CPSC 221: Algorithms and Data Structures
Assignment #1, due Wednesday, 2014 May 28 at 17:00 (5pm) PST

Submission Instructions

Type or write your assignment on clean sheets of paper with question numbers prominently labeled. Answers that are difficult to read or locate may lose marks. We recommend working problems on a draft copy then writing a separate final copy to submit.

Each submission should include the names and student IDs of the authors at the top of each page. (You are strongly encouraged to work in pairs, but may not work in groups of three or more. Each pair submits a single copy of the assignment.) On your first page, also sign the statement "I have read and complied with the CPSC 221 2014S1 academic conduct policy as posted on the CPSC 221 course website." (See: `http://www.ugrad.cs.ubc.ca/~cs221/2014S1/syllabus.shtml#conduct`.) In keeping with the policy, you should also acknowledge on your first page any collaborators or resources that helped you with the assignment. Finally, staple your submissions pages together! We are not responsible for lost pages from unstapled submissions.

Submit your assignment to Box 31, in room ICCS X235. Late submissions are not accepted.

Questions

[10] 1. Stacks.

You are given a stack of integers. The stack has been implemented using an array. Assuming that the stack's initial state is given below,

(a) Draw the stack after each push operation has completed.

0	1	2	3	4	5	6	7	8	9
5	2	8	4						

Solution:

`push(5)`

0	1	2	3	4	5	6	7	8	9
5	2	5							

`push(6)`

0	1	2	3	4	5	6	7	8	9
5	2	5	6						

`push(24)`

0	1	2	3	4	5	6	7	8	9
5	2	5	6	24					

`push(7)`

0	1	2	3	4	5	6	7	8	9
5	2	7							

(b) If every time `top` is called, we also write down the integer value obtained, what list of numbers do we get?

Operations: `pop()`, `top()`, `pop()`, `push(5)`, `push(6)`, `top()`, `push(24)`, `top()`, `pop()`, `pop()`, `pop()`, `top()`, `push(7)`, `top()`.

Solution: 8, 6, 24, 2, 7.

[15] 2. Queues.

Consider the following C++ linked list implementation of a queue of integers (`int`).

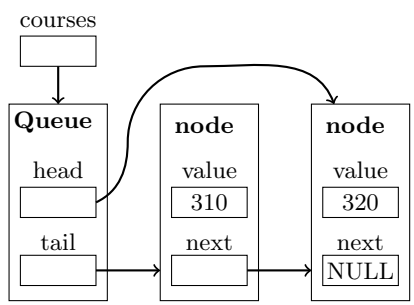
```
class Queue
{
   struct node {
      int value;
      node *next;
   };
   node *head = NULL, *tail = NULL;

   void enqueue(int n)
   {
      node *a = new node();
      a->value = n;
      if (head==NULL)
         tail = a;
      else
         head->next = a;
      head = a;
   }

   bool isEmpty()
   {
      return head==NULL;
   }

   int dequeue()
   {
      if (isEmpty())
         throw "Nothing to dequeue.";
      node *temp = tail->next;
      int n = tail->next;
      if (head==tail)
         head = NULL;
      delete tail;
      tail = temp;
      return n;
   }
};
```

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

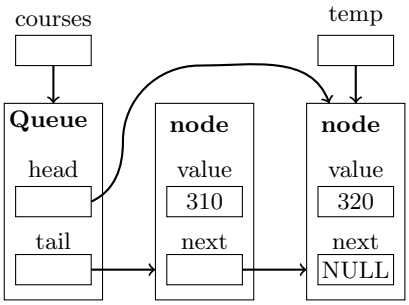


Two **SUCCESSIVE** calls are made on the object pointed to by the pointer `courses`: `courses->dequeue()` and `courses->enqueue(221)`. For each operation, draw models representing the state of memory at the specified times.

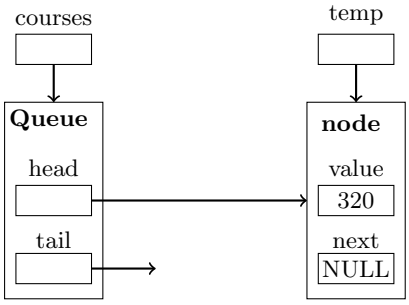
Solution:

