CSE 2011A Fall 2019 – Exam fun
Instructor: Jeff Edmonds

Questions:

1. First Order Logic
2. ADT
3. Software Eng
4. Pointers
5. Loop Invariants
6. Running Time
7. BigOh Notation and Sums
8. Recursive
9. Balanced Binary Search Trees
10. Heaps
11. Hash Tables
12. Probabilistic Alg
13. Graphs
14. Linear Programming (Network Flows)
15. Greedy Alg
16. Dynamic Programming
17. NP

Time 3 hrs.
Keep your answers short and clear.
Do not repeat the question.
USE A PEN OR DARK PENCIL SO THE SCAN IS CLEAR
DON'T WRITE OUTSIDE SPECIFIED AREAS.
DON'T WRITE OUTSIDE SPECIFIED AREAS.
No question needs more than 4 or 5 sentences.

1. First Order Logic: Use Jeff's game to prove or disprove:
   $\exists y,\ \forall x,\ x \times y = 0$, ie "There exists a y such that for all x, $x \times y = 0$"
   What does this sentence say about mathematics? ie Meaning.

   - Answer: This states that there is a multiplicative zero.
     Jeff uses the following game to prove a first order logic statement. Read the expression left to right. If a variable is quantified with an exists ($\exists$), then the prover produces an object for that variable. If it quantified with a forall ($\forall$), then the adversary does. The prover wins the game, if the inner statement is true. The statement is true, if the prover can always win. The proof of our example is as follows.

Proof of $\exists y,\ \forall x,\ x \times y = 0$:
Let y be 0.
Let x be an arbitrary real number.
Note that $x \times y = x \times 0 = 0$.

2. What are the Abstract Data Types we learned, what are their operations, how are they used, how are they stored, what are their loop invariant, what are their algorithms, what are their running times. ...

   - Answer:
     Talk about
     (a) Positions vs Pointers vs array indexes
     (b) (linked list, array) Stack
     (c) (linked list, array) Queue
     (d) (heap) Priority Queue
     (e) (hash tables) Dictionary
     (f) (matrix, pointers) Graph
     (g) (pointers) Rooted Tree
     (h) (Bal BST) Ordered searchable list with insert and delete
     (i) (pointers) Union find data structure
     (j) Linked list
     (k) Array
     (l) Balanced binary search tree
     (m) Matrix
     (n) Pointers
     (o) Heaps
     (p) Hash table
     (q) Depth first search
     (r) Breadth first search
     (s) Dykstra's algorithm
     (t) Network flows
     (u) Linear programming
     (v) Minimal spanning tree
     (w) NP-completeness

3. Software Eng:

   (a) Software Eng: $(7 = 1 + 1 + 1 + 4$ marks)
      i. Stacks, Priority Queues, Dictionaries, and so on are all examples of what? Give its three word name.
      ii. A *Stack* is an ordered list of objects in which the operations acting on it are restricted to *Pushing* a new object on its *top* end, *Popping* and returning the most recently pushed object from the top, and determining if the stack is empty.
      What is this description an example of?
      iii. What does this such a description specifically not specify?
      iv. Give four software engineering principles that this facilitates. Explain.

      - Answer:
        i. Abstract Data Types (ADTs)
        ii. System Invariant, contract, List of Methods.
        iii. An ADT does not specify how the data are stored or how the operations are implemented.

iv. Practices:
   - Modularity: Breaking complex software systems into distinct modules where each module has a well-specified job and modules communicate through well-specified interfaces.
   - Encapsulation: Each object reveals only what other objects need to see. Internal details are kept private:
     This allows the user and the implementer of the ADT not to worry about the other's job or be influenced by small changes in what the other is doing. This allows the programmer to implement or change the object as she or he wishes, as long as the requirements of the abstract interface are satisfied.
   - Facilitates reasoning about, conceptualization, maintenance and designing large systems of many interacting data structures.
   - Readable and understandable: Allows correctness to be verified, and software to be easily updated.
   - Correct and complete: Works correctly for all expected inputs
   - Robust: Capable of handling unexpected inputs.
   - Adaptable: All programs evolve over time. Programs should be designed so that re-use, generalization, and modification is easy.
   - Portable: Easily ported to new hardware or operating system platforms.
   - Efficient: Makes reasonable use of time and memory resources.

