COMP 285 (NC A&T, Spr '22) Homework 1

Due. Sunday, January 23th, 2022 @ 11:59 PM!

Homework Expectations: Please see the Homework part of the Course Website (comp285.ml/policies) for guidance on what we are looking for in homework solutions. We will grade according to these standards, and you should cite all sources you used outside course material.
What we expect: Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Exercises The following questions are exercises. We suggest you do these on your own. As with any homework question, though, you may ask the course staff for help.
Points This assignment is worth a total of 100 points.

Fun with Algorithms

This course will be a mixture of written homework and coding assignments. Both types of assignments will be submitted through Gradescope.

In order to reduce the overhead to the course staff, all coding assignments will be released through repl.it, an in-browser IDE. *This will be the only supported IDE* for the class.

By *supported*, we mean that if you run into issues on repl.it, the course staff will do everything in their power to resolve them. We *strongly encourage* you to use repl.it for the coding portions of the homework assignments.

1 Exercise: Setting up repl.it

(20 pt.) As such, the this graded exercise will walk you through setting up your repl.it account, joining the course, and getting your copy of the starter code. Even if you **do not plan** to use repl.it for writing/developing your solutions (because you have another preferred IDE), you will use it to obtain your starter code.

First, let's start by getting you an account and joining the right project:

1. If you don't already have a repl.it account with your "aggies.ncat.edu" email, head over to the repl.it homepage. On the top-right, click the 'Sign Up' and use your NCAT email address to sign-up.
2. Complete the registration process (eg, there's a verification email you should get)

3. Join the @COMP285 Team by following this link.

Next, let's get your starter code. From your repl.it homepage, you should have a menu down the left-hand side of the page (if not, maybe click the hamburger menu to open it?). Click on "Teams" as shown in Figure 1. You should see 'North Carolina A&T' under 'Education', as well as the 'COMP 285' Team Project.

Click on 'COMP 285', which will take you to the Team Homepage. From there, under 'Projects', you should see a link to get started on HW1-Code. Clicking it will give you the starter code, as well as set you up with an online.

If the above does not work, you can also try going directly to the project link.

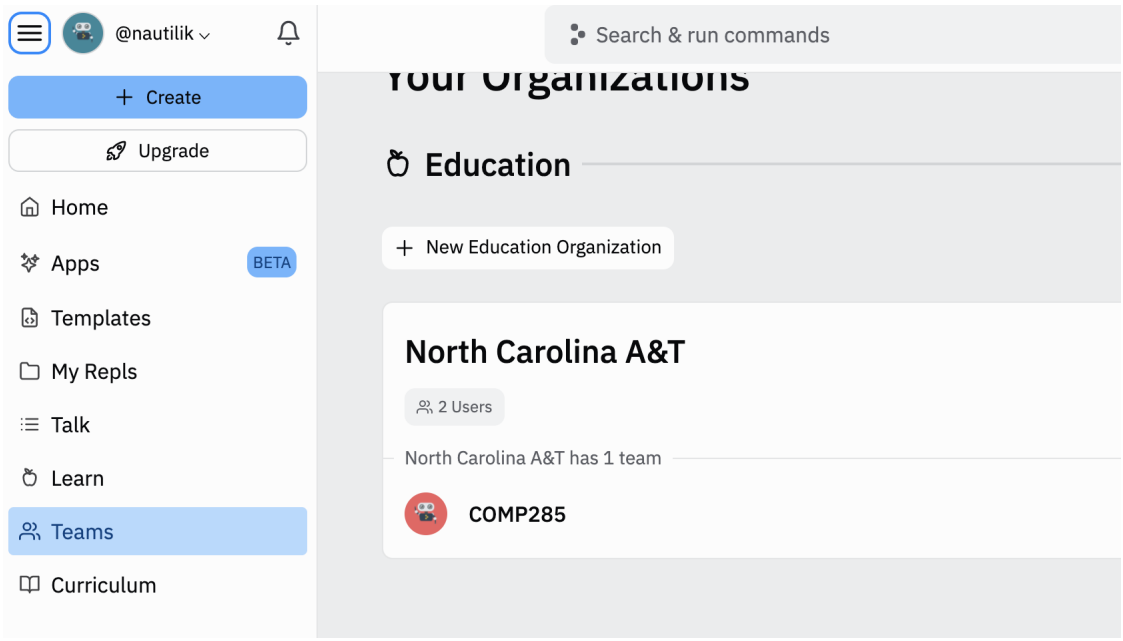

Figure 1: View of the repl.it Homepage for a student.

