UCF Computer Science Foundation Exam Guide

by http://www.reddit.com/user/DaffUCF/
August 17, 2014

There are no shortcuts or secrets to doing well on the exam. You need to know your stuff. This guide contains my tips, links to resources, and flash cards I used to study for the exam.

If you follow through with this guide, in addition to what you learned in your classes, you should know your stuff. Good luck!

How much time should I study?
This really depends on you and how well you understood the material in your classes. I've known some students who didn't study at all and they passed the first time. Others need to relearn entire topics. Just be honest with yourself about how well you know the topics shown below. If you're not good with many of them, then give yourself a lot of time to study.

It's best to start early. It's a lot easier to study for 2 hours every day for a month (60 hours of studying -- better than a typical student!) than it is to study for 4 hours every day for two weeks. It will give your brain more time to digest the information and it won't take over your life.

Don't leave it until Finals week before you start studying. You'll be too busy and tired with your classes!

Study in short bursts of 45 minutes, then stand up and take a break for 10-15 minutes to rest your mind. Go for a short walk, do your laundry, eat some fruit, or whatever. Get away from the paper and laptop.

If you're near the beginning of your semester, you could study well for only an hour every day. You would have 100+ hours of study (overkill!) and will probably do wonderful. That's a lot of practice for just one exam.

If you have procrastination problems, do whatever it takes to get rid of them. Internet bothering you? Unplug your modem, or leave the house and study in the library. Addicted to your mobile Reddit? Give your family/friend your phone and don't let them give it back until you're done. It's a lot easier to just get rid of the distraction than to try using your willpower to overcome it.

General Tips:

- Never leave a question blank. At least write down what you're thinking. Even if you're only given 1 point for it, it's better than nothing. Every point counts.

- For each topic shown below, first review the lectures, then read the related material from the textbook, then do at least a few exercises for practice. If you know the topic very well, you can skip reading the textbook.

  As you complete each topic, cross it off the list. You'll see how you're progressing over time. Review completed items periodically, especially if you were rusty with them.

- Specific questions aren't usually repeated. For example, if they asked a Rules of Inference question on the last exam, they *probably* won't ask it on the next exam. There's no guarantee, but historically we see this is true.

  However, certain topics are always asked. You're guaranteed recursion questions. You're guaranteed some sort of binary tree question. You're guaranteed to have a proof. You'll be asked about probabilities. And so on.

- Some people just practice the previous foundation exams. That's not bad, but might not be enough. For certain problems (such as proofs) it won't help you unless you have a good understanding of it. So take the time to read the lectures and/or textbook when you need to. Having a good understanding is more important than memorizing a handful of problems.

- When doing foundation exam practice, try giving yourself only 10 minutes to complete a question. This is to help train your brain to work quickly, because you only have 3 hours to answer around 20 questions on the exam.

- Try alternating between CS and DS topics each day. E.g. if you were planning to study for 4 weeks, don't do all CS weeks 1-2 and then DS for weeks 3-4. By the end of week 4, you may have forgotten some of the CS stuff. Mix it up, keep both topics fresh in your mind. You could read up on Stacks for CS, and then Proof by Induction for DS.

- Quizlet.com is great for creating flashcards. I'd recommend creating flashcards to help remember Big O times for common algorithms, along with anything else you may have trouble remembering (such as formulas for combinations and permutations). A list of 50+ flashcards is provided at the end of this document. Practice them every day for 10-15 minutes and you'll remember them easily.

- Find programs online to practice heap and AVL tree and Binary Search Tree inserts/deletes/etc. Just do a Google search for "AVL tree applet" and things like that. For example: http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html

  You can use these to help create practice problems for yourself. Try creating tricky insertion and deletions for AVL trees which would require multiple rotations. They probably won't give you a simple one on the exam. Know how to identify which nodes are

"A, B, C" (see Sarah's lectures).

- Practice writing your coding solutions on paper, just like you'll be doing on the real exam. It feels very different than typing it on a computer.
    - Practice EVERYTHING on paper, actually. Simulate the real exam as closely as possible.

- The night before the exam get a good amount of sleep. Have a healthy breakfast. You don't want to feel drowsy or have a headache during the exam… trust me!

