CSCI 2300 — Data Structures and Algorithms
Sections 8, 9, 10
Exam 1 Review Solutions

Overview

- There will be no makeup exam if you miss the exam.
- The exam is closed-book and closed-notes. No crib sheets are allowed.
- Any formulas that you need will be given on the exam.
- Syntax mistakes will not be penalized unless the mistakes make the code ambiguous or unclear.
- STL syntax will also not be a concern unless you mistakenly assume operations that do not and should not exist, such as a push_front function for a vector.

Coverage

- Linear structures: linked lists, pointers, arrays, stacks, queues, vectors.
- Applications of linear structures: polynomial ADTs.
- Induction and recursion.
- Order notation.
- Algorithm analysis, including recurrences.
- Trees, binary trees, and binary search trees.

You will be asked to write code, debug code, analyze code and algorithms, and do an induction proof.

Emphasis In Studying

Do not memorize anything! Concentrate on the example code, the labs, the project, the homework problems and the review problems below. When looking at the example code, cover up sections of the code and try to see if you can write something similar. When examining code, ask yourself why the code was written as it was. Certainly, there are alternative methods, but are these methods just as good?

Warning

Exam problems will be at least as difficult as homework problems.
Practice Problems

To prepare for the exam you should review the class notes, labs and projects, and you should rework the homework problems. Included below are a few extra problems to help you prepare for the exam. Solutions will be posted on the web page. This does not completely cover the course material.

1. Why is the standard library vector container class not a good implementation for queues?

   Solution: The standard vector container class provides neither push_front nor pop_front operations because these are inefficient operations on an array: all remaining values would have to be shifted over by one location to fill the empty space — a linear time operation. The standard library is designed to explicit prevent such inefficiencies by not providing such operations. Hence, the vector container class should not be used for a queue.

2. Write a function to reverse the direction of the pointers in a singly-linked list and another one to reverse the direction of the pointers in a doubly-linked list. The former is much harder than the latter. Assume the structure

   ```
   template <class T>
   struct Node {
       T value;
       Node * next;
   };
   ```

   for singly-linked lists and

   ```
   template <class T>
   struct Node {
       T value;
       Node * next;
       Node * prev;
   };
   ```

   for doubly-linked lists. Assume each function call is

   ```
   void Reverse( Node* & head )
   ```

   and that head is to be changed to point to the node that previously was the tail.

   Solution: Here is the solution for singly-linked list. Note that at the end of each loop iteration, p points to the head of the reversed part of the list and q points to the head of the unreversed part and *p is the former predecessor of *q.
void Reverse( Node* & head )
{
    if ( !head ) return;
    Node * p = head;
    Node * q = head->next;
    while (q) {
        Node * r = q->next;
        q->next = p;  // reverse the direction of node *q
        p = q;
        q = r;
    }
    head = p;
}

Here is the solution for doubly-linked list. Note that fewer scratch pointers are needed because each node knows both its predecessor and successor.

void Reverse( Node* & head )
{
    if ( !head ) return;
    do {
        Node * old_next = head->next;
        head->next = head->prev;
        head->prev = old_next;
        if ( !old_next ) break;  // end of list has been reached
        head = old_next;
    } while (true);
}

3. Give an inductive proof showing that for all integers \( n \geq 5, 4n + 4 < n^2 \).

Solution:

Basis Case: When \( n = 5 \), \( 4n + 4 = 24 \) and \( n^2 = 25 \). Since \( 24 < 25 \), the basis case is proved.

Induction Step: For \( n > 5 \), if \( 4k + 4 < k^2 \) for \( 5 \leq k < n \) then

\[
4n + 4 = [4(n - 1) + 4] + 4 \\
< (n - 1)^2 + 4 \quad \ldots \text{ by the Inductive Hypothesis} \\
= n^2 - 2n + 5 \\
< n^2 \quad \ldots \text{ since } -2n + 5 < 0 \text{ if } n > 5
\]
4. Give an inductive proof showing that for all positive integers \( n \),
\[
\sum_{i=1}^{n} \frac{1}{(2i-1)(2i+1)} = \frac{n}{2n+1}.
\]

**Solution:**

**Basis Case:** \( n = 1 \). The left hand side is
\[
\sum_{i=1}^{1} \frac{1}{(2i-1)(2i+1)} = \frac{1}{1 \cdot 3} = \frac{1}{3}.
\]
The right hand side is
\[
\frac{1}{2 \cdot 1 + 1} = \frac{1}{3}.
\]
Since these are equal, the basis case is established.

