CPSC 221: Theory Assignment 1

Last Updated: January 25, 2017 Due: 9pm, Tuesday, February 7, 2017.

Submission Instructions

Submit your solutions using **`handin`**. You can write your solutions by hand and scan the pages or take pictures of them with your phone; or use a word processing package to typeset your solutions and produce a .pdf file. If using a picture, please don't use a high resolution as it can quickly consume a lot of space and upload/download time.

To submit: Copy the files that contain your solutions to the directory `~/cs221/theory1` in your home directory on an undergraduate machine. (You may have to create this directory using `mkdir ~/cs221/theory1`) Then run **`handin cs221 theory1`** from your home directory.

We encourage you to work in pairs. **Be sure to include the names and ugrad login IDs of both partners on all solution pages, but only one partner should submit the assignment.**

Late submission policy: The late penalty is 3% per hour (or portion thereof), with no late assignments being accepted after 12 hours. For example, if you hand in your program at 02:10 AM on the morning after the due date, then this is 6 hours late; so, you would lose 18% of the maximum possible mark.

Part 1 – Induction:

1) Let P(n) be the statement that $1^3 + 2^3 + \ldots + n^3 = (n(n+1)/2)^2$ for any positive integer n.
   a. What is the statement P(1)?
   b. Show that P(1) is true, completing the base case of this proof.
   c. What is the inductive hypothesis?
   d. What do you need to prove in the inductive step?
   e. Complete the inductive step.
   f. Explain why these steps show that this formula is true whenever n is a positive integer.

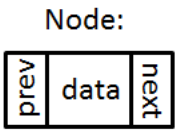
2) Prove the following statement:

   For all integers n>= 1, $1 + 6 + 11 + 16 + \ldots + (5n-4) = n(5n - 3)/2$

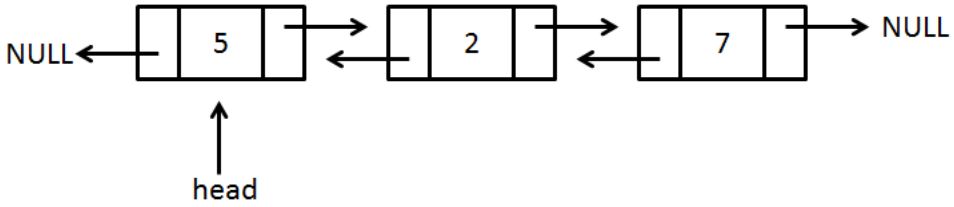
Part 2 – Pointers:

Consider the following class declaration for a node in a "doubly-linked list", which creates a class whose objects have three fields: an integer (int) value, a pointer to the next node, and a pointer to the previous ("prev") node:

```
class Node {
public:
   Node(int v) { data = v; }
   int data;
   Node *next;
   Node *prev;
};
```



The diagram below shows the state of memory just before some code executes:



3) Draw a diagram of memory after the following five lines of code execute:

```
// trying to insert a node at the front (head) of the list
Node *toAdd = new Node(4);
toAdd->next = head->next;
head->next->prev = toAdd;
head = toAdd;
```

4) The code in (3) isn't correct. Write code that uses a doubly-linked list to properly add a node to the front of the list.
5) Draw a diagram representing the state of memory when your corrected code completes.
6) Using the original diagram, draw a memory diagram after the following lines of code execute:

```
// trying to remove a node at the front of the list
head->next->prev = head->prev;
delete head;
head = head->next;
```

7) Explain why the above code is not safe.
8) Fix the code so that it is safe and so that it properly removes the node at the front of the list.

Part 3 – Algorithm Analysis:

9) Find the smallest integer x such that f(n) is $O(n^x)$, and positive constants c and n_0 to satisfy the definition for Big-O defined in class, for each of the following functions:
   a. $f(n) = n^2 + 25n - 4$
   b. $f(n) = 5n^3 + 3n^2 \log n$
   c. $f(n) = (n^4 + n^2 + 1) / (n^3 + 1)$

10) Explain, by determining the expected run-time, the differences (if any) using an array-based versus a linked-list-based implementation for the following operations:
   a. `find(int index);  // Returns the value at given index`
   b. `remove(int index); // Removes the value at index and`
      `// maintains adjacent ordering from`
      `// start to finish (e.g., non-`
      `// decreasing order).`

11) Give tight asymptotic **worst-case bounds** for the time complexity of each of the following pieces of C++ code. Show your work. How much time does each loop take? First calculate the running time, T(n), (which you can assume is the number of lines of code executed as a function of n) then express its order of growth using Θ-notation.

   a. Let size represent n, the number of elements in the sorted array:

```
// Reverses the values in the array arr
void reverse(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        int temp = arr[0];
        for (int j = 0; j < size - i - 1; j++) {
            arr[j] = arr[j+1];
        }
        arr[size-i-1] = temp;
    }
}
```

(Part b for this question is on the following page)

b. Let size represent n, the number of elements in the sorted array:

```
// Searches a sorted array for the value n
// Returns index of n, or -1 if array does not contain n
int search (int arr[], int size, int n) {
    int min = 0, max = size-1;
    int mid;
    int max = size-1;
    while (min <= max) {
        mid = (min + max)/2;
        if (arr[mid] < n) {
            min = mid+1;
        } else if (arr[mid] > n) {
            max = mid-1;
        } else {
            return mid;
        }
    }
    return -1;
}
```