Either way, you should now have the starter code. Clicking the big green button at the top of the IDE will execute the `main` function defined in `main.cpp`. It'll fail, but that's okay. The rest of the assignment will walk you through implementing different functions in your starter code.

[We are expecting: You should have a copy of your starter code now available! You can use replit to make edits and run your code. The course staff can see who has started the project, so that is how you'll get credit for this question. No need to submit anything**]**

2 Interview Practice: Who's Missing!

(20 pt.) Trinitee is just wrapping up her software engineering internship at Foogle [1]. She recently developed an algorithm, `genereateIDs`, that gives unique IDs, starting from 0 and in increasing order, to each Google account in a particular region. This is how she defined her algorithm, in a '.h' file.

```
#include <vector>

// Returns a list of unique IDs for the specified `region'.
//
// `region' is a string that specifies a county, place, or location.
// This algorithm looks up every Foogle account in the location,
// and assigns a unique ID. IDs are generated sequentially
// (starting at with 0) and up to `n-1', where `n' is the number
// of people in the given region.
//
// For example:
//      If a region has 3 accounts, return {2, 0, 1}.
//      If a region has 5 accounts, return {1, 0, 3, 2, 4};
//
// Note that the returned IDs are not necessarily sorted.
std::vector<int> generateIDs(const std::string& region);
```

However, she's running into a few issues with her code. It seems that the returned vector is always too small - precisely, it seems to always be *exactly one element* too small.

For example, Trinitee knows that in the country of Algolandia [2], there are exactly $2^{20} = 1,048,576$ accounts. However, when she runs the following lines of code:

```
const auto ids = genereateIDs("Algolandia");
std::cout << "Length of my ids: " << ids.size();
```

It prints $1,048,575$, which is precisely one short. Confused, Trinitee decides to ask Asha, a fellow intern, for some help.

2.1 Is it sorted?

(5 pt.) Asha suggest to Trinitee that they first write a program to check if the IDs she's getting back are **sorted in descending order**. She thinks that if they're sorted, it'll be easy for them to find which one is missing!

[1] This is a meant to be a fake company.

[2] This is a fake country! Don't worry if you've never heard of it!

She starts writing out the program to do this (check if a vector of integers is **sorted in descending order**), but suddenly, someone comes a long and deletes part of her code! Below is what she has left.

```
1  // Checks if a vector<int> called `elements' is sorted in
2  // descending order. That is:
3  // {1, 2, 3, 4} returns false, because it's sorted but increasing.
4  // {4, 3, 2, 1} returns true, because it's sorted and decreasing.
5  bool isSortedDescending(const std::vector<???>& elements) {
6      for (int i = 0; i < element.size(); i++) {
7          if (???) {
8              return false;
9          }
10     }
11     return true;
12 }
```

Flabergasted, Asha turns to you for help!

1. **(2 pt.)** What should go in the first set of '???' in line 5 above? [3]
2. **(2 pt.)** What should go in the second set of '???' in line 7 above? [4]
3. **(1 pt.)** What is the big-Oh running time of the `isSortedDescending` function, if we let n be the number of elements?

[We are expecting: You should provide an answer for each of the three sub-questions above. The first two are snippets of code (just write it out), and the last one should be $O(\dots)$ where you fill-in the .... In total, they are worth 5 points out of 20 for this problem.**]**

2.2 So what account is missing?

(5 pt.) Sadly, it turns out the lists are not sorted! I guess the documentation was right this time! You, Trinitee, and Asha start brainstorming other solutions. How can we figure out which ID is the one that's missing?