(b) (3 marks) From the book we got the concept of a *position*.

i. What was position an example of?

ii. Abstractly, what did it represent?

iii. Then in the assignment we *instantiated* it. What data structures did it become?

- Answer: Position was an interface.
  Abstractly it is just a position within a data structure.
  It became a (pointer to) a node of our binary tree.

4. Pointers: Write code to play with pointers to play with linked lists, trees, ...

5. Loop Invariant:

(a) What are three most important steps in proving a program works using loop invariant. (Not counting defining the LI, the exit cond, or anything to do with progress.) Write as "This and this gives you this and this".'

- Answer:
  Establishing the Loop Invariant: $\langle pre{-}cond \rangle$ & $code_{pre\text{–}loop} \Rightarrow \langle loop{-}invariant \rangle$
  Maintaining the Loop Invariant: $\langle loop{-}invariant' \rangle$ & *not* $\langle exit{-}cond \rangle$ & $code_{loop} \Rightarrow \langle loop{-}invariant'' \rangle$
  Ending: Obtaining the postcondition $\langle loop{-}invariant \rangle$ & $\langle exit{-}cond \rangle$ & $code_{post\text{–}loop} \Rightarrow \langle post{-}cond \rangle$

(b) What is a loop invariant and what does it mean to maintain it?

- Answer: A loop invariant is a picture of that is true at the top of the loop. Maintaining it means proving that if is true at the beginning of an iteration it is true at the end and hence at the beginning of the next iteration.

(c) What is the loop invariant for binary search? How do you maintain this loop invariant while making progress?

- Answer: LI: If the sought after key is in the list, it is in the sublist $L(a, b)$. Compare it to the middle $L((a + b)/2)$ and keep one half.

(d) Give the loop invariant and follow each of the three key steps in defining an iterative algorithm that on input M outputs $S = \sum_{i=1}^{M} i^2$. (Each of these steps is of the form "This and this gives you this and this" is not the exit cond, or anything to do with progress.)

- Answer:
  Loop Invariant: $s = \sum_{i=1}^{m} i^2$.
  Establishing the Loop Invariant: Setting $m = 0$ and $s = 0$ gives that $s = \sum_{i=1}^{0} i^2 = 0$.
  Maintaining the Loop Invariant: Let m_t and s_t be the values of m and s at the top of the iteration and m_{t+1} and s_{t+1} after going around again.
  $\langle loop{-}invariant' \rangle \Rightarrow s_t = \sum_{i=1}^{m_t} i^2$.
  $code_{loop} \Rightarrow m_{t+1} = m_t + 1$ and $s_{t+1} = s_t + {m_{t+1}}^2$.
  $\Rightarrow s_{t+1} = \sum_{i=1}^{m_t} i^2 + {m_{t+1}}^2 = \sum_{i=1}^{m_t} i^2 + (m_t + 1)^2 = \sum_{i=1}^{m_t+1} i^2 = \sum_{i=1}^{m_{t+1}} i^2 \Rightarrow \langle loop{-}invariant'' \rangle$
  Ending:
  $\langle loop{-}invariant \rangle \Rightarrow s = \sum_{i=1}^{m} i^2$.
  $\langle exit{-}cond \rangle \Rightarrow m = M$.
  $code_{post\text{-}loop} \Rightarrow S = s$.
  $\Rightarrow S = s = \sum_{i=1} Mi^2$.
  $\Rightarrow \langle post{-}cond \rangle$

6. Running Time: I give you code and you tell me the $\mathcal{O}(running\ time)$.
   (a) Multiple loops
   (b) logs
   (c) Recurrence relations
   (d) Complexity measured wrt size and size is the number of bits to write down the input.
   (e) Searching for solution vs verifing solution.

7. BigOh Notation and Sums
   (a) Give the exists forall statement defining BigOh
   (b) Does $\Theta(1)$ include $sin(n^2) + 5$? Explain this class of functions.
   - Answer: The n^2 was a trick because the sine is always between -1 and 1. Hence, $f(n) \in \Theta(1)$. Though it changes with n, the value is always bounded between the constants 4 and 6 and hence is said to be constant.
     Not asked for: This would be the same for $f(n) = \{$ 1 if n is odd, 2 if n is even $\} \in \Theta(1)$.
     Not asked for: If $f(n) = sin(n^2)$, then $f(n)$ would not be in $\Theta(1)$ because things in $\Theta(1)$ should be positive for big n.

