVL Trees (continued)

3. Remove the value 48 from the AVL tree below:

   24
   /   
  16    36
   /     /
  8    20   48
       /   /
      22   26

4. Add the value 64 to the AVL tree below:

   32
   /   
  16    48
   /     /
  24    40   56
      /     /
     22    60

5. Add the value 16 to the AVL tree below:

   20
   /   
  12    96
     /     
    4
Question 7 – Binary tree traversals (20 points)

For each question below, write out the values of the nodes of the given binary tree in the order in which they would be processed by the specified traversal. Write the values in the spaces provided. (5 points each)

1. Level-order:

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

2. Post-order:

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

3. Pre-order:

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

4. In-order:

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>