2016 Exam

1a)

```
int is_prime(int num) {
        int i;
        // Take absolute value of number
        if(num < 0){
                num = -num;
        }

        // Only need to check up to square root as any values beyond square root are
        // paired with a factor on the other side of the multiplication.
        for( i = 2; i < pow(num,0.5) + 1; i++ ){
                if(num % i == 0){
                        // Not a prime if other factors than 1 and num.
                        return 0;
                }
        }

        prime(num);
        return 1;
}
```

1b)

```
// Note: One student has pointed out that there is a much better O(n) algorithm
//  which can be used to do this, you wouldn't be expected to come up with the
//  better algorithm under exam conditions though you're certainly welcome to
//  try if you've finished the rest of the questions.
int max_sum_inner(int *vector, int len){
        int sum = 0;
        int i;
        for( i = 0; i < len; i++ ){
                // Only sum primes in list.
                if(is_prime(vector[i]){
                        sum += vector[i];
                }
        }
        return sum;
}

int max_sum(int *vector, int len){
        // -∞
        int curmax = INT_MIN;
        // -∞ in case no change in sum for loops.
        int sum = INT_MIN;
        int i,j;

        // Loop through all starting positions
        for( i = 0; i < len; i++ ){
                // If first number not prime, next run will be the same result, so skip
                // this one.
                // Would be efficient to remove these values from the array completely,
                // so we only check the primes once, but for the puzzle we're solving,
                // we haven't been told whether we can modify the original array, so
                // this is about the best we can do without creating any auxiliary data
                // structures.
                if( ! is_prime(vector[i]) ){
                        continue;
                }
                // Shrink array from other side to get all subarrays starting from i.
                for( j = len - i; j > 0; j-- ){
                        // If final number not prime, next run will be the same result, so
                        // skip this one.
                        if( ! is_prime(vector[i+j - 1]) ){
                                continue;
                        }
                        sum = max_sum_inner(vector + i, j);
                }
                if (sum > currmax){
                        currmax = sum;
                }
        }
        return currmax;
}
```

1c)

```
#define LENGTH 6

int main(){
        int *foo;
        int i;
        int sum;
        // Assume LENGTH >= 6
        foo = (int *) malloc(sizeof(int)*LENGTH);
        assert(foo);

        foo[0] =  1;
        foo[1] =  1;
        foo[2] = -5;
        foo[3] =  3;
        foo[4] = -3;
        foo[5] =  2;

        sum = max_sum(foo, 6);
        if( sum == INT_MIN ){
                // Assume we don't almost underflow and that this is -∞
                printf("No primes summed\n");
        } else {
                printf("%d\n",sum);
        }

        free(foo);
        return 0;
}
```

2a) O(n)

2b) O(n^2)

2c) O(log n)

2d) O(1)

2e) O(n)

2f) O(1)

3ai)

```
Merge (10,20)
  {10,20} {30} {40} {50} {60}
    10     30   40   50   60
  20

Merge (50,60)
  {10,20} {30} {40} {50,60}
    10     30   40     50
  20                 60

Merge (30, 40)
  {10,20} {30,40} {50,60}
    10     30        50
  20     40        60
Merge (30, 60)
{10,20} {50,60,30,40}
    10             30
  20             50  40
               60


Merge (30, 10)
          10
        20    30
            50  40
          60
(10)(20)(30)(40)(50)(60)
[ 0,  0,  0,  2,  2,  4]
```

3aii) O(1), in naïve array-based union-find, we select a representative and update all items in the array to have the new representative's value, hence checking if two sets are the same set is simply as expensive as checking whether they share the same representative.

3b)

```
makepq → O(1)
┌-while → |V|
| u = deletemin → O(|V|) [Deletion is θ(log n) worst case for balanced bst, but
|                         because we're using an in-order traversal, it's O(n)]
```

```
| for(edges of u) → edges of u
|       update → O(log(|V|)) → deletion + add worst case (balanced bst)
| └-------------------------------------------------------------┘
└  sum of all edges of u for all u * O(log(|V|)) = |E|log|V| across all runs
Total: O(1 + |V||V| + |E|log|V|) = O(|V|²)
```

3c) Assuming head node is 0, nodes are at 3n + 1, 3n + 2, 3n + 3

4ai) Kruskal's O((|V| + |E|)log|E|), performs better when less edges.

4aii) Prim's O((|V| + |E|)log|V|), performs better when less vertices.

4bi) false, unconnected (or weakly connected) components may be missed.

4bii) false, BFS doesn't consider path weights, so needs unweighted.

4biii) false, only true if heuristic is admissible.

4biv) false, need polynomial reduction.

4c) sum v=1 to V out_degree(v) = sum v=1 to V in_degree(v)

4d) negative loop - in this example, dijkstra might find a,c,b,d as the shortest path, while an infinite path traversing acba would be shorter. This is because Dijkstra's algorithm assumes the path is not made shorter by following later negative cycles.

```
   a
-9/ \ -16
 /   \
/     \
b ----- c
\ -12 / -14
 \   /
-19\ /
   d
```

5a)

```
1 2 3 4 5 6 7
|_| |_| |_| |
|_____| |___|
|___________|
```

1 call for mergesort, as bottom-up mergesort is handled using a queue. We make 7 (or 6) calls to merge.

5b) This question asks you to correct the array in a conceptually quicksort way, but doesn't want you to change the order where that is not constrained. So we get:

```
   -13-
       -4- 5 2 3 1      43 22
                  -2- 1 3   5
                         -1- 3 5
                                -3--5- -43-
                                           -22-
13, 4, 2, 1, 3, 5, 43, 22
```

5c) Using DFS / Post-order traversal with insertion at head of linked list.

```
1 -> 2 -> 4 -> 6
|    ├-->[6 - already visited]
|    └--> 3 -> 5
|              ├-->[4 - already visted]
|              └-->[6 - already visited]
└--> [4 - already visited]

Topological sort result: 1->2->3->5->4->6
```

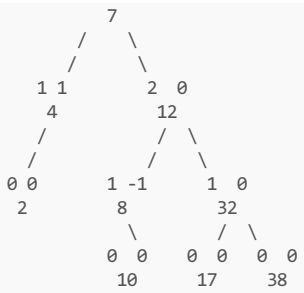
5di) `[ 79 1 89 23 43 44 - - - 99 ]`

5dii) Yes, chaining would take 3 operations (99,79,89) instead of 4 (99,79,1,89) or potentially even only 1 operation if we insert at the head of the list.

5e) Right rotate on 12-32, as the right subtree is left-leaning so needs to be corrected to the right.

Final Tree:

```
           3 1
```


7
1 1
2 0
4
12
0 0
1 -1
1 0
2
8
32
0 0
0 0
0 0
10
17
38