   (c) **Summation:** Approximate the sum $\Theta(\sum_{i=1}^{n} f(i))$. Which result did you use and how did you use it? What type of sum is it?
   i. $f(n) = 5 \cdot n^{5.5} \log^{100}(n)$
   ii. $f(n) = 5 \cdot 3^{4n} n^{100}$
   iii. $f(n) = 5 + sin(n^2)$
   - Answer:
     i. **Class:** $f(n) = 5 \cdot n^{5.5} \log^{100}(n) = \Theta(n^{5.5} \log^{100}(n))$. For big n, this is bounded between $n^{5.5}$ and $n^{5.6}$. Hence, $f(n) \in n^{\Theta(1)}$. This is a polynomial function.
     Sum: $b^a = 1$ and $d = 5.5 > -1$. Hence, the sum is arithmetic and half the terms are approximately equal. Hence, $\sum_{i=1}^{n} f(i) = \Theta(n \cdot f(n)) = \Theta(n^{6.5} \log^{100}(n))$.
     ii. **Class:** $f(n) = 5 \cdot 3^{4n} n^{100} = \Theta(3^{4n} n^{100})$. For big n, this is bounded between 2^{1n} and 2^{100n}. Hence, $f(n) \in 2^{\Theta(n)}$. This is an exponential function.
     Sum: $\sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} \Theta(3^{4i}) n^{100}$. $b^a = 3^4 = 81 > 1$, the sum is geometric increasing and dominated by the last term. Hence $\sum_{i=1}^{n} f(i) = \Theta(f(n)) = \Theta(3^{4n} n^{100})$.

iii. **Class:** $f(n) = 5 + sin(n^2)$. The n^2 was a trick because the sine is always between -1 and 1. Hence, $f(n) \in \Theta(1)$. Though it changes with n, the value is always bounded between the constants 4 and 6 and hence is said to be constant.
Not asked for: This would be the same for $f(n) = \{$ 1 if n is odd, 2 if n is even $\} \in \Theta(1)$.
Not asked for: If $f(n) = sin(n^2)$, then $f(n)$ would not be in $\Theta(1)$ because things in $\Theta(1)$ should be positive for big n.
Sum: $b^a = 1$ and $d = 0 > -1$. Hence, the sum is arithmetic and half the terms are approximately equal. Hence, $\sum_{i=1}^{n} f(i) = \Theta(n \cdot f(n)) = \Theta(n)$.

8. Recursion:

   (a) Give a recursive algorithm for summing up all the values in a binary tree.

   - Answer: If the given tree is the empty tree then the answer is zero.
     Otherwise, give one friend the left subtree and another the right.
     They give you the sums in these respective trees.
     Our answer is the sum of these plus the value at the root.

   (b) Describe a Merge Sort like algorithm whose running time is given by $T(n) = 3T(n/3) + \Theta(n)$.

   - Answer: I split the list to be sorted into 3 parts of size $n/3$ each. I have three friends (ie recurse to the same program). Each sorts one of the three parts. The size of their input is $n/3$ so it takes them $T(n/3)$ time each. I merge these three sorted lists into one. This takes me time $\Theta(n)$. This is the time of my stackframe (excluding recursive calls to the same program).

   (c) Solve this recurrence relation $T(n) = 3T(n/3) + \Theta(n)$. How does it compare with that of Merge Sort?

   - Answer: Here $\frac{\log a}{\log b} = \frac{\log 3}{\log 3} = 1 = c$. The top level work is n^1. The number of basecases is n^1. This means that the $\log n$ levels all have about the same amount of work, giving $Q(n) = \Theta(n \log n)$. Within a multiplicative constant, this is the same as Merge Sort.

   (d) When making a recursive call in any recursive program, what are the requirements on the instance that you give it?
   When describing a recursive algorithm or proving that it works, would Jeff want you to trace out the tree of stack frames? Why or why not.
   How does he say you should relate to your (recursive) friends.

   - Answer: The subinstances must meet precondition and be smaller.
     Definitely do not trace the computation because it is too confusing. Jeff wants you to trust friends and not micro manage them.