Trinitee gives another suggestion. On the white board, she writes the numbers:

$$0, 1, 2, 3, 4, 5, 6, 7$$

And asks:

2.2.1 **(1 pt.)** What do the numbers above add up to?

[3] If you're having trouble, read the comment for the function carefully!

[4] If you're having rouble, this is very similar to what we did in class, but instead of **ascending**, here we want to check if the list is **descending**.

She then erases the numbers, and writes:

$$(0, 7); (1, 6); (2, 5); (3, 4)$$

She turns to you, and asks:

2.2.2 **(1 pt.)** What do each of the pairs add up to? How many pairs are there?

After thinking about the above for a bit, all three of you jump up excitedly! Asha goes to the board and writes:

$$0, 1, 2, 3, 4, \ldots, n-3, n-2, n-1, n$$

and then writes:

$$(0, n); (1, n-1); (2, n-2); (3, n-3); (4, n-4), \ldots$$

Turns out both Asha and Trinitee are quite smart! They turn to you, and ask:

2.2.3 **(3 pt.)** If we add up all the numbers from 0 to n, what do they add up to? Is there a closed-form formula? [5]

[We are expecting: An answer for each of the above questions. The first two should be a number, and the last should be a formula which uses n.**]**

2.3 Finding The Missing ID

(10 pt.) Having figured out the above, the three of you start coding! You come up with the following pseudo-code:

```
algorithm discoverMissingID
  Input: The output from the malfunctioning generateIDs function above.
  Output: The missing ID.

  s1 = the sum of all numbers in the given vec<int>
  s2 = what the sum should be if no numbers were missing. See answer to 2.2.3.
  return s2 - s1
```

Now, Trinitee and Asha look at you, and say: "Can you code it up for us?"

[We are expecting: You should implement the above pseudo-code in `answers.cpp`, which is part of the starter code you retrieved in Exercise 1**]**.

[5] If you're having trouble, check-out this video

3 Interview Practice: The Best Basketball Player

(20 pt.) Hassan has been following closely the latest happening with the Denver Nuggest [6]. He's a bit disappointed with their performance this season, so he reaches out to his friend Caleb.

Caleb is a data scientist working for the U.S. Census. He tells Hassan that a big issue with the Denver Nuggest this season is that they're players are too short! What they really need is a tall player - wait, not a tall player, **the tallest player** in America.

Thankfully, Caleb has just the data for the job! He shows Hassan that the census collects every person's height as an integer in inches [7] (they're too lazy, so they always round to the nearest inch).

3.1 Finding the Tallest American

(10 pt.) Hassan and Caleb feel like they've done a good job so far, so they reach out to you. They know you're an expert in C++, do they ask you to write a function - `findTallestPerson` - which takes in a vector of integers correspondong to each person's height from the census data Caleb has, and returns **the index** of the largest height. If there are multiple largest heights, it returns the smaller index.

They give you the following examples:

```
findTallestPerson({1, 3, 4, 5, 2});  // should return 3, the index of 5
findTallestPerson({-1, -3, -4, -5, -2});  // should return 0, the index of -1
findTallestPerson({});  // should throw std::invalid_argument
```

You look at them a bit funny, and wonder to yourself: "Who in the world has negative height? And why would the list ever be empty?" Still, you want to do a good job, so you do what they ask.

[We are expecting: You should implement `findTallestPerson` in `answers.cpp`, which is part of the starter code you retrieved in Exercise 1**]**.

3.2 Is it Correct?

(5 pt.) After implementing the above algorithm, you want to show all your friends how good you've gotten at *correcness* and *runtime* analysis. You write down the pseudocode:

```
algorithm findTallestPerson
    Input: A vector of integers representing height in inches
    Output: The index of the tallest person.
```

[6] If you're like me and do now know who they are, suffice it to say that they're a basketball team

[7] This is not true. It would be a violation of privacy if it was...

```
if input is empty:
    throw error

tallestIndex = 0
for index in {1, ..., input.size() - 1}:
    if input[index] > input[tallestIndex]:
        tallestIndex = index
return tallestIndex
```

You want to *prove* to Caleb and Hassan the algorithm above is correct. You start walking them through a proof by induction. Fill out the missing pieces.