**Induction Step:** For \( n > 2 \), if \( \sum_{i=1}^{k} \frac{1}{(2i-1)(2i+1)} = \frac{k}{2k+1} \) for \( 1 \leq k < n \) then
\[
\sum_{i=1}^{n} \frac{1}{(2i-1)(2i+1)} = \sum_{i=1}^{n-1} \frac{1}{(2i-1)(2i+1)} + \frac{1}{(2n-1)(2n+1)}
\]
\[
= \frac{(n-1)}{2n-1} + \frac{1}{(2n-1)(2n+1)} \quad \text{... by the inductive hypothesis}
\]
\[
= \frac{1}{2n-1} \left[ \frac{(n-1)(2n+1)}{2n+1} + \frac{1}{2n+1} \right]
\]
\[
= \frac{1}{2n-1} \cdot \frac{2n^2-n}{2n+1}
\]
\[
= \frac{1}{2n-1} \cdot \frac{2n^2-n}{2n+1}
\]
\[
= \frac{1}{2n+1}.
\]

5. (This one is a bit harder, but good practice.) Prove using mathematical induction that \( f_n > (3/2)^n \), for \( n \geq 5 \). Here, \( f_n \) is the \( n \)th Fibonacci number. Use the definition of the Fibonacci numbers given in the text (Chap. 1.2.5). Study the example inductive proof on page 6 carefully.

**Solution:**

**Basis case:** Two basis cases are needed. (This is important!) For \( n = 5 \),
\( f_5 = 8 \) and \( (3/2)^5 < 7.594 \). For \( n = 6 \), \( f_6 = 13 \) and \( (3/2)^6 < 11.391 \).
In each of these cases, \( f_n > (3/2)^n \).
Induction hypothesis: For all \( k, 5 \leq k < n \), \( f_k < (3/2)^k \).

Induction step: For \( n \geq 7 \),

\[
 f_n = f_{n-1} + f_{n-2} > (3/2)^{n-1} + (3/2)^{n-2} \quad \text{Induction Hypothesis} \\
 = \frac{(3/2)^n}{3/2} + \frac{(3/2)^n}{(3/2)^2} \\
 = \frac{2}{3}(3/2)^n + \frac{4}{9}(3/2)^n \\
 = \frac{2}{3} + \frac{4}{9}(3/2)^n \\
 = \frac{10}{9}(3/2)^n > (3/2)^n
\]

Hence, \( f_n > (3/2)^n \).

6. Write the code necessary to accomplish the merging step in MergeSort. This code should follow the comments in the main MergeSort function.

Solution:

The following is probably more terse than what you came up with on your own. Be sure you understand what it is doing. It could be made more efficient, mostly by eliminating the need for allocating temp.

```c
T* temp = new T[high-low+1]; // scratch array for merging
int i=low, j=mid+1, loc=0;

// while neither the left nor the right half is exhausted,
// take the next smallest value into the temp array
while ( i<=mid && j<=high ) {
    if ( pts[i] < pts[j] ) temp[loc++] = pts[i++];
    else temp[loc++] = pts[j++];
}

// copy the remaining values --- only one of these will iterate
for ( ; i<=mid; i++, loc++ ) temp[loc] = pts[i];
for ( ; j<=high; j++, loc++ ) temp[loc] = pts[j];

// copy back from the temp array
for ( loc=0, i=low; i<=high; loc++, i++ ) pts[i]=temp[loc];
delete [] temp;
```
7. Show that \( \sum_{i=1}^{n} 2i^3 = O(n^4) \).

Solution:
\[
\sum_{i=1}^{n} 2i^3 < \sum_{i=1}^{n} 2n^3 = n \cdot 2 \cdot n^3 = 2n^4 = O(n^4)
\]

8. For each of the following, find \( f(n) \) such that \( t(n) = O(f(n)) \). Make \( f(n) \) as small and simple as possible, i.e. don’t write \( t(n) = O(n^4) \) when \( t(n) = O(n^3) \). Justify your answers.

(a) \( t(n) = 13n^2 + 2^n \)

Solution:
\( t(n) = O(2^n) \) since exponentials of base greater than 1 grow faster than polynomials.

(b) \( t(n) = 5(n + 3\log n)(n\log n + 13)\log n + 13n^2 \)

Solution:
\( t(n) = O(n^2(\log n)^2) \). Looking at the factors of the first term yields, \((n+3\log n) = O(n), (n\log n+13) = O(\log n), \) and \( \log n = O(\log n) \). Multiplying these yields \( O(n^2(\log n)^2) \) which is bigger than \( O(n^2) \).

9. Exercise 2.7a from the text. Note program fragment (6) is quite difficult, and beyond what might be asked on an exam.

Solution:

In each case, the running time depends on the total number of iterations of the inner loop.

(1): \( O(N) \) since there is just one for loop and it executes \( N \) times.

(2): \( O(N^2) \). There are two nested for loops and each loop has \( N \) iterations. This means the inner loop statement executes \( N^2 \) times.