(a) For the call to `courses->dequeue()`, draw the memory after the lines

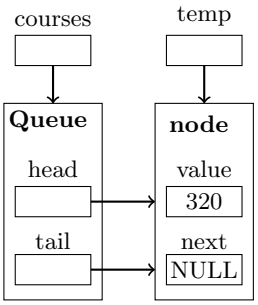
i. `node *temp = tail->next;`



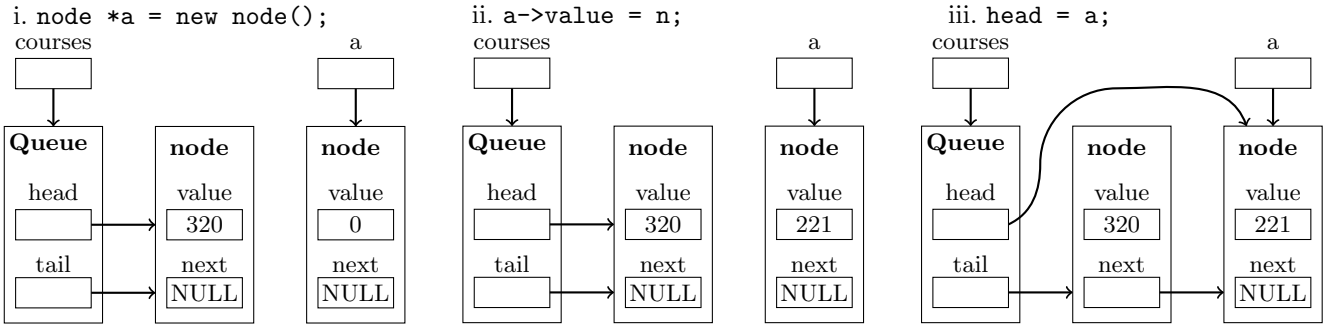
ii. `delete tail;`



iii. `tail = temp;`



(b) For the call to `courses->enqueue(221)`, draw the memory after the lines



[20] 3. Simplifying to Asymptotic Bounds.

For each of the following definitions of $T(n)$, give the big-Θ bound, as simplified as you can. You do not need to prove your result. However, to get partial credit for any minor mistakes, you should show how you arrived at your answer.

(a) **Example:** $T(n) = 42$, answer $\Theta(1)$.

(b) $T(n) = n^3 + n^2 + n$
Solution: $T(n) \in \Theta(n^3)$ since we can discard lower order terms of polynomials.

(c) $T(n) = (n+4)(n+2) - n^2$
Solution: $T(n) = (n+4)(n+2) - n^2 = n^2 + 6n + 8 - n^2 = 6n + 8$ so $T(n) \in \Theta(n)$ since we can discard lower order terms of polynomials.

(d) $T(n) = \frac{n \log n}{2n} + n$
Solution: $T(n) = \frac{n \log n}{2n} + n = \frac{1}{2} \log n + n$ so $T(n) \in \Theta(n)$ since asymptotically $n > \log n$.

(e) $T(n) = \sum_{i=0}^{n-2} (3i + 2)$
Solution: $T(n) = \sum_{i=0}^{n-2} (3i+2) = 3 \sum_{i=0}^{n-2} i + 2(n-1) = 3 \cdot \frac{(n-2)(n-1)}{2} + 2n - 2 = \frac{3}{2}n^2 - \frac{5}{2}n + 1$ so $T(n) \in \Theta(n^2)$ since we can discard lower order terms of polynomials and constants.

(f) $T(n) = \sum_{i=0}^{n-2} (3i+2)^2$
Solution: $T(n) = \sum_{i=0}^{n-2} (3i+2)^2 = 3 \sum_{i=0}^{n-2} (9i^2 + 12i + 4) = 9 \sum_{i=0}^{n-2} i^2 + 12 \sum_{i=0}^{n-2} i + 4(n-1) = 9 \cdot \frac{(n-2)(n-1)(2(n-2)+1)}{6} + 12 \cdot \frac{(n-2)(n-1)}{2} + 4(n-1)$. Simplifying out yields a polynomial of degree 3, so $T(n) \in \Theta(n^3)$ since we can discard lower order terms of polynomials and constants.

(g) $T(0) = 1$ and $T(n) = 2T(n-1) + 2$.
Solution: Notice that $T(n) = 2T(n-1) + 2 = 2^2 T(n-2) + 2^2 + 2 = 2^3 T(n-3) + 2^3 + 2^2 + 2 = \cdots = 2^n T(0) + (2^n + 2^{n-1} + \cdots + 2^2 + 2)$. Then, $2^n + 2^{n-1} + \cdots + 2^2 + 2$ is a geometric series, so $2^n + 2^{n-1} + \cdots + 2^2 + 2 = 2(2^{n-1} + \cdots + 2^1 + 2^0) = 2(2^n - 1)$. Putting everything together,

$$T(n) = 2^n T(0) + (2^n + 2^{n-1} + \cdots + 2^2 + 2) = 2^n + 2(2^n - 1) = 3 \cdot 2^n - 2.$$

Then, $T(n) \in \Theta(2^n)$.

