Midterm Exam review/study guide

COMP 3760, Winter 2020

Topic areas covered on the exam

Math stuff & algorithm analysis (Lectures 1&2)

Identifying basic operations
Setting up summation formulas
Solving summation formulas
Big-O efficiency classes

Brute force/exhaustive search (Lecture 3)

Linear Search
Selection Sort
Bubble Sort
Optimization problems (TSP, Knapsack, Assignment)

Decrease and conquer (Lecture 4)

Three categories: Constant amount, Constant factor, Variable amount
Insertion sort, Generating permutations and subsets
Binary Search, Exponentiation by Squaring, Fake Coin

Divide and conquer (Lecture 5)

Difference from Decrease and conquer
The Master Theorem (analyzing D&C)
Mergesort
Binary tree algorithms

Transform and conquer (Lecture 6)

Pre-sorting algorithms ("instance simplification")

Sample questions – short answer

About 2/3 to 3/4 of the midterm will consist of "short answer" questions such as the ones in this section. These questions will be worth 1 point each. The midterm is worth 25 points, so there could be approximately 17 to 20 questions like these on the midterm.

The questions given below are meant to serve as examples for review/study purposes. This is *not* a comprehensive list, but if you can do well on all of the kinds of questions here, I think you will be in pretty good shape.

What is the basic operation in the pseudocode (or Java code) below?

```
[there would be some pseudocode or Java code here]
```

A: Could be multiple choice, could be short-answer.

What statement gets executed the most times in the pseudocode (or Java code) below?

```
[pseudocode or Java code here]
```

A: Could be multiple choice, could be short-answer.

What is the big-O efficiency class for the function shown here?

[function expression here]

A: Could be multiple choice, could be short-answer, typically a selection from some of the usual band of suspects O(1), O(logn), O(n), O(nlogn), etc.

What is the closed-form for the summation formula shown here?

[summation expression here]

A: Could be multiple choice, could be short-answer, selection from a few algebraic expressions

Here is a bit of pseudocode/Java code that does something (typically with an array). What is the output/result/etc. if the array initially contains the values [x,x,x,x,x,x]?

The following is just one example

```
Algorithm Secret(A[0..n-1])
//Input: An array A[0..n-1] of n real numbers
minval = A[0]
maxval = A[0]
for i = 1 to n-1 do
    if A[i] < minval
       minval = A[i]
    if A[i] > maxval
       maxval = A[i]
return (maxval - minval)
```

A: Could be multiple choice, could be short-answer.

What is the big-O efficiency class for the bit of pseudocode/Java code shown below?

```
i = 0
while i < n do
  j = 0
     while j < i do
       x = x + 2
       j = j + 1
   // end of while j
   i = i + 1
// end of while i
```

A: Could be multiple choice, could be short-answer. As usual, it could be any of the usual suspects. For the example currently shown above it's O(n^2).

Suppose you have an array [x,x,x,x,x,x]. What are the {first/last} two numbers that will be swapped if you sort the array using {Bubble sort, Selection sort}? Note: Insertion sort and Mergesort don't really fit into this question.

A: A pair of numbers.

Suppose you have an array A=[x,x,x,x,x]. Show the contents of A after each iteration of {Bubble sort, Selection sort, Insertion sort}. Note: Mergesort doesn't really fit into this question.

A: Show the contents of A after each step. Your answer will be N-1 copies of the array as it is being sorted, e.g.:

[x,x,x,x,x]

[x,x,x,x,x]

[x,x,x,x,x]

[x,x,x,x,x]

[x,x,x,x,x]

Which sorting algorithm is the fastest {list of sorting algorithms here: Bubble, Selection, Insertion, Merge}?

A: Multiple choice

Compare the running times of Linear Search and Binary Search. Which one is faster? Why would you ever use the slower one?

A: Could be T/F, multiple choice, short answer.

What is the {minimum/maximum} number of key comparisons that might be made by Linear Search to find a key in the array [x,x,x,x,x,x]?

A: Could be multiple choice, could be short answer.

What is the number of character comparisons made by the Brute Force String Matching algorithm with the pattern and text shown below?

```
P: {some pattern to look for}
T: {some text in which to search for the pattern}
```

A: The requirement in this case is to count what happens by tracing the algorithm with that actual pattern P and text T.

If you have a text T that is 14 characters long and a pattern P that is 4 characters long, what is the *best case* (smallest) number of character comparisons for a "Found" result by Brute Force String Matching?

A: It's 4.

If you have a text T that is 14 characters long and a pattern P that is 4 characters long, what is the *worst case* (largest) number of character comparisons that could result from the Brute Force String Matching algorithm?

A: It's 44.

If you have {some given number N} of coins in your bag, what is the {best/worst} case number of weighings for the fake coin algorithm (two-piles version)?