- **Inductive Hypothes**: `tallestIndex` points to the index of the largest element in `input[:i+1]` after the i-th iteration (of the for loop)
- **(3 pt.) Base Case (i=0)** *TODO: FILL THIS OUT*
- **(2 pt.) Inductive step** Assume our **inductive hypothesis** is true for $i = k$ (eg, at the k-th) iteration. *Explain why* it will also be true for $i = k + 1$ (eg, at the next iteration of the for-loop).
- **Conclusion**: As shown above, the inductive hypothesis holds for $i = 0$, (**base case**). We also argued as part of the **inductive step**, that if it holds for $i = 0$, it is also true for $i = 1$, and if it's true for $i = 1$, it's true for $i = 2$, and so on. As such, it must be true for $i = n - 1$ (aka, when our for-loop finishes). Therefore, `tallestIndex` points to the index of the largest element in `input[:n]` after the $n - 1$-th iteration, which is exactly what we wanted.

[We are expecting: You should provide written responses to the two missing pieces in the above proof. The first is the base case, the second is an explanation for why the inductive step works.**]**

3.3 How fast is it?

(5 pt.) You also want to show everyone how good you are at analyzing the runtime of your algorithm. For your reference, you write down the formal definition of big-Oh: $T(n) = O(g(n))$ if and only if:

$$\exists c, n_0 \text{ s.t. } \forall n > n_0, T(n) \leq c \cdot g(n)$$

Then, you answer the following:

- **(2 pt.)** What is the big-Oh runtime of the pseudo-code shown above?
- **(2 pt.)** After practicing the guitar, Vincent (another amazing C++ coder), tells you: "I implemented your code, and counted up all the operations. There are exactly $T(n) = 3n + 1$ operations in my implementation! I think $T(n) = O(n)$". What values of c, n_0 would convince you that Vincent is right, and his implementation is in-fact $O(n)$?

- **(1 pt.)** Vincent thinks for a bit, and says - I think $T(n) = O(n^2)$! Is he right? If yes, what are the n_0 and c values that convince you he's right? If no, briefly explain why such values cannot exist.

[We are expecting: And answer to each of the above. The first should be of the form $O(\dots)$, where you've filled in the $\dots$. The second should be two numbers, the c and n_0 that convince you. The last should be either two numbere (the c and n_0 that convince you), or a short sentence explaning why they don't exist.**]**

4 Interview Practice: Game Development

(20 pt.) Nkumbu is a world-renowed game developer working for Big Corp Studios [8]. His game works by randomly generating pets, each which costs a random amount of coins (only integer values). The player is then allowed to purchase precisely *two* pets.

Being an awesome developer, Nkumbu wants to make sure that no matter how many coins the player has, there are always *two* pets that we can purchase at the cost of all his coins.

4.1 Finding the Most Expensive Pets

(10 pt.) Having been stumped with this problem for quite a bit, he reaches out to Isaiah, whose an avid gamer and recently took COMP 285: Analysis of Algorithms for help. He asks Isaiah to write a C++ function `findTwoPets`, which takes as input a vector of integers (corresponding to the cost of each pet) as well as a target value (corresponding to how many coins the player currently has), and returns a pair that represents two **distinct**[9] indices of pet costs that sum up to the target value.

For some reason, Nkumbu also wants the smaller index to be first. To be super clear, he gives a few examples to Isaiah:

```
std::vector<int> pet_costs = {0, 2, 3, 4, 5};
std::pair<int, int> out1 = findTwoPets(pet_costs, /*coins=*/6);
std::pair<int, int> out2 = findTwoPets(pet_costs, /*coinst*/10);

std::cout << out1.first << " " << out1.second;  // should output 1 3
std::cout << out2.first << " " << out2.second;  // should output -1 -1
```

In the first example, `vec[1] + vec[3] = 2 + 4 = 6`. In the second example, we returned `-1, -1` because there are not two distinct elements within the vector that sum to 10 (you can't use 5 twice).