9. Balanced Binary Search Trees: What are their operations, how are they used, how are they stored, what are their loop invariant, what are their algorithms, what are their running times, trace out an example.

10. Heaps: What are their operations, how are they used, how are they stored, what are their loop invarients, what are their algorithms, what are their running times, trace out an example.

11. Hash Tables: What are their operations, how are they used, how are they stored, what are their loop invariant, what are their algorithms, what are their running times, trace out an example.

12. Probabilistic Algorithms:
    We say that problem P is deterministically computable in time 100 iff
    $\exists A \; \forall I \; A(I) = P(I)$ and $Time(A, I) \leq 100$.
    In the first order logic game, the prover first gives the algorithm A and then the adversary, knowing A, gives the worst case input I.

    (a) Explain how this is used with Hash Tables

(b) As done above for deterministic algorithms, give the first order logic statement to state that there is probabilistic algorithm that always gives the correct answer and has expected time at most 100 per operation.
Use your own words to explain how the associated game is different.

- Answer: $\exists A\ \forall I\ \ (\forall R\ A_R(I) = P(I)$ and $\text{Exp}_R[Time(A_R, I)] \leq 100)$.
The prover gives algorithm A without knowing the input. Knowing A but not the random $R = \langle a, b\rangle$, the disprover gives an input I that he thinks is a worst case for this algorithm. No matter what the random $\langle a, b\rangle$ is, the algorithm always is correct. However, for every input, over the choice of $\langle a, b\rangle$, the expected time is $T(|I|) = \Theta(1)$. For each input, there are worst case coin flips, but there are not worst case inputs.

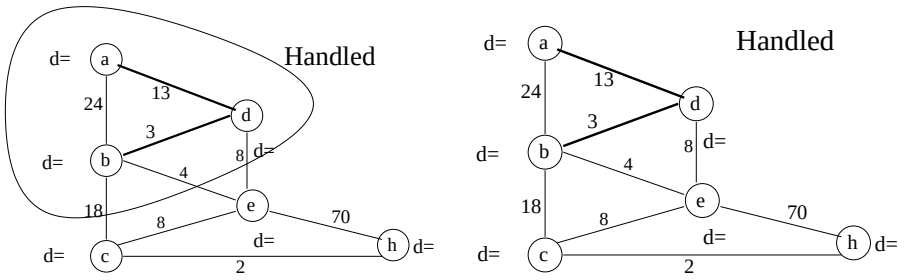
13. Graphs:

(a) How to store it

(b) Depth First Search (Stack) (Linear Order)

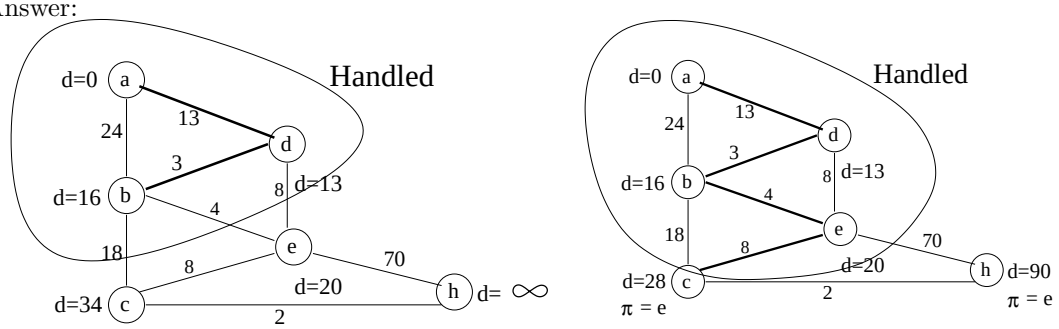
(c) Breadth First Search (Queue)

(d) Dijkstra's Algorithm (Priority Queue on priorities $d(v)$:
Consider a computation of Dijkstra's algorithm on the following graph when the circled nodes have been handled. The start node is a. On the left, give the current values of d. (Be sure to think about what the loop invariant says about the values d.)



On the right, change the figure to take one step in Dijkstra's algorithm. Include as well any πs that change.

