EECS 311 Data Structures  
Final Exam  
Don’t Panic!

1. [10 pts] Show how Quicksort would sort the array below. Pick the pivot with median-of-three, using integer division to get the center. Don’t sort the three, just swap the pivot with the last element. Be very clear about what goes where in each partitioning phase, e.g., write something like the following for each partitioning:

   sorting from __ to __, pivot __ swaps with __  
   quicksort pass swaps __ with __, __ with __, __ with __, ...  
   result = ...

Circle the pivot in its final location. When a partition is 3 elements or fewer, just indicate the swaps needed, if any, to directly sort it.

<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>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>
<td>10</td>
</tr>
<tr>
<td>23</td>
<td>51</td>
<td>4</td>
<td>18</td>
<td>8</td>
<td>72</td>
<td>31</td>
<td>42</td>
<td>17</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

Comment [CKR1]: Average 9.2, Median 10

Comment [CKR2]: Most got this. Common mistakes:  
- Including the pivot in later sorts  
- Picking the middle element, not the median of three  
- Not swapping pivot with last element first.
2. [10 pts] In the table below, show the values that Prim’s algorithm would find while creating a minimum spanning tree for the graph below starting from vertex D. Initial values are shown. Under best edge weight and best edge vertex put the sequence of weights and vertices of the best edge leading to the given vertex, in the order found. Under when done put 1 for the 1st edge finished, 2 for the 2nd, and so on. Draw a line on the graph edges used in the final MST. Is this unique? Why or why not?

<table>
<thead>
<tr>
<th>vertex</th>
<th>when done</th>
<th>best edge weight</th>
<th>best edge vertex</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>(\infty) 6 3</td>
<td>- D F</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>(\infty) 4 2</td>
<td>- E A</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>(\infty) 10 6</td>
<td>- B F</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>(\infty) 5</td>
<td>- D</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>(\infty) 11 3</td>
<td>- H I</td>
</tr>
<tr>
<td>G</td>
<td>8</td>
<td>(\infty) 6 3</td>
<td>- J</td>
</tr>
<tr>
<td>H</td>
<td>3</td>
<td>(\infty) 9 2</td>
<td>- D E</td>
</tr>
<tr>
<td>I</td>
<td>2</td>
<td>(\infty) 1</td>
<td>- E</td>
</tr>
<tr>
<td>J</td>
<td>7</td>
<td>(\infty) 3</td>
<td>- I</td>
</tr>
</tbody>
</table>

Unique answer. All edges available as next choices were eventually selected anyway.

Comment [CKR3]: Average 8.5 Median 9

Comment [CKR4]: Most common mistakes:
- Using path weights not edge weights
- Not choosing cheapest edge at each point
- Not updating cheapest available edges
- Assuming multiple choices sufficient for multiple answers

Comment [CKR5]: Many possible orderings, but
- E, I and H must be start
- B must immediately follow A
- C must be last
3. [10 pts] In the table below show the values Dijkstra’s algorithm would generate to find the shortest path from A to G. Under **best path** put the sequence of path distances found, in order, for each vertex. Under **best vertex** put the path vertices found, in order. Under **when done** put 1 for the first vertex that is finished, 2 for the 2nd vertex finished, etc.

<table>
<thead>
<tr>
<th>vertex</th>
<th>when known</th>
<th>best path</th>
<th>best vertex</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>∞ 2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>∞ 3</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>∞ 4 6</td>
<td>A B</td>
</tr>
<tr>
<td>E</td>
<td>∞ 10</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>F</td>
<td>∞ 12 8</td>
<td></td>
<td>C D</td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td>∞ 7</td>
<td>D</td>
</tr>
</tbody>
</table>

**Comment [CKR6]:** Average 8.1  
Median 8

**Comment [CKR7]:** Most common mistakes:  
- Using edge weights not path weights  
- Not picking cheapest next path weight  
- Not updating cheapest available path  
- Not stopping when G done (no points deducted)
4. [10 pts] Use the dynamic programming approach to sequence alignment for the problem below. Matching letters score +4 points, mismatching letters score -3, and a gap in either sequence scores -2. Draw small arrows from each cell X to the previous cell(s) whose score leads to the one in X. Show an optimal alignment that follows from this table. Is it unique? Why or why not?

Sequence 1: GTATCGA
Sequence 2: GAATCGAA

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>-6</td>
<td>-8</td>
<td>-10</td>
<td>-12</td>
</tr>
<tr>
<td>A</td>
<td>-4</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>-6</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>-8</td>
<td>-2</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>G</td>
<td>-10</td>
<td>-4</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>A</td>
<td>-12</td>
<td>-6</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>A</td>
<td>-14</td>
<td>-8</td>
<td>-2</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>14</td>
</tr>
</tbody>
</table>

Two solutions: GTATCGA – A and GTATCGA - G – A T C G A A and G – A T C G A A

Comment [CKR8]: Average 8.4
Median 9

Comment [CKR9]: Most common mistakes:
• Wrong deltas, e.g., -3 going horizontally
• Not picking best transition, e.g., going horizontal when diagonal better
• Not showing a resulting optimal alignment
• Showing only half an alignment
• Saying alignment was unique
The dynamic programming formula for the maximum sum $M(A, j)$ of a contiguous subsequence ending on position $j$ of an integer array $A$ is:

$$M(A, j) = \max(M(A, j - 1) + A[j], A[j])$$

Given the C++ template class below (from class)

```cpp
template < class Arg, class Result >
class UnaryMemoFunction {
private:
    typedef std::map< Arg, Result > Cache;
    Cache cache;
public:
    Result operator() ( Arg a ) { return memo( a ); }  
protected:
    Result memo( Arg a ) {
        typename Cache::const_iterator it = cache.find( a );
        return it == cache.end() ? cache[ a ] = call( a ): it->second; }
    virtual Result call( Arg n ) = 0;
};
```

fill in the code below to make the example test and ones like it pass:

```cpp
class MaxSum : public UnaryMemoFunction<int, int> {
public:
    MaxSum( std::vector<int> a ) : a( a ) {}
protected:
    int call( int j )
    {
        return j == -1 ? 0 :
            std::max( memo( j - 1 ) + a[ j ], a[ j ] );
    }
private:
    std::vector<int> a;
};

TEST(MaxSum)
{
    int a1[] = { -2, 11, -4, 13, -5, -2 }; 
    std::vector<int> v( a1, a1 + 6 );
    // make a function object that sums over v
    MaxSum maxSummer( v );
    //find largest subsequence sum in v 
    int result = -1;
    for ( unsigned int i = 0; i < v.size(); ++i)
        result = std::max( result, maxSummer( i ) );
    CHECK_EQUAL( 20, result );
}
```