CS302 – Data Structures using C++

Study Guide for the Midterm Exam Fall 2018

This document serves to help you prepare towards the midterm exam for the Fall 2018 semester.

1. What topics are to be covered by the midterm?

The midterm exam will emphasize on:

- Section 1: What should be known
- Section 2: Link-based Implementation
- Section 3: Stacks
- Section 4: Lists
- Section 5: Efficiency of Algorithms
- Section 6: Sorting Algorithms & Efficiency
- Section 7: Sorted Lists

The rest will be examined in the Final exam.

2. How should I study for the midterm?

You are expected to be knowledgeable about:

- All what is discussed inside the class and captured by the online slides.
- All the material developed and designed in the framework of your homework Assignments #1-#4.
- All the material captured by the Quizzes inside the class.

Beyond this itemized list, it is important to highlight that:

- You are expected to answer theoretical questions, for example with respect to sorting algorithms efficiency.
- You are expected to be able to demonstrate understanding of how to design an abstract data type – that is to follow the design process outlined in the lectures of the class.
- You are expected to be able to code subsets of the implementation relevant to certain data types (discussed in Section 1 – 7) or sorting algorithms. In the midterm you will not have to code on paper a particularly large piece of software. But you are definitely expected to be able to code subsets so prepare yourselves accordingly.

3. What code examples should I focus on?

You are expected to be familiar with:

- The code examples discussed inside the slides of the lectures.
- The code relevant to Assignments #1-#4.
- Further problem examples defined below and others like that.

On Recursion

1. Implement a recursive function that computes a^n, where a is a real number and n is a nonnegative integer.

On the Array-based ADT Bag Implementation

1. Provide a diagram presenting the process of removing an entry from an array-based bag.

On Stacks

1. For each of the following strings, trace the execution of the balanced-braces algorithm and show the contents of the stack at each step
a. x{{yz}}}
b. {x{y{{z}}}
c. {{{x}}}

2. Convert the infix expression a/b*c to postfix form by using a stack. Be sure to account for left-to-right association. Show the status of the stack after each step.

On Lists

1. Write specifications for a list whose operations insert, remove, getEntry, and replace always act at the end of the list.

2. Add a constructor to each of the classes ArrayList and LinkedList that creates a list containing the entries in a given array.

3. Write an array-based implementation of the ADT list that expands the size of the array of list entries as needed so that the list can always accommodate a new entry.

4. Using recursion, revise the destructor in the class LinkedList so that it deletes each node of the underlying linked chain.

On Algorithms Efficiency

1. Consider the following two loops:

```
// Loop A
for (int = 1; i<=n; i++)
      for (int j=1; j<= 10000; j++)
            sum = sum+j;

// Loop B
for (int i=1; i<=n; i++)
      for (int j=1; j<=n; j++)
            sum = sum + j;
```

What is the Big O of each loop? Design and implement an experiment to find a value of n for which Loop B is faster than Loop A.

On Sorting Algorithms

1. Write an in-place merge sort algorithm that does not require a temporary array to merge the two halves. What is the efficiency of your solution?

2. You can sort a large array of integers that are in the range of 1 to 100 by using an array count of 100 items to count the number of occurrences of each integer in the array. Fill in the details of this sorting algorithm, which is called a bucket sort, and write a C++ function that implements it. What is the order of the bucket sort? Why is the bucket sort not useful as a general sorting algorithm?

On Sorted Lists

1. Define a template interface for the ADT sorted list that is derived from ListInterface. Then define the class SortedListHasA that is derived from your new interface.

2. Consider the LinkedSorted List. Write a driver program to fully test in a manner that verifies that its whole set of operations is correct.

4. What are some indicative theoretical/design questions I should be able to respond to?

Below is an indicative set.

On ADT

1. What steps should you take when designing an ADT?

On Recursion

1. Given an integer n>0, write a recursive function countdown that writes the integers n, n-1, …, 1.

2. Write a program that uses recursion to solve the Towers of Hanoi problem.

3. Write the postfix expression that represents the following infix expression: (a*b – c)/d + (e-f)

4. What is the cost of the Towers of Hanoi with respect to the amount of moving operations?

5. Consider the following recursive function

```
int p(int x)
{
      if (x <= 3)
            return x;
      else
            return p(x-1) * p(x-3);
} // end p
```

Let m(x) be the number of multiplication operations that the execution of p(x) performs.

a. Write a recursive definition of m(x)
b. Prove that your answer to the above question is correct by using mathematical induction

On Array-based Implementation of the ADT Bag

1. Consider a bag of integers. Write a client function that computes the sum of the integers in the bag aBag.
2. Write a recursive array-based implementation of the method toVector for the class ArrayBag.

3. Add the methods union, intersection, and difference to the class ArrayBag.

On Link-based Implementation of the ADT Bag

1. Revise the destructor in the class LinkedBag so that it does not call clear but instead directly deletes each node of the underlying linked chain.

2. If headPtr is a pointer variable that points to the first node of a linked chain of at least two nodes, write C++ statements that delete the second node and return it to the system.

On Stacks

1. Write a function that uses a stack to test whether a given string is a palindrome.

2. Design and implement a class of postfix calculators. Use the algorithm given in class to evaluate postfix expressions as entered into the calculator. Use only the operators +, -, *, and /. Assume that the postfix expressions are syntactically correct.

On Lists

1. Given a nonempty list that is an instance of LinkedList, at what position does an insertion of a new entry require the fewest operations? Explain

2. How does the insert method enforce the precondition of getNodeAt?

On Algorithm Efficiency

1. Provide a graphical cmparison of typical growth rate functions

2. What order is an algorithm that has a growth-rate function:
a. $8 \times n^3 - 9 \times n$
b. $7 \times \log_2 n + 20$
c. $7 \times \log_2 n + n$

3. Suppose that your implementation of a particular algorithm appears in C++ as

```
for (int pass = 1; pass <= n; pass++)
{
      for (int index = 0; index<n; index++)
      {
            for (int count = 1; count <10; count++)
            {
                  …
            }
      }
}
```

The previous code shows only the repetition in the algorithm, not the computations that occur within the loops. These computations, however, are independent of n. What is the Big O of the algorithm? Justify your answer.

On Sorting Algorithms

1. Show that the merge sort algorithm satisfies the four criteria of recursion.

2. Trace a quick sort's partitioning algorithm as it partitions the following array: 38 16 40 39 12 27

On Sorted Lists

1. Write specifications for the operation insertSorted when the sorted list must not contain duplicate entries.

2. Describe the concept of containment in the framework of a SortedListHasA being composed of an instance of the class LinkedList.