CS310: Advanced data structures and algorithmns

Sample questions – Fall 2017

Instructor – Nurit Haspel

1. Runtime analysis (from Midterm F16)

   a. Given the following piece of code:

   ```
   public static int someFunction(int n)
   {        int i, j, sum = 0;
            for (i=0; i < n; i+=2)  /* loop 1 */
                sum += i*i;
            for (i=0; i < n; i++)  /* loop 2 */
                for (j=1; j < n; j=j*2)  /* loop 3 */
                   sum += i*j;
            return sum;
   }
   ```

   What is the run time of:

   (a) Loop 3

   $O(\log n)$ (j is growing in multiples of 2, so after at most $\log n$ iteration it will reach n)

   (b) Loop 2 (including the runtime of loop 3).

   Loop 2 runs loop 1 n times, so overall $O(n \log n)$

   (c) Loop 1.

   $O(n)$ (notice that while the loop runs $\frac{n}{2}$ times, the big-O expression is the simplest, no coefficients).

   (d) The whole function, explain briefly.

   $O(n \log n)$. First loop 1, followed by loop 2 (3 is nested within 2). So the overall runtime is the sum of both. Notice that the correct expression is $O(n \log n)$ and not $O(n + n \log n)$ because we use the simplest form, no coefficients or polynomials, and $O(n \log n)$ dominates $O(n)$.

   b. An algorithm takes 15 seconds to solve a problem of size 1000. If the algorithm is quadratic – i.e., runs as $O(N^2)$, how large a problem can be solved in 60 seconds?

   (a) 2000
   (b) 4000
   (c) 6000
   (d) none of the above

   The answer is 2000. A quadratic algorithm takes four times more when the input size is doubled.

2. Java Data Structures (from Midterm F16): For (a-c) below, determine what is the best data structure to use out of the ones we discussed in class: List, Set, Map. If more than one acceptable answer exists, use the most efficient one (with respect to runtime) that has the power you need. Also say what type you use: LinkedList vs. ArrayList, HashSet vs. TreeSet or HashMap vs. TreeMap. Please provide an explanation.

(a) You are working on a banking program. Each day, the checks for one account are processed and (assuming they don't "bounce") an appropriate Check object is added to the Account object, to wait for end-of-month processing to write the bank statement. The check objects have fields number (of type int), received (of type Date) and amount (of type Money) and must be kept in original order by time of arrival, and are only used once (in our simplified system) to write the statement, where they are reported in the same order.

List<Check>, Set<________________>, Map<________________, ________________>

Both LinkedList and ArrayList are good. Notice that a TreeSet that compares by date is not good because dates may have duplicates and a List is the simplest way to keep items in the order they were inserted.

(b) In the same banking scenario, each account has a unique id (an Integer) and one or more owners identified by social security numbers (also ints). Also bank account holders may have several accounts, each with a unique id. Explain how you can use two Collections API classes working together to support looking up the bank accounts ids for given social security numbers.

List<________________>, Set<________________>, Map<Integer,Set<Integer>>

HashMap and HashSet are best.

(c) You want to be able to display the courses a student took (represented as Strings, like "CS310") and their grades (represented as Characters between 'A' and 'F', assume there are no "A-" grades etc.), **sorted by the grade**. Explain briefly.

List<________________>, Set<________________>, Map<Character, Set<String>>

TreeMap of TreeSet are good. Letter grades can have duplicates, so just a Map from a Character to a String won't do. Another option is to make a List<Set<String>> with 5 entries, from A to F (no E, of course).

3. Hash tables (from Midterm F16):

a. You created a hash table using separate chaining and forgot to rehash. You then realized that you inserted n log n elements into the hash table whose array size is n. What would be the average lookup time for your hash table now? Explain briefly.

Assuming a good hashing function, the average lookup time is the average number of elements per chain, which is $O(\log n)$.

b. What (really bad thing) could happen to the insert/lookup time for a scenario as in (a) above if you used linear probing?

This was one of the trickier questions, maybe because the answer is quite straightforward – you are bound to get stuck in an infinite loop after the n^{th} insert...

c. We implement a hash table of integers, using an array of size 7. The hashing function for an integer x is h(x) = x%7. We use linear probing to resolve collisions. The elements are inserted in the following order: 1,15,14,3,10,5,25. We do not rehash. Draw the final configuration of the table and determine how many collisions each element will cause. For your convenience you may use the following table and illustration of the hash table.

