CS 106X Autumn 2016 Midterm Exam
ANSWER KEY

1.

```
a)
output:  c=18 z=18
return:  18

b)
output:  c=33 c=16 c=14 x=16 y=14 x=33 y=30 z=63
return:  63
```

2.

```
31 530 0x777700 499
32 32 0xcc00 0
2 33 0xbb00 31
2 33 32 0x777700 0xbb00
0xaa00 0xbb00 0xcc00 530 33
```

3.

i. O(N log N)
ii. O(N)
iii. O(N)
iv. O(N^2)

4.

Every code-writing problem can be solved in multiple ways.

```
void combineHelper(Queue<int>& q1, Queue<int>& q2, Queue<int>& q3) {
    if (q1.isEmpty() && q2.isEmpty()) {
        return;
    }

    if (q2.isEmpty()) {
        q3.enqueue(q1.dequeue());
    } else if (q1.isEmpty()) {
        q3.enqueue(q2.dequeue());
    } else {
        if (q2.peek() < q1.peek()) {
            q3.enqueue(q2.dequeue());
        } else {
            q3.enqueue(q1.dequeue());
        }
    }

    combineHelper(q1, q2, q3);
}

Queue<int> combineQueues(Queue<int>& q1, Queue<int>& q2) {
    Queue<int> q3;
    combineHelper(q1, q2, q3);
    return q3;
}
```

5.

```
void findBestFriends(ifstream& file) {
    Map<Vector<string>, int> convoCounts;

    Set<string> peopleInConvo;
    string line;
    int largest = 0;
    while (getline(file, line)) {
        if (line != "---") {
            int nameDelim = line.find(":");
            string name = line.substr(0, nameDelim);
            peopleInConvo.add(name);
        } else {
            Set<string> others = peopleInConvo;
            for (const string& person : peopleInConvo) {
                for (const string& partner : others) {
                    if (person != partner) {
                        Vector<string> pair { person, partner };
                        largest = max(largest, ++convoCounts[pair]);
                    }
                }
                others.remove(person);
            }
            peopleInConvo.clear();
        }
    }

    for (Vector<string> pair : convoCounts.keys()) {
        if (convoCounts[pair] == largest) {
            cout << pair[0] << " and " << pair[1] << endl;
        }
    }
}
```

6.

```
void coverHelper(Set<int>& universe, Set<Set<int>>& sets,
                 Set<Set<int>>& best, Set<Set<int>>& chosen,
                 Set<int> chosenElements,
                 int chosenTotal, int& bestTotal) {
    if (chosen.size() > best.size()) {
        return;
    } else if (chosenElements == universe) {
        if (chosen.size() < best.size() || (chosen.size() == best.size() &&
                (bestTotal < 0 || chosenTotal < bestTotal))) {
            best = chosen;
            bestTotal = chosenTotal;
        }
    } else if (!sets.isEmpty()) {
        Set<int> first = sets.first();
        sets.remove(first);

        // try without this set
        coverHelper(universe, sets, best, chosen, chosenElements, chosenTotal, bestTotal);

        // try with this set
        chosen.add(first);
        coverHelper(universe, sets, best, chosen, chosenElements + first,
                    chosenTotal + first.size(), bestTotal);
        chosen.remove(first);

        sets.add(first);
    }
}

Set<Set<int>> findSetCover(Set<int>& universe, Set<Set<int> >& sets) {
    Set<Set<int>> best = sets;
    Set<Set<int>> chosen;
    Set<int> chosenElements;
    int bestTotal = -1;
    coverHelper(universe, sets, best, chosen, chosenElements, /* chosenTotal */ 0, bestTotal);
    return best;
}
```

7.

```
ListNode* temp = list1->next;        // temp -> 2
list1->next = list1->next->next;     // 1 -> 3
delete temp;                         // delete 2
temp = list2;                        // temp -> 4
list2 = list2->next->next;           // list2 -> 6
list2->next = list1->next;           // 6 -> 3
list1->next = temp;                  // 1 -> 4
temp = temp->next;                   // temp -> 5
temp->next = list1;                  // 5 -> 1
list1 = temp;                        // list1 -> 5
list1->next->next->next = nullptr;   // 4 /
```