Helpful Links

- http://www.cs.ucf.edu/academics/foundationexam.php
- http://bigocheatsheet.com/
- Probability -- http://www.math.uconn.edu/~wilkins/math3160s13/
- https://openlibrary.org/books/OL1087438M/Data_structures_algorithms_and_software_principles_in_C

Helpful Computer Science Lectures
Sarah: http://www.cs.ucf.edu/courses/cop3502/fall2011/Lectures/index.html

Jonathan: http://www.cs.ucf.edu/courses/cop3502/spr2011/notes/notes.html
(The PDF links work)

Personally, I never understood Recurrence Relations until I read Jonathan's slides on the topic.

Arup: http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3502/

Helpful Discrete Structures Lectures
Arup: http://www.cs.ucf.edu/~dmarino/ucf/transparency/cot3100/

Also, the textbook is very good at explaining this subject... In my humble opinion!

Other Discrete links:
http://www.site.uottawa.ca/~lucia/courses/2101-10/
http://faculty.ucmerced.edu/mhyang/course/cse115/
http://www.cs.ucf.edu/courses/cot3100/spr2010/

The Topics

Below are the topics for the exam. This list was provided by Arup on the CS web site, and I've added my notes & tips.

Computer Science Part A

Recursive Coding

i. Need a terminating condition
ii. Need an algorithm for non-terminating case.
iii. In particular, you must reduce a question to “smaller” instances of the same question.
iv. Do not try to think of an iterative solution!!!
v. Towers of Hanoi solution and recursion
vi. Permutation
vii. Floodfill

Daff’s Tips:

- It’s unlikely they will ask you to code these exact problems (such as Floodfill), although possible. The reason he asks you to understand this is because if you know how to do Floodfill, you can figure out other questions they might ask you.

- Even if you’re not sure how to do it, at least write down something to get some easy points. For example, if you look over the grading guides for the previous exams, you’ll notice you usually get 2 points just for checking if the head of a linked list is null. Do it! Never give up easy points. Many people only pass or fail the exam by a few points. Every point matters.

Summations and Algorithm Analysis

i. Break them down into multiple summations if necessary
ii. Evaluate each of those using summation formulas.
iii. Remember that indices of summation are important.
iv. The n in the formula is JUST a variable!!!
v. Deriving recurrence relation from code
vi. Using iteration to solve recurrence relations

Daff’s Tips:

- Just make sure to do a lot of practice problems. You’ll understand these by doing them. Also for recurrence relations, I found Jonathan’s lecture notes to be a godsend.

- Memorize the five Summation formulas from Sarah’s CS1 slides. Arup also has the formulas in his notes (he also has a 6th addition formula).

i. Stacks

a. Converting infix to postfix expressions
b. Evaluating postfix expressions
c. Array Implementation
d. Linked List Implementation

ii. Queues
a. Array Implementation
b. Linked List Implementation

Tips:
- Infix and postfix are pretty easy because you can check the answer to make sure it works. Just do some practice problems for converting them in both directions. Also review Arup's code on how to implement stacks and queues.

4. Advanced Data Structures - Tracing
i.Binary Trees
a. Traversals
ii. AVL Trees
a. Insertion
b. Deletion

Tips:
- Review the lectures to make sure you understand how the AVL rotations work. Then practice, practice, practice. Try giving yourself complicated and weird insertion/deletion cases and see if you can do them.

iii. Hash Tables
a. Hash Function Properties
b. Linear Probing Strategy
c. Quadratic Probing Strategy
d. Separate Chaining Hashing

Tips:
- These problems are usually straightforward. Just understand how they work, the three different strategies, and the Big O times and space complexity.

iv. Binary Heaps
a. Insertion
b. Delete Min/Max

Tips:
- Youtube has helpful videos on Heapify and binary heaps in general.

Conversions
i. Conversion from binary to decimal.
ii. Conversion from decimal to binary.
iii. Conversion between binary, octal and hex.

Tips:

- Just practice these and know how to do them. Simple division and multiplication, really. These are easy points if you get asked.

Computer Science Part B

1. Algorithm Analysis
i. Known Data Structures
ii. Best, Average, Worst Cases
iii. Based on various implementations
iv. New Problem Analysis

Tips:

- Study and understand how these really work. If you do, you can figure out the Big O in your head and reasoning. But just to help, I'd recommend writing the Big Os on flashcards and memorizing them too.

