December 10, 2009 Name ______________________________

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 = ...

Comment [CKR1]: Average9.2, Median 10

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.

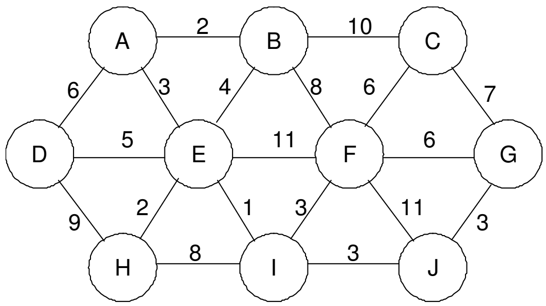
0	1	2	3	4	5	6	7	8	9	10
23	**51**	**4**	**18**	**8**	**72**	**31**	**42**	**17**	**9**	**5**

sorting from 0 to 10, median of 23, 72 and 5 is 23, swap with last element 5
then swap 51 with 9, 72 with 17
then swap pivot 23 with 31 to put pivot in final location 6
result = 5 9 4 18 8 17 [23] 42 72 51 31
sorting from 0 to5, median of 5, 4 and 17 is 5, swap with last element 17
then swap 17 with 4
then swap pivot 5 with 9 to put pivot in final location 1
result = 4 [5] 9 8 18 17 [23] 42 72 51 31
sorting from 2 to 5, median of 4, 18 and 17 is 17
no pivot swap needed, no quicksort swaps needed
swap pivot 17 with 18 to put pivot in final location 4
result =4 [5] 9 8 [17] 18 [23] 42 72 51 31
sort 9 and 8 directly
result =[4 5 8 9 17 18 23] 42 72 51 31
sorting from 7 to 10, median of 42, 72and 31 is 42, swap with last element 31
then swap 72 with 51
swap pivot 42 with 51 to put pivot in final location 7
result =[4 5 8 9 17 18 23] 31 [42] 72 51
sort 72 and 51 directly
result =4 5 8 9 17 18 23 31 42 51 72

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 cells 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?

Comment [CKR3]: Average 8.5 Median 9



vertex	when done	best edge weight	best edge vertex
A	4	∞ ~~6~~ 3	- ~~D~~ E
B	5	∞ ~~4~~ 2	- ~~E~~ A
C	9	∞ ~~10~~ 6	- ~~B~~ F
D	0	0	-
E	1	∞ 5	- D
F	6	∞ ~~11~~ 3	- ~~E~~ I
G	8	∞ ~~6~~ 3	- ~~F~~ J
H	3	∞ ~~9~~ 2	- ~~D~~ E
I	2	∞ 1	- E
J	7	∞ 3	- I

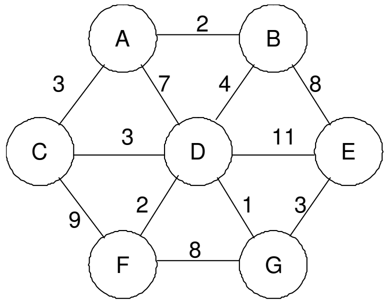
Unique answer. All edges available as next choices were eventually selected anyway.

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.

Comment [CKR6]: Average 8.1 Median 8



vertex	when known	best path	best vertex
A	0	0	-
B	1	∞ 2	A
C	2	∞ 3	A
D	3	∞ ~~7~~ 6	~~A~~ B
E		∞ 10	B
F		∞ ~~12~~ 8	~~C~~ D
G	4	∞ 7	D

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?

Comment [CKR8]: Average 8.4 Median 9

Sequence 1: G T A T C G A
Sequence 2: G A T C G A A

		G	T	A	T	C	G	A
	0	-2	-4	-6	-8	-10	-12	-14
G	-2	4	2	0	-2	-4	-6	-8
A	-4	2	1	6	4	2	0	-2
T	-6	0	6	4	10	8	6	4
C	-8	-2	4	3	8	14	12	10
G	-10	-4	2	1	6	12	18	16
A	-12	-6	0	6	4	10	16	22
A	-14	-8	-2	4	3	8	14	20

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

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

5. [10 pts] 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:

Comment [CKR10]: Average 5.0
Median 4

M(A, j) = max(M(A, j – 1) + A[j], A[j])

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

```
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:

```
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;
};
```

Comment [CKR11]: Comon mistakes:

- Not including the superclass
- Including superclass as data member
- Naming function maxSum() instead of call() and memo()
- No base case
- Writing a loop

```
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 );
}
```