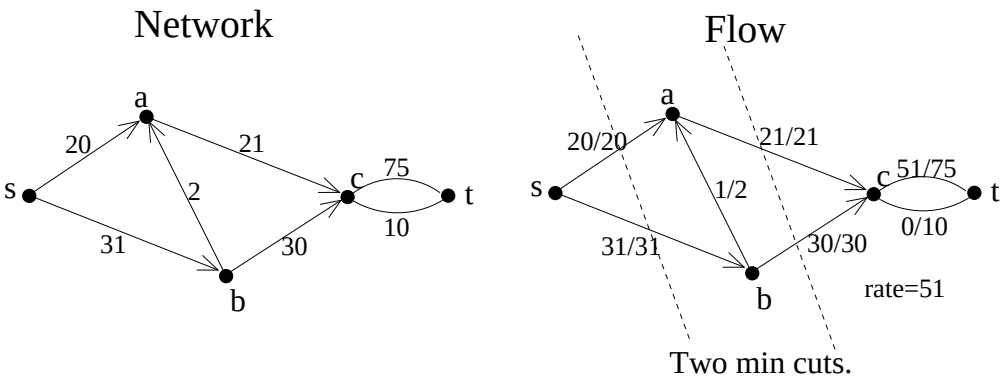
- Answer:



(e) Network Flow: Consider the following network with source s and terminal (or sink) t. The label on an edge is the capacity of the edge.

i. Give flow along each edge in a maximum flow in this graph.

ii. Give a minimum cut which can be used to argue that the flow cannot be more than stated.



- Answer:

(f) Network Flow:

i. Describe the hill climbing algorithm to finding such an optimal flow.

ii. How can you prove to your boss that you can achieve a flow as good as you claim?

iii. How can you prove to your boss that it is impossible to achieve a better flow?

- Answer: Each iteration either find a better flow or a matching cut.
  The flow you found witnesses the fact that a flow this good is obtainable.
  A cut with equal value witnesses the fact that no flow can be bigger than this.

14. What is a Linear Program and how do you solve it.
What is Network Flows and how is it an example of Linear Programming?

15. Greedy Algorithim:

(a) What is a greedy alg?

(b) What is its loop invariant?

(c) What is its running time?

(d) Give a Greedy Algorithim for something.

16. Consider the Dynamic Programming Algorithm for finding shortest paths in a graph.

(a) If this an iterative algorithm, give it's loop invariant.
If this a recursive algorithm, what are the subinstances recursed on (given to my friends).

- Answer: This is an iterative algorithm. The loop invariant is that the answer is saved for each u and v for paths with $\ell/2$ edges.

(b) Give one line to compute $Dist(u, v, \ell)$ giving the length of the shortest path from u to v with at most ℓ edges.

- Answer: $Dist(u, v, \ell) = Min_k Dist(u, k, \ell/2) + Dist(k, v, \ell/2)$.

(c) What is its running time of the entire algorithm to find $Dist(u, v, n)$ for each pair $\langle u, v \rangle$?

- Answer: $\Theta(n^3 \log n)$.

17. NP

(a) How do you convince your boss that the computational problem he wants you to solve is likely too hard to solve in general?

- Answer: Show that is is NP-complete (or even worse undecidable).

(b) What does it mean for a problem P to be in NP (Non-Deterministic Polynomial Time)?

- Answer: Given an instance and a solution, it is easy to test its validity.
  $\exists$ poly time verifier $Valid$ such that
  $\forall I$, $[I \in P$ iff $\exists S\ Valid(I, S)]$.

(c) Why is 3-Col in NP?

- Answer: Because given a graph G as input and a colouring as a solution S from your fairy god mother, it is easy to check that S is a valid colouring of your graph G.

(d) What does it mean for a problem P to be NP-Complete.

- Answer: It is in NP and
  it is as hard as any problem in NP and is in NP,
  i.e. $\forall P' \in NP\ P' \leq_{poly} P$.

18. What did Godel proof about proof systems?
    Give few sentence intuition of the proof.

- Answer: Given any proof system there is a statement that is either not proved or is proved incorrectly.
  Godel's proof: If there was a proof system S, then the sentence Φ = "Φ does not have a proof in S" is a problem. If Φ is true, then what its says is that it does not have a proof. If it is false, then what its says is that it does have proof.
  Turing's proof: Such a proof system would lead to an algorithm that would decide whether a math sentence is true or not which would lead to an algorithm that decides the halting problem.