Timing questions
i. Set up correctly with an unknown constant
ii. Solve for the constant.
iii. Use direct formula to answer the question
iv. For loop questions, write out summations

Tips:

- Practice the examples and make sure you know how summations work. Summations might look weird but just think of them as a For loop.

Linked Lists – Coding
i. How to allocate space for a new node (malloc)
ii. When to check for NULL
iii. What free does
iv. Iteration vs. Recursion
v. Insertion
vi. Deletion
vii. Structural Modification

Tips:

- Hopefully you had a lot of coding practice with these from your class. Review Arup's sample programs if you need more help understanding linked lists.

Binary Trees – Coding
i. How to allocate space for a new node (malloc)

ii. When to check for NULL
iii. What free does
iv. Using recursion with trees
v. Computing sum of nodes
vi. Computing height
vii. Other variants

Tips:

- These questions usually require you to have a good understanding of moving through the different levels of a tree. Often times the questions require you to use the Mod operator to know if something is even or odd.

- Recursion problems are usually fairly short. If you're coding a 20-line solution, you're probably overcomplicating it.

Sorting
i. Insertion Sort
ii. Selection Sort
iii. Bubble Sort
iv. Merge Sort (Merge)
v. Quick Sort (Partition)

Tips:

- Study how these work and how to code them. They usually don't ask questions about the same sort twice in a row. So if they asked students to run through a sample Selection Sort on the last exam, then on the next exam they will probably ask about Bubble or Insertion sort. But nothing is guaranteed -- so know them all.

Discrete Structures Part A

Summations and Induction
i. Use of standard summation rules + meaning of a summation.
ii. Divisibility Problems
iii. Matrix Problems
iv. Other - any assertion for non-negative ints.
v. How to plug into an inductive hypothesis and step.
vi. Some "tricks"
vii. How to deal with inequalities
viii. Standard Algebra rules

Tips:

- Hopefully you are good with your Algebra because many times these questions will require an algebra technique, such as Completing The Square. Other common things like grouping like terms are often used.

- These problems are usually worth 15 points, so they are important. Practice and read the textbook if needed.

Logic
i. Meaning of and, or, not, implication and contrapositive, xor
ii. Truth Tables
iii. Logic Laws
iv. Laws of Implication
v. Quantifiers (for all and there exists)

Tips:
- You are given the table of laws for the exam. Use them and double-check your work to make sure you're using the correct law.

Sets
i. Defn of empty set, subset, intersection, union, complement, set difference, set product, and power set
ii. Set Tables
iii. Proof by Contradiction
iv. Direct Proof
v. Disproof by Example

Tips:
- The Rosen textbook and Arup's lectures provide a lot of examples for these. If you are given a problem that can be solved with a direct proof, consider yourself blessed.

- You must understand how proofs work in order to do well on the DS portion, as usually there are multiple questions that depend on this type of reasoning. Don't take proofs lightly!

Number Theory
i. Definition of division
ii. Euclid's algorithm (and extended Euclid)
iii. Unique Prime Factorization + Proof
iv. Least Common Multiple
v. Modular Arithmetic
vi. Proof of an Infinite Number of Primes
vii. Proof of the irrationality of sqrt(2)

Tips:

- Sometimes the Number Theory problems can be quite easy (like a simple GCD problem) or can be very confusing. Just pay attention to what he is asking you and try to think through it. These are hard to prepare for because they could ask so many things.

- Although he likely won't ask you for the exact proof of the irrationality of sqrt(2), the reason you should study it is because if you can do that one, you can probably do others that he might ask.

Counting
i. Meaning of combination vs. permutation
ii. Sum principle
iii. Product principle
iv. Number of subsets of a set
v. Binomial Theorem
vi. Inclusion-Exclusion principle
vii. Counting by subtracting from the whole
viii. Combinations with Repetition

Tips:

- You have to really think about what's going on and what the question is asking for.

- Know the four formulas:
    - Combinations without repeat
    - Combinations with repeat
    - Permutations without repeat
    - Permutations with repeat
    - See this helpful calculator (also shows formula): http://www.mathsisfun.com/combinatorics/combinations-permutations-calculator.html

Probability
i. Sample Space
ii. Two Counting Questions
iii. Inclusion-Exclusion Principle
iv. Tree Diagrams
v. Conditional Probabilities
vi. Bayes Law
vii. Binomial Distribution
viii. Independent Events
ix. Mutually Exclusive Events

Tips:

- If you've taken Statistics already, it will help you with these problems.