14	1	15	3	10	5	25

Number	Hash value	# collisions
1	1	0
15	1	1
14	0	0
3	3	0
10	3	1
5	5	0
25	4	2

4. Induction. Prove the following formulas. Don't forget to state explicitly the three stages:

   - Base case
   - Inductive hypothesis for $1 < k < n$
   - The inductive step from $n-1 \to n$ (or $n \to n+1$ if it works better)

   (a) For all $n \geq 1$, show that $1 + 4 + 7 + ... + (3n-2) = \frac{n*(3n-1)}{2}$

   - Base case: If $n = 1$, then $\frac{n*(3n-1)}{2} = \frac{1*2}{2} = 1$.
   - Assume it's true for any $1 \leq k < n$.
   - Show for n:
     $\sum_{k=1}^{n}(3k-2) = \sum_{k=1}^{n-1}(3k-2) + 3n - 2 = \frac{(n-1)*(3n-4)}{2} + 3n - 2$ (first part by inductive hypothesis).
     Expand the equations:
     $\frac{(n-1)*(3n-4)}{2} + 3n - 2 = \frac{3n^2-7n+4}{2} + \frac{6n-4}{2} = \frac{3n^2-n}{2} = \frac{n*(3n-1)}{2}$.

   (b) For any positive integer n, $6^n - 1$ is divisible by 5.

   - Base case: If $n = 1$, then $6^1 - 1 = 5$, which is divisible by 5 (obviously).
   - Assume it's true for any $1 \leq k < n$.
   - Show for n:
     $6^n - 1 = 6 * 6^{n-1} - 1 = (5+1) * 6^{n-1} - 1 = 6^{n-1} - 1 + 5 * 6^{n-1}$.
     $6^{n-1} - 1$ is divisible by 5 by inductive hypothesis. $5 * 6^{n-1}$ is divisible by 5 for any $n \geq 1$ (I don't have to explain why, right?). The sum of two numbers divisible by 5 is also divisible by 5.

5. Graphs:

   (a) Remember that the degree of a vertex in an undirected graph is defined as the number of edges touching this graph. Prove that in any simple undirected graph, the number of vertices with odd degrees is even. (a simple graph is a graph where every two vertices are connected by at most one edge).

   Hint: This question has a VERY simple answer, please don't over-think it. The answer follows from the definition of the degree of a vertex and the definition of an edge. 1-2 sentences will suffice. Also, remember that an example does not constitute a proof.

   If you can't possibly get the answer, you can prove by induction on the number of edges in a graph. A correct proof by induction will still get you full marks.

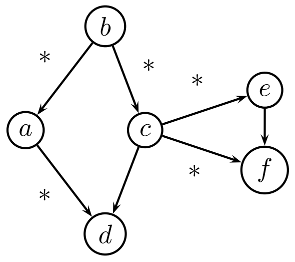
   Short answer: The overall sum of the degrees of the vertices in a simple undirected graph is even, because every edge connects two vertices (no self loops), adding overall 2 to the sum of the degrees. The fact that the number of vertices with odd degrees must be even follows directly from this, otherwise the total sum of the degrees would be odd.

   Longer answer:

   - Base cases: A graph with no edges has all degrees 0. A graph with two vertices connected by an edge: Both vertices have a degree of 1.
   - Assume it's true for any graph with $1 \leq k < n$ edges (notice the number of edges is the property here, much easier to prove than by induction on the vertices IMO).

- (to be as general as possible let's go "top down"): Given a simple graph with n edges. Let's remove an arbitrary edge $e = (u, v)$. We have a now a graph with $n - 1$ edges where, by inductive hypothesis, the number of the vertices with odd degrees is even, say $2m$. Now let's add the edge back. If u and v both had odd degrees before the addition of the edge, both their degrees are now even, so the number of the vertices with odd degrees is $2m - 2$, still even. If u and v both had even degrees before the addition of the edge, both their degrees are now odd, so the number of the vertices with odd degrees is $2m + 2$, still even. If one was odd and one was even, they are now even and odd, respectively, so the number of the vertices with odd degrees is $2m$, still even.

(b) Trace the run of Breadth-First search (BFS) algorithm starting from b in the graph below. For tracing, use the same notation as in the class notes. Do not show the queue, just the order in which the edges are explored. Mark an * near the edges that participate in the final tree.



Order of edges:

b-a

b-c

a-d

c-d (not a tree edge)

c-e

c-f

e-f (not a tree edge)

Distances: b – 0, a – 1, c – 1, d – 2, e – 2, f – 2