[15] 4. Comparing Asymptotic Behaviours.

(a) Let $f(n) = 8^{2 \lg n}$ and $g(n) = 3n^7 + 7n$.

i. Asymptotically, does $f(n)$ or $g(n)$ grow faster? For this question, you may find it useful to use logarithm identities. Show your work for partial marks.

Solution: Using some logarithm identites, $8^{2 \lg n} = (2^3)^{2 \lg n} = 2^{6 \lg n} = 2^{\lg (n^6)} = n^6$. Then, for large n, $n^6 < n^7 < 3n^7 + 7n$, so $f(n) < g(n)$. This means that $g(n)$ grows faster asymptotically.

ii. Fill in the following blanks:

A. $f(n)$ is big-_____ of $g(n)$. **Solution:** $f(n)$ is big-O of $g(n)$.

B. $g(n)$ is big-_____ of $f(n)$. **Solution:** $g(n)$ is big-Ω of $f(n)$.

(b) Suppose a function $h(n)$ describing the runtime of one program is big-O of another function $k(n)$ describing the runtime of another program, can we say that at some point (for some sufficiently large input size n) the second program will take longer than the first? You may use examples in your answer.

Solution: No, we cannot say this. Consider the following counterexample: let $h(n) = n$ and $k(n) = \frac{n}{2}$. $h(n)$ is big-O of $k(n)$ since $\lim_{n\to\infty} \frac{n}{n/2} = 2$ is less than infinity. However, $h(n) > k(n)$ for any positive n you choose, no matter how large. ($1 > \frac{1}{2} \implies n > \frac{n}{2}$ provided $n > 0$).

[25] 5. Analyzing Runtime and Some Proofs.

Consider the following two functions `foo` and `bar` implemented in C++ below.

```
void foo(int n)
{
   for (int i=0; i<n; ++i)
   {
      if (i%2==0)
         cout << "foo" << endl;

      for (int j=0; j<n; ++j)
         cout << j << endl;
   }
}
```

```
void bar(int n)
{
   for (int i=0; i*i<n; ++i)
   {
      foo(n);

      for (int j=i; j<n; ++j)
         cout << j << endl;
   }
}
```

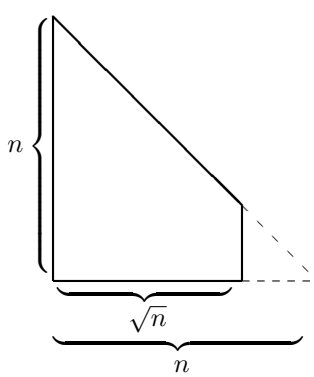
(a) Find an equation for the time complexity of each of the functions `foo` and `bar`. In your answer, you may assume that `cout` operations all take the same amount of time, and label this time with the constant c.

Solution: Let the function describing the runtime of `foo` be $T_1(n)$ and the function describing the runtime of `bar` be $T_2(n)$.

- When looking at `foo`, we can tell that the outer loop runs n times, so $T_1(n)$ will be the product of n and the runtime of the contents of the outer `for` loop. The `if` statement is true for roughly half of the time (when `i` is even), so on average it takes $\frac{c}{2}$ time to execute. The inner `for` loop runs n times, so it takes cn time. Putting everything together, we get $T_1(n) = n\left(\frac{c}{2} + cn\right) = cn^2 + \frac{c}{2}n$.
- When looking at `bar`, things are not so easy. Since the `for` loop ends as soon as `i*i` is larger than or equal to `n`, we can estimate that the loop ends roughly after $i^2 = n \to i = \sqrt{n}$. From this estimate, we know the outer loop runs $\sqrt{n}$ times.

  Then, the total time spent executing `foo(n)` is the product of the number of times the outer loop runs and the time it takes to execute `foo`, so $\sqrt{n} \cdot T_1(n) = \sqrt{n} \cdot \left(cn^2 + \frac{c}{2}n\right) = cn^{2.5} + \frac{c}{2}n^{1.5}$.

  Now, for the inner `for` loop. This is probably the trickiest part of the homework assignment. Drawing a picture (analogous to the one in lecture 2 on slide 61), we get something that looks like the drawing below.