Relations
i. Definition of a relation
ii. Reflexive, irreflexive, symmetric, antisymmetric, & transitive
iii. Equivalence relation
iv. Partial Ordering relation
v. Relation composition
vi. Inversion
vii. Use same proof techniques as with sets

Tips:

- No tips except to just practice and be sure to provide your reason along with your answer on the exam questions. These are usually some of the easiest points on the DS exam.

Functions
i. Definition of a function
ii. Domain, Co-Domain, Range
iii. Inversion
iv. Injection, surjection and bijection
v. Function Composition

Tips:

- Inversion is pretty easy with the trick of swapping the X and Y variables, solving, and then swapping them back around. You can find examples of doing this on Youtube, search for "inverse functions".

General Advice from Arup
A. Do what you think are the easy questions first.

B. Only write what you think you know, don't make stuff up!!!

C. Don't forget to think. This test should not be mechanical, try to be creative. If you see a question you've never seen before, think about "subproblems" that may be easier to attack. See what you can deduce based on the given information, even if the question doesn't ask for it.

D. Read over all of your proofs when you are done.
Make sure you are specific. See if your proofs make sense.

Flash Cards

Injective Function (one-to-one):
f(a) = f(b) -> (a=b) for all a & b in domain f
Never more than one incoming arrow, but not always an arrow from left to all of right values
if A ->B then |A| <= |B|

Surjective (onto):
For all b in B, there exists an a in A such that f(a) = b
Multiple arrows can go to an answer on the right, and every value on right will have an arrow going to it.
if A->B then |A|>=|B|

Bijective:
Both injective and Surjective. One-to-one and only functions to have inverses
if A->B then |A|=|B|

What value is in every set:
Null

A = {1} B = {2}
AUB = ?:
{1,2}

A = {1} B = {2}
AnB = ?:
{NULL}

Assume: p, p->q
Then q:
Modus Ponens

Assume p->q, q->r
Then p->r:
law of syllogism

Assume p->q, -p
Then -q:
Modus Tollens

A|B means:
A divides B so A is a factor of B

A|B and A|C implies?:
A|(B+C)

What are the three parts of Inductions:
Base Case, Inductive Hypothesis, Inductive Step

A = R(mod B) is equivalent to:
A = Bn + R

Reflexive:
aRa for every element a in A

Irreflexive:
a does not Ra for every element a in A

Symmetric:
if aRb -> bRa for all a, b in A

Antisymmetric:
if aRb and bRa -> a=b for all a,b in A

Transitive:
if aRb & bRc -> aRc for all a,b,c in A

[x] = {a|a is an element of A ^ aRx} is an example of:
Equivalence classes

Big O best and worst case of Insert of linked list:
Best: O(1)
Average: O(n)
Worst: O(n)

Big O best and worst case of Quick sort:
Best: O(nlogn)
Average: O(nlogn)
Worst: O(n^2)

Big O best and worst of hash table access of value:
Best: O(1)
Worst: O(n)

Big O best, average, worst of binary tree search:
Best: O(1)
Average: O(lgn)
Worst: O(n)

Big O of Bubble sort:
Best: O(n)
Average & Worst: O(n^2)

Big O of Merge sort:
O(nlgn)

Definition of set difference:
n(negC U negB) becomes - (CnB)

Definition of implication:
(q->r) == (-q v r)

Equivalence relation:
reflexive, symmetric, and transitive

Partial Ordering Relation:
Reflexive, Anti-symmetric, and transitive

Binary Search on sorted list: Average and worst case, Space time complexity:
Avg: O(logn)
Worst: O(logn)
Space complexity: O(1)

Quicksort:
Best: O(n log(n))
Avg: O(n log(n))
Worst: O(n^2)
Space complex. worst: O(n)

Mergesort:
Best: O(n log(n))
Avg: O(n log(n))
Worst: O(n log(n))
Space worst: O(n)

Heapsort:
Best: O(n log(n))
Avg: O(n log(n))
Worst: O(n log(n))
Space worst: O(1)

Bubble Sort:
Best: O(n)
Avg: O(n^2)
Worst: O(n^2)
Space: O(1)

Insertion Sort:
Best: O(n)
Avg: O(n^2)
Worst: O(n^2)
Space: O(1)

Selection Sort:
Best: O(n^2)
Avg: O(n^2)
Worst: O(n^2)
Space: O(nk)

Bucket Sort:
Best: O(n+k)
Avg: O(n+k)
Worst: O(n^2)
Space: O(nk)

Radix Sort:
Best: O(nk)
Avg: O(nk)
Worst: O(nk)
Space: O(n+k)

How does Selection Sort work?:
1. Finds the smallest element in the list and swaps it with the element in the first position.
2. Then it finds the second smallest element and swaps it with the element in the second position.
3. It does this until we reach the last position.