A: Follow the algorithm and count the weighings, being careful about best/worst case and also what happens when there is an odd number of coins.

A particular divide and conquer algorithm involves dividing the problem into {4} subparts, each of size {n/3}. The "combine" step requires F(n)={something} steps. What is the correct expression for the running time of the algorithm?

A: Could be multiple choice or short answer, looking for the answer "T(n) = aT(n/b)+F(n)" with proper values filled in for a, b, and F(n).

Given an algorithm with T(n) = aT(n/b) + F(n) (with specific values given for a, b, and F(n)), what is the big-O efficiency class of the algorithm?

A: Could be multiple choice or short answer. The idea here is to apply the "master theorem" – comparing n^logba and F(n), etc.

What is the big-O efficiency class of {many different algorithms – or *parts* of algorithms such as the "merge" part of Mergesort – that we have studied}?

A: Linear and binary search, all of the sorting algorithms, the fake coin problem, brute force string matching, binary tree algorithms, exponentiation, generating subsets and permutations, more.

What type of algorithm is {many possible question choices here}?

A: Example answers would be "Brute Force" or "Divide and Conquer" or "Decrease and Conquer" etc. Should be able to answer this for all of the algorithms we have studied.

Under what circumstances is it advantageous to pre-sort a data set?

A: Typical situation (many variations possible): Brute-force algorithm takes a certain running time, usually worse than O(nlogn) – e.g. maybe O(n^2). Pre-sorting takes O(nlogn). Is there an algorithm that runs faster than (or equal to) O(nlogn) if you can assume the data is sorted?

Another possibility is if the "what you're doing" part is going to be done a large number of times. Even if "brute force" is technically faster than sorting (e.g. O(n)), if the algorithm on sorted data is very fast, the cumulative savings eventually overcomes the initial cost of sorting. Pre-sort + binary search is the classic example of this.

{TO-DO – Small sample of T/F questions on Math/analysis, Brute Force, Decrease-and-conquer, Divide-and-conquer, Transform-and-conquer}

Sample questions – writing and analyzing pseudocode

About 1/4 to 1/3 of the exam will require writing and/or analyzing some pseudocode for a given problem or algorithm. These kinds of questions will usually be worth about 3-5 points and may consist of multiple parts. The sky's the limit here (in a sense), but here are some problems to practice with.

As you think about these problems, be sure to think not only about the solution to the problem, but also about the big-O efficiency class of your algorithm.

NOTE: Unless otherwise stated, it's always acceptable to define "helper functions" as part of your algorithms. Be sure to account for the running times of your helper functions when you do an analysis of the running time of your algorithm.

Write a decrease(by constant 2)-and-conquer algorithm to determine if a string is a palindrome (result is a boolean). Assume that the only allowable properties/methods on strings are: check the length, get the first character, get the last character, find a substring from position i to j.

Given a binary tree T, determine if ALL the leaves are at the same height (distance from the root).

Given an unsorted array of N distinct integers, find the "next-largest" value for every element in the array. For the largest element in the array, the next-largest value is "inf" (for infinity), which you can assume is a special constant that exists in your programming language. For example, if the input array is [8,6,7,5,3,0,9] then the output would be [9,7,8,6,5,3,inf]. 1) Write a brute-force algorithm to solve this problem in O(n^2). 2) Use pre-sorting to write an algorithm that will solve this problem in O(n) (plus the time required for sorting, so O(nlogn) overall).

Write a divide(by 2)-and-conquer algorithm to reverse a string (result is another string).

Given an integer array A and an integer K, determine whether A contains two numbers that add up to the value K. This problem has an “obvious” brute-force algorithm that is O(n^2). There is another algorithm that uses pre-sorting to solve this problem faster.

Write a decrease(by constant 1)-and-conquer algorithm to reverse a string (result is another string).

Make a list of all the binary strings of length N using a decrease-and-conquer algorithm. (No bit manipulations allowed here, and no loops counting over the numbers from 1 to 2^n.)

Given a binary tree T, find the length of the *shortest* path from the root to any leaf node. (Path length is the number of edges along the way.)

Given a matrix A with size NxN, determine whether A is symmetric. Symmetric means that the matrix can be reflected along the main diagonal. In other words, A[i,j] is equal to A[j,i] for all values of i and j.

Write a divide-and-conquer algorithm to count the number of vowels (a,e,i,o,u) in a string. (Need some reasonable assumptions about allowable operations on strings.)

Given a string that represents a binary number, write a divide-and-conquer algorithm to determine if the string is “balanced”, i.e. the number of 1s in the first half of the string is exactly the same OR at most one different than the number of 1s in the second half of the string.

Given a binary tree T where each node contains a number, use a divide-and-conquer algorithm to determine if the root element contains the maximum value in the entire tree.