CS 106B Midterm

Rules:

- This exam is to be completed by each student individually, with no assistance from other students.
- You have 2 hours to complete this exam.
- This test is closed-book, closed-notes. You may only have one 8.5x11” double-sided sheet of notes and the reference sheet (which we will distribute).
- You may not use any computing devices, including calculators, cell phones, iPads, or music players.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- On code-writing problems, you do not need to write a complete program, nor #include statements. Write only the code (function, etc.) specified in the problem statement.
- Do not abbreviate code, such as writing ditto (""" or dot-dot-dot marks (...). Pseudo-code will be given no credit.
- Follow the Stanford Honor Code on this exam and correct/report anyone who does not do so.
- You must turn your WiFi off during the exam. Once you are done with the test, you can turn WiFi on and use your 2-step device to submit.
- You will only be able to submit the exam once, so do not submit until you are done.

Good luck!
Problem One: Big-Oh (6 Points)

Give a tight bound of the nearest runtime complexity class for each of the following code fragments in Big-Oh notation, in terms of variable N. (In other words, the algorithm’s runtime growth rate as N grows.) Write your answers in the designated space at the bottom of the page.

a)
```
int x = 0;
for (int i = 0; i < 5000; i++) {
    for (int j = 0; j < N; j++) {
        x++;
    }
}
cout << x << endl;
```

b)
```
Vector<int> vec;
for (int i = 0; i < 2*N; i++) {
    vec.add(i);
}
cout << vec.toString() << endl;
```

c)
Be sure to read the comment of the below function before determining the runtime complexity class.
```
/*@ 
* s1 contains N elements.
* words contains M words.
* Your answer might contain both M and N.
*/
void checkWords(Set<string>& s1, Vector<string>& words) {
    for (int i = 0; i < words.size(); i++) {
        if (s1.contains(words[i])) {
            cout << words[i] << " is included." << endl;
        }
    }
}
```

Your answers:

a) O(   )
b) O(   )
c) O(   )
Problem Two: ADTs (6 Points)

Consider the following code, written using Stanford library ADT implementations:

```cpp
void collectionMystery(Vector<int>& vec) {
    Stack<int> s;
    Queue<int> q;
    for (int i = 0; i < vec.size(); i++) {
        if (vec[i] % 2 == 1) {
            s.push(vec[i]);
        }
    }
    cout << s << endl;
    while (!s.isEmpty()) {
        q.enqueue(s.pop());
        if (!vec.isEmpty()) {
            q.enqueue(vec[0]);
            vec.remove(0);
        }
    }
    cout << q << endl;
}
```

Now write the output of the above code, given the following inputs. In the inputs shown, the leftmost entry is the first item in the vector, and the rightmost entry is the last item in the vector. The same is true for queues and stacks, the leftmost entry is the first item added, the rightmost entry is the last item added. Your output should adhere to this convention.

a) vec = {20, 2, 1, 47}
b) vec = {3, 6, 8, 19, 21, 1}

Output for part a:

Output for part b:
Problem Three: Recursion (6 Points)

For each call to the recursive function below, write the output it would produce as it would appear on the console.

```cpp
void mystery(int x, int y) {
    cout << x << " " << y << " ";
    if (x < 0) {
        mystery(-x, y);
    } else if (y < 0) {
        mystery(x, -y);
    } else if (x < 10) {
        cout << "DONE ";
    } else {
        cout << "{ 
        mystery(x / 10 + y % 10, x % 10);
        cout << x + y << "] " ;
    }
}
```

a) `mystery(61, 22);`

b) `mystery(1045, -432);`

Output for part a:

Output for part b:
Problem Four: ADTs (Collections) (11 Points)

It is the middle of democratic primary season, but oh no! The entire New York Times staff got food poisoning from some egg salad sandwiches. They contact you to help them maintain coverage of who is winning in the polls. Your job (which requires zero political knowledge) is to read in a file and then print out certain statistics about that file. The file will be in the below format:

State name
Candidate Name:Number of Votes
Candidate Name:Number of Votes
Candidate Name:Number of Votes
Candidate Name:Number of Votes
Candidate Name:Number of Votes
Next State name
Candidate Name:Number of Votes
Candidate Name:Number of Votes
Candidate Name:Number of Votes
Candidate Name:Number of Votes
Candidate Name:Number of Votes

So an example file could be:

California
Biden:24
Harris:23
Sanders:16
Warren:19
Iowa
Biden:24
Harris:16
Sanders:19
Warren:17
Buttigieg:7
Arizona
Sanders:42

Note that the polls are incomplete so each state may have different numbers of candidates. We make no guarantee on the number of candidates per state. There can be any number of states in the file, though we guarantee at least one state and one candidate. We do promise that the file is well formatted. You do not need to do any error checking on the file. You also may assume that the passed in filename is a valid name of a file. State names will not have colons (":") in their names.