How does Insertion Sort work?:
1. Starting with the second element, you go one-by-one and insert it (by swapping) into the already sorted list to its left in the correct order.
2. So by the time you reach the end of the list, it will put the last element where it should be and then the list is sorted.

How does Bubble Sort work?:

1. Going left to right, you compare consecutive elements.
2. Whenever two are out of place, you swap them. At the end of a single iteration, the max element is in the last spot.
3. So Joe -> Bob would be swapped to Bob -> Joe.
4. This repeats n times.
5. On each pass, the maximal element is moved to its correct spot at the end, as if the maximum is "bubbling up".

How does Merge Sort work?:
1. Arrays can be broken into half to form two smaller arrays. You can continue to do this if needed.
2. When you have two arrays, keep track of the smallest value in each one that hasn't been placed in order in the larger "Merged" array yet.
3. Compare the two smallest values from each array, place the smallest in next location in Large array.
4. Then you adjust the minimum value marker in the array the smallest element was removed from.
5. Continue until large "Merged" array is full.

How does Quicksort work?:
1. Randomly pick a pivot value (or best: pick the 1st, last, and middle element... and then go with the median value)
2. Compare all the rest of the elements to this value
3. If they are greater than this value, put them to the right of it.
4. If they are less than this value, put them to the left.
5. Put a "Low" counter at the first element, "High" counter at the last
6. Advance the Low forward until a value higher than pivot is found, and advance the High backward until a value lower than pivot is found.
7. When these two conditions are met, swap these numbers and repeat the process.
8. When the Low and High counter line up, swap their value with the pivot.
9. Now repeat the process for each partition; the left part and right part.

In general, for k successes in n Bernoulli trials we have a probability of:
C(n,k)*p^k*q^(n-k)

An equivalence relation is a relation that is:
reflexive, symmetric, and transitive

What is Open Addressing in a Hash table?:
Every hash table entry contains only one key. If there is a conflict, we use Linear Probing or Quadractic Probing to place the new key into an empty slot.

What is Separate Chaining in a Hash table?:

Every hash table entry contains a pointer to a linked list of keys that hash to the same entry.

What is Linear Probing?:
The probing continues sequentially through the table and wraps after the last slot (slot m-1) to the beginning (slot 0).
h(k,i) = (h(k) + i) mod m, where m is the size of the table

What is Quadratic Probing?:
Similar to linear probing except it examines cells 1,4,9 and so on away from the original probe point because its formula has i^2
h(k,i) = (h(k) + i^2) mod m, where m is the size of the table.

What two conditions must be met to ensure Quadratic Probing always finds an empty location (if one is available)?:
The table size must be a prime number AND the table must be at least half empty. Otherwise it could end up in a loop, never able to find an available spot even though one exists.

How do you expand a hash table if needed?:
1. Pick a prime number that is ~2x as large as the current table size
2. Use this number to change the hash function
3. Rehash ALL the values already stored in the table
4. Then you can now begin hashing new values to be stored.

How are linked lists used in separate chaining?:
In a collision you just insert new items to the front of the corresponding linked list.

A binary tree is full if:
each node is either a leaf or has exactly two child nodes.

A binary tree is complete if:
all levels except possibly the last are completely full, and the last level has all its nodes to the left side

For AVL trees, when choosing the nodes A, B, and C, remember they are always...:
on the longest path to the bottom of the tree. E.g. this means that when we find an imbalanced node after deleting, the node to the opposite side is guaranteed to be down the longer path

A descending list would be very slow in what kind of sort?:
Insertion sort

When you delete from a min heap, you delete the root (top) node. What do you replace it with?:

You replace it with the last link in the tree, i.e. the last node added in the tree or array. Then you percolate down (if needed) by swapping with the minimum child until it is in the right spot.