Here, the `i` index is horizontal and the `j` index is vertical. We already know that the `i` index runs from 0 to $\sqrt{n}$, so we can cut off the image there. Next, since the inner loop is from `i` to `n`, as `i` increases, the number of values for `j`, $n - i$, decreases. This is the downward sloping line. Then, the total time spent inside both of these loops is the area inside the bold line (the trapezoid) times c. The figure's area is the difference of two triangles

$$\frac{1}{2}n^2 - \frac{1}{2}(n - \sqrt{n})^2 = n^{1.5} - \frac{1}{2}n$$

(Remember we still need to multiply by c). Adding this to the time spent in `foo(n)`, we get

$$T_2(n) = \left(cn^{2.5} + \frac{c}{2}n^{1.5}\right) + c\left(n^{1.5} - \frac{1}{2}n\right) = cn^{2.5} + \frac{3c}{2}n^{1.5} - \frac{c}{2}n$$

(b) Prove formally using the definition of big-O that `foo` has a runtime that is $O(n^3)$.

Solution: Using the limit definition of big-O, we need to verify that $\lim_{n\to\infty} \frac{T_1(n)}{n^3}$ is equal to some finite number.

$$\lim_{n\to\infty} \frac{T_1(n)}{n^3} = \lim_{n\to\infty} \frac{cn^2 + \frac{c}{2}n}{n^3} = 0$$

The limit goes to zero since the largest power of n in the denominator is greater than the one in the numerator ($3 > 2$). By the definition of big-O, $T_1(n) \in O(n^3)$.

(c) Prove formally using the definition of big-Ω that `foo` has a runtime that is not $\Omega(n^{2.5})$.

Solution: Using the limit definition of big-Ω, we need to verify that $\lim_{n\to\infty} \frac{T_1(n)}{n^{2.5}}$ is not greater than 0.

$$\lim_{n\to\infty} \frac{T_1(n)}{n^{2.5}} = \lim_{n\to\infty} \frac{cn^2 + \frac{c}{2}n}{n^{2.5}} = 0$$

The limit goes to zero since the largest power of n in the denominator is greater than the one in the numerator ($3 > 2.5$). By the definition of big-Ω, $T_1(n) \notin \Omega(n^{2.5})$.

(d) Prove formally using the definition of big-Θ that `bar` has a runtime that is $\Theta(n^{2.5})$.

Solution: Using the limit definition of big-Θ, we need to verify that $\lim_{n\to\infty} \frac{T_2(n)}{n^{2.5}}$ is greater than 0 but less than infinity.

$$\lim_{n\to\infty} \frac{T_2(n)}{n^{2.5}} = \lim_{n\to\infty} \frac{cn^{2.5} + \frac{3c}{2}n^{1.5} - \frac{c}{2}n}{n^{2.5}} = c$$

The limit goes to c since the largest power of n in the denominator is equal to the one in the numerator, and discounting the other terms, these cancel out to leave c. However, c is neither 0 nor ∞ so, by the definition of big-Θ, $T_2(n) \in \Theta(n^{2.5})$.

[15] 6. Proof Practice.

Suppose you have two positive functions $f(n)$ and $g(n)$ where $f(n) \in O(h(n))$ and $g(n) \in O(k(n))$. If $k(n) \in O(h(n))$, show that $f(n)+g(n) \in O(h(n))$. (Hint: try combining the limits you get from $f(n) \in O(h(n))$, $g(n) \in O(k(n))$, and $k(n) \in O(h(n))$).

Solution:

Proof. By definition of big-O, there exist numbers c_1, c_2, and c_3 such that

$$\begin{aligned} f(n) \in O(h(n)) \quad &\leftrightarrow \quad \lim_{n\to\infty} \frac{f(n)}{h(n)} = c_1, \\ g(n) \in O(k(n)) \quad &\leftrightarrow \quad \lim_{n\to\infty} \frac{g(n)}{k(n)} = c_2, \\ k(n) \in O(h(n)) \quad &\leftrightarrow \quad \lim_{n\to\infty} \frac{k(n)}{h(n)} = c_3. \end{aligned}$$

Then, it follows that we have

$$\begin{aligned} \lim_{n\to\infty} \frac{f(n)+g(n)}{h(n)} &= \lim_{n\to\infty} \left[\frac{f(n)}{h(n)} + \frac{g(n)}{k(n)} \cdot \frac{k(n)}{h(n)}\right] = \left(\lim_{n\to\infty} \frac{f(n)}{h(n)}\right) + \left(\lim_{n\to\infty} \frac{g(n)}{k(n)}\right) \cdot \left(\lim_{n\to\infty} \frac{k(n)}{h(n)}\right) \\ &= c_1 + c_2c_3. \end{aligned}$$

We are permitted to split the limit since each of the component limits converges. It follows that, since $c_1 + c_2c_3$ is finite, $f(n) + g(n) \in O(h(n))$. $\square$