Help Isaiah by writing an algorithm (in C++) that solves this problem. Your solution should be written in `answer.cpp` which is part of the starter code.

For context, there's a straight-forward $O(n^2)$ algorithm that you can start with (this is probably the first one you'll think of). There is also an $O(n \log n)$ algorithm (Hint: what if the list was sorted?), and even a far more efficient $O(n)$ algorithm.

For this homework, *there are no explicit time complexity constraints* here (i.e. algorithm just needs to work as intended). You *might* receive bonus points if you implement one of the faster ones, though.

[8] Again, fake name.

[9] Should be pretty obvious that we're not going to let a player buy the same pet twice!

[We are expecting: You should implement `findTwoPets` in `answer.cpp` which is part of the starter code. It does not need to be the best algorithm, it just needs to work.**]**

4.2 Correctness of Algorithm

(5 pt.) For whichever algorithm you implemented, please provide a few sentences *explaning why* it is correct. This does not need to be formal (but feel free to try a proof by induction).

[We are expecting: A few sentences explaining why the algorithm you write works. You can be informal about it, but try to be precise.**]**

4.3 Runtime of Algorithm

(5 pt.) For whichiever algorithm you implemented, what is the big-Oh runtime?

[We are expecting: An answer in the form of $O(\dots)$ with $\dots$ filled in.**]**

5 Exercise: Refresher on Nodes, Pointers, etc.

(20 pt.) In this exercise, we'll write some code that will set you up for implementing Karatsuba's algorithm later in the class (maybe as extra credit!).

5.1 Adding Two Numbers

(10 pt.) Almost every programming languages already has '+' and '*' implemented for you, on integers. So to make this more interesting, your task is to write the function `add`, which given two numbers represented as linked lists of single-digit integers, returns their sum as a linked list of integers. Return the head node.

The linked list is represented as follows:

```
template <typename T>
struct Node {
  T value_;
  Node<T>* next_ = nullptr;
};
```

We have provided a util function `makeList` to convert an integer to its corresponding list representation. Feel free to take a look at the function in `util.cpp` from the starter code and note how we extract the digits (`num % 10` and `num / 10` in the loop).

For convenience, the list stores the digits from the lowest to the highest, e.g., `makeList(123) => head -> [3 -> 2 -> 1]`.

Some examples for `add`:

```
Node<int>* q4bInput1 = makeList(123);  // [3 -> 2 -> 1]
Node<int>* q4bInput2 = makeList(242);  // [2 -> 4 -> 2]
Node<int>* q4bOut1 = add(q4bInput1, q4bInput2);
std::cout << listToString(q4bOut1)
        << std::endl;  // should output [5 -> 6 -> 3] (meaning 365)
freeList(q4bInput1);
freeList(q4bInput2);
freeList(q4bOut1);

Node<int>* q4bInput3 = makeList(999);
Node<int>* q4bInput4 = makeList(1);
Node<int>* q4bOut2 = add(q4bInput3, q4bInput4);
std::cout << listToString(q4bOut2)
        << std::endl;  // should output [0 -> 0 -> 0 -> 1] (meaning 1000)
freeList(q4bInput3);
freeList(q4bInput4);
freeList(q4bOut2);
```

Refresher on pointers, memory allocation, and deallocation for your reference:

- To create a `Node` object, use `Node<int>* ptr = new Node<int>()`. `new` dynamically allocates memory for this.
- We provided a `freeList` function that will traverse the linked list and free every node. `freeList` is all we need here, but always remember when you have allocated memory (i.e. with 'new') to free it when finished (i.e. 'delete ptr').
- To check whether there are "memory leaks" (i.e., allocated memory but not deallocated), you can run valgrind on repl.it with 'valgrind –leak-check-full ./main'

Please note that this question is at the upper-end of the C++ intricacies we will learn on in the course. Having familiarity with pointers here will serve us well later on (e.g. with trees).