(3): \( O(N^3) \) since are two nested for loops, the inner loop has \( N^2 \) iterations and the outer loop has \( N \) iterations.

(4): \( O(N^2) \). There are two nested for loops. The inner loop has \( i \) iterations, but this is a loop variable, so we need to get rid of it. We do so by finding an upper bound on \( i \), which is of course \( N \). Since the outer loop also has \( N \) iterations, the result follows.

(5): \( O(N^8) \). The outer loop has \( N \) iterations. The second loop has \( i^2 \) iterations for each iteration of the outer loop. The upper bound on this is \( N^2 \). The inner loop has \( j \) iterations, but the upper bound on \( j \) is also \( N^2 \). Combining these gives the result.
(6): This one is tricky! The innermost loop is only reached \( i \) times for each value of \( j \) — once each time \( j \) is a multiple of \( i \). The if test, however, is done for each value of \( j \). Thus, I have broken up the count of the `for (int j ...)` loop into two parts — one for the if test and one for the innermost `for` loop, which is executed in its entirety once each time \( j \) is a multiple of \( i \). The latter can be written as a summation of \( i, i^2, i^3 \), etc. Together, these give

\[
\sum_{i=1}^{N} (N^2 + \sum_{m=1}^{i} mi) = N^3 + \sum_{i=1}^{N} i(i+1) \leq N^3 + \sum_{i=1}^{N} N\frac{(N+1)}{2} = N^3 + N^2\frac{(N+1)}{2} = O(N^4)
\]

10. Give an efficient algorithm to determine if there exists an integer \( i \) such that \( i = A_i \) in an array of integers \( A_0 < A_1 < \cdots < A_{n-1} \). What is the running time of your algorithm?

**Solution:**
The simplest solution is to just test each entry of the list. This clearly requires \( O(n) \) time. There is a faster way.

The following algorithm requires \( O(\log n) \) time because it is essentially the same as binary search. The trick depends on two facts: the array holds integers and the array is in strictly increasing order (there are no duplicates). To see the trick, consider what we know if \( a[i] < i \) for some \( i \). Then, because of the two facts, \( a[i-1] \leq a[i] - 1 < i - 1 \), so \( a[i-1] < i-1 \). Similarly, we can show that \( a[i-2] < i-2 \), and so on, inductively, to show that \( a[j] < j \) for \( 0 \leq j \leq i \). Similarly, we can show that if \( a[i] > i \), then \( a[j] > j \) for \( i \leq j \leq n-1 \). This gives the following algorithm:

```c
int FindEqual( int A[], int n, int &loc )
{
    int low = 0, high = n-1, mid;

    while ( low <= high ) {
        mid = (low + high) / 2;
        if ( A[mid] == mid ) {
            loc = mid; return true;
        } else if ( A[mid] < mid )
            low = mid+1;
        else
            high = mid-1;
    }
    return false;
}
```
} return false; // at this point low > high and there is
    // nothing left to search
}

Since the search range in the array is split in half each time, at most
O(log n) iterations are required. And, since O(1) operations are done for
each iteration, the total number of operations is O(log n).

11. Evaluate the following recurrence:
    \[ T(n) = 2T(n/2) + n^2 \]

Solution:
We can solve this problem by iterated expansion of the recurrence.

\[
T(n) = 2T(n/2) + n^2 \\
= 2[2T(n/2^2) + (n/2)^2] + n^2 \\
= 2^2T(n/2^2) + n^2(1 + 1/2) \\
= 2^2[2T(n/2^3) + (n/2^2)^2] + n^2(1 + 1/2) \\
= 2^3T(n/2^3) + n^2(1 + 1/2 + 1/2^2) \\
\vdots \\
= 2^iT(n/2^i) + n^2(1 + 1/2 + 1/2^2 + \cdots + 1/2^{i-1}) \\
= 2^iT(n/2^i) + 2n^2(1 - 1/2^i)
\]

The expansion bottoms out after \( i = \log n \) expansions, giving
\[ T(n) = 2^{\log n}T(1) + 2n^2(1 - 1/2^{\log n}) \]

Since \( 2^{\log n} = n \),
\[ T(n) = nT(1) + 2n^2(1 - 1/n) = nT(1) + 2n^2 - 2n \]
Therefore \( T(n) = O(n^2) \).

12. Given an empty binary search tree of integers, show the structure of the
tree after each of the values 5, 2, 4, 7, 8, 1, 3 is inserted and then show
the change to the resulting tree when 2 is deleted.

Solution:
After deleting 2

After inserting 5, 2, 4

After inserting 7, 8

After inserting 1, 3

After deleting 2

OR

5

2

4

3

1

4

8

5

3

1

4

8

5

2

7

3