You must read in this file and then print out the overall frontrunner, as well as the number of votes in each state for that candidate. For this file it would look like this:

Sanders is winning!
Arizona: 42
California: 16
Iowa: 19

You can assume there will be no ties for the overall leader. If a candidate is not listed in a state, it is the same as if that candidate received zero votes in that state. You do not need to print this state out if that candidate ends up being the overall leader, though you may if you want to. You may print the states out in any order.
Constraints: For full credit, obey the following restrictions:

- Do not use structs or classes
- Do not rewind the stream or read the file more than once.
- Do not declare any global variables.
- You can declare as many local variables as you like.

```c
void printStatistics(string filename) {
```
**Problem Five: Recursion (10 Points)**

Write a function `bool isAscending(int n)` that takes in an integer and returns a bool indicating whether the single digits of that number are in ascending order. The digits are in ascending order if each digit is larger than the preceding digit. Negative numbers should be handled just like positive numbers. Example inputs and outputs are shown below.

<table>
<thead>
<tr>
<th>Call</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>isAscending(1234);</td>
<td>true</td>
</tr>
<tr>
<td>isAscending(-1234);</td>
<td>true</td>
</tr>
<tr>
<td>isAscending(29);</td>
<td>true</td>
</tr>
<tr>
<td>isAscending(3);</td>
<td>true</td>
</tr>
<tr>
<td>isAscending(41);</td>
<td>false</td>
</tr>
<tr>
<td>isAscending(1123);</td>
<td>false</td>
</tr>
<tr>
<td>isAscending(-1123);</td>
<td>false</td>
</tr>
<tr>
<td>isAscending(801);</td>
<td>false</td>
</tr>
</tbody>
</table>

*Constraints:* For full credit, obey the following restrictions:

- **Do not use any strings.** This problem is a lot easier if you simply convert the number to string. Don't do that!
- **Do not use any loops;** you must use recursion. If you do not use recursion, you will receive almost no points.
- **Do not use any auxiliary data structures** like Vector, Map, Set, Stack, Queue, array, string, etc.
- Do not declare any global variables.
- You can declare as many local variables as you like.
- You are allowed to define other "helper" functions if you like; they are subject to these same constraints.

Write answer on next page; work on this page will not be graded.
bool isAscending(int n) {
Problem Six: Backtracking (11 Points)

If you are unfamiliar with Wikipedia, it is an online encyclopedia, where each page links to many different related pages. Often, you can find amazing relationships between two completely different topics by clicking on random links. But why click on random links when you can have your computer tell you if and how certain topics are related?

Write a function bool getRelation(string start, string end, Vector<string>& path, const Map<string, Set<string>>& links) that takes in a string start and a string end respectively representing the titles of the start and end links. Your function will also take in a Vector by reference, which is initially empty. You should fill in that Vector with the path you followed to get from start to end. Lastly, your function receives a map called links. links maps from a string representing a page title to a set of strings containing all of the page titles that are linked from that page. For example, suppose that the page “Stanford” has links to “Harvard” and “UC Berkeley”, and suppose that the page “United States” has links to “California” and “Texas”. No other states have links. Then the Map would look like this:

```
{
    "Stanford": {"Harvard", "UC Berkeley"},
    "United States": {"California", "Texas"}
}
```

You may assume that links contains keys for all page titles in wikipedia.

For example, if we called your function with start="A Star Is Born" and end="Love", your method should return true, with a path like:
```
path = {"A Star Is Born","Christian Bale","Batman","Philanthropy","Love"}
```
because the page "A Star Is Born" links to "Christian Bale" which links to "Batman" and so on...

A few important notes:

- links is marked as const. You may **not** modify it.
- If the start and end passed in are the same, you should return true and fill the path with the page title.
- You may fill the parameter path with any valid path, it does not have to be the shortest path.
- For this problem, assume it is possible for a path not to exist. In this case, your function should return false. We do not care what is in the parameter path if a path could not be found, though we assume your code will probably leave the vector empty.
- Do not declare any global variables.
- Your code can contain loops if needed to solve the problem, but to receive full credit, your overall algorithm must be recursive.
- You may use a helper function, but it is also subject to these constraints.

Write answer on next page; work on this page will not be graded.
bool getRelation(string start, string end, Vector<string>& path,
    const Map<string, Set<string>>& links) {