[We are expecting: You should implement `add` in `answers.cpp` in your starter code.**]**

5.2 Runtime of Addition

(10 pt.) The algorithm above forces you to implement addition the way you would normally do it by hand (eg, add numbers and carry when they add to more than 10). In this context, let us define "operation" as being a 1-digit addition. What is the big-Oh running time of this algorithm, in terms of n where n is the length of the input lists (eg, numbers).

[We are expecting: A single answer in the form $O(\dots)$ where you've filled in $\dots$. No need to prove or argue this is correct, simply provide your answer.**]**

Submitting the Assignment

This assignment is a combination of written and programming questions. Both portions of the assignment should be submitted through Gradescope before the assignment due date on Januray 23rd, 2022 @ 11:59PM.

The "Homework 1: Fun with Algorithms" assignment is the written portion, for which you should submit a **typed** response to the non-coding questions above. Each response should clearly be marked with its corresponding number (for example, 2.1 or 2.2.2) etc. You are free to use the provided templates, or to simply type your responses on a blank document (whatever works for you).

The "Homework 1: Coding" is the programming portion of the assignment. For this portion, download the ".zip" file from replit (see Figure 2), and upload this ".zip" file as your answer to Gradescope (see Figure 2). You can upload the assignment as many times as you want - each time a set of automatic tests will run that test your code for correctness. Feel free to use this to detect when your code is wrong.

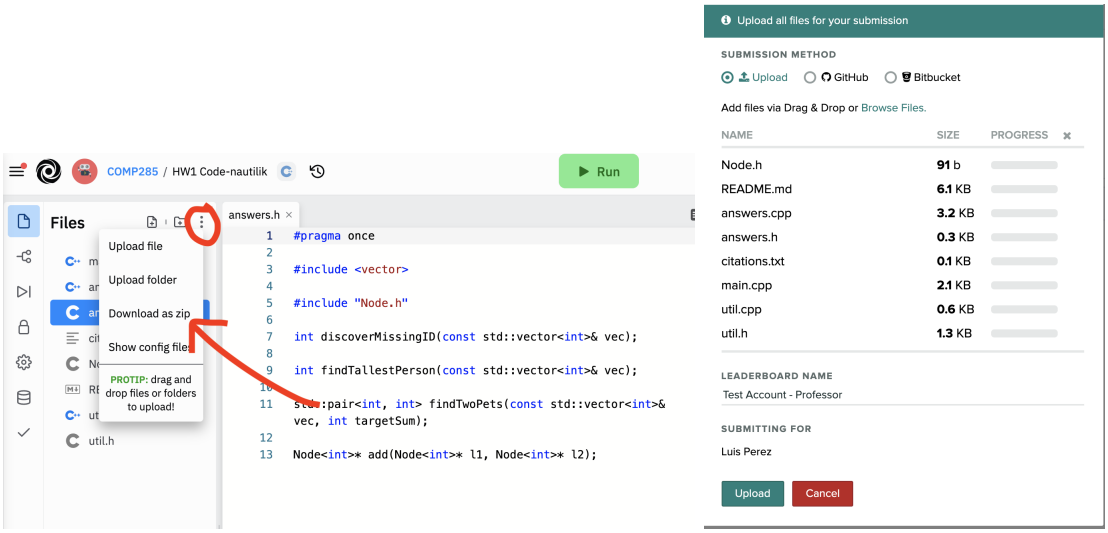

Figure 2: (left) View of the hamburger menu which holds the option to download your source code as a ".zip" which can be directly uploaded to Gradescope. (right) View of the screen you should see after uploading your ".zip" file. Make sure all the files are present, since they're requires for the autograder to run.

To summarize, do the following:

1. Submit your **typed** responses to the non-coding portion of the assignment on Gradescope for "Homework 1: Fun with Algorithms"
2. Submit your **code files** as a ".zip" to Gradescope for "Homework 1: Coding".
3. Fill out this **Google Form** (https://forms.gle/pvoM4pYV2u4yYXc96) to help me track your thoughts on the homework.