1. Exercise 9 (page 599)

(a) The highest-priority element is the one with the smallest time stamp

(b) Enqueue

Assign the next largest time stamp to the new element
Put new element in the queue (position is unimportant)

```cpp
template <typenameItemType>
voidQueType<ItemType>::Enqueue(ItemType newItem)
{
    QueNode<ItemType>tmp;
    tmp.data = newItem;
    tmp.priority = nextTimestamp;
    nextTimestamp++;
    PQType<ItemType>::Enqueue(tmp);
}
```

Dequeue

Find the element with the smallest time stamp
Assign this element to item (to be returned)
Remove this element from the queue

```cpp
template <typenameItemType>
voidQueType<ItemType>::Dequeue(ItemType item)
{
    QueNode<ItemType>tmp;
    PQType<ItemType>::Dequeue(tmp);
    item = tmp.data;
}
```

(c) Enqueue, like those in Chapter 5, has $O(1)$. Dequeue has $O(N)$, as compared with the $O(1)$ operations developed in Chapter 5. If the priority queue is implemented using a heap with the smallest value the highest priority, Enqueue has $O(1)$ and Dequeue has $O(\log_2 N)$. 
2. Exercise 11, 12, 13 (pages 600, 601) To get credit, demonstrate DFS and BFS like in the examples we did in class.

Table 1: Employee Graph adjacency matrix

<table>
<thead>
<tr>
<th></th>
<th>Brent</th>
<th>Darlene</th>
<th>Fran</th>
<th>Fred</th>
<th>Jean</th>
<th>John</th>
<th>Lance</th>
<th>Miss</th>
<th>Sander</th>
<th>Susan</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Iter:

BFS

<table>
<thead>
<tr>
<th>Iter</th>
<th>Init: E9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D9 M9 E1,4,5,8</td>
</tr>
<tr>
<td>2</td>
<td>D1 M1 E7</td>
</tr>
<tr>
<td>3</td>
<td>D4 M4 E6</td>
</tr>
<tr>
<td>4</td>
<td>D5 M5 E2</td>
</tr>
<tr>
<td>5</td>
<td>D8 M8 E2</td>
</tr>
<tr>
<td>6</td>
<td>D7 M7 EØ</td>
</tr>
<tr>
<td>7</td>
<td>D6 Stop</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Iter:

DFS

<table>
<thead>
<tr>
<th>Iter</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pop 9 Mark 9 Push 1, 4, 5, 8</td>
<td>Pop 8 Mark 8 Push 2</td>
<td>Pop 2 Mark 2 Push 5, 6</td>
<td>Pop 6 Done</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
3. Exercise 19 (page 599)

\[ V(\text{StateGraph}) = \{\text{Oregon, Alaska, Texas, Hawaii, Vermont, New York, California} \} \]
\[ E(\text{StateGraph}) = \{(\text{Alaska, Oregon}), (\text{Hawaii, Alaska}), (\text{Hawaii, Texas}), (\text{Texas, Hawaii}), (\text{Hawaii, California}), (\text{Hawaii, New York}), (\text{Texas, Vermont}), (\text{Vermont, California}), (\text{Vermont, Alaska})\} \]

Note: The order of the elements in the sets is not important.

4. Exercise 24 (page 603)

Deleting a vertex is more complicated than deleting an edge for two reasons. First, in addition to removing the vertex from the set of vertices, we must also remove the edges to all its adjacent vertices in the set of edges. Second, we must decide what to do about the now unused vertex number. The best solution is to keep a list of returned vertex numbers and assign new numbers from there first.

```cpp
template <typename ItemType>
void GraphType<ItemType>::DeleteVertex( ItemType vert )
{
    int i=0, j;
    while( i<numVertices && (not (vert==vertices[i])))
    {
        i++;
    }
    if ( i!=numVertices )
    {
        vertices[i] = vertices[numVertices-1];
        for (j=0; j<maxVertices-1; j++)
        {
            edges[i][j] = edges[maxVertices-1][j];
            edges[j][i] = edges[j][maxVertices-1];
        }
        numVertices--;
    }
}
```

5.
Dijkstra's Algorithm
Bellman Ford’s Algorithm
- 4 passes (since there are 5 vertices)
- One more pass to check for negative cycles
- Note that results depend on the order of relaxing the edges.