CSE 373 18AU: Practice Exam (key)

Name:

UW email address:

Instructions

- **Do not start the exam until told to do so**.
- You have 110 minutes to complete the exam.
- This exam is closed book and closed notes.
- You may not use a calculator, cell phone, or any other electronic devices.
- Write your answers neatly in the provided space.
  Be sure to leave some room in the margins: we will be scanning your answers.
- If you need extra space, use the back of the page.
- If you have a question, raise your hand.
- **Unless specified, when asked for a runtime give the worst-case tight big-$\mathcal{O}$ bound.**

Question	Points
1	4
2	6
Total:	10

Question	Points
3	8
4	8
Total:	16

Question	Points
5	5
6	6
7	8
8	6
Total:	25

Advice

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit.
- The questions are ordered roughly in the increasing order of difficulty.
- Relax. You are here to learn.

Warmup

This part will test some of the basic concepts in the course. This section has multiple choice questions, and short-answer questions.

1. For each of the following questions, choose the best (most correct) option, unless stated otherwise explicitly. *(1 point per question.)*

   (i) How many MSTs can a graph have?

   A. 0
   B. 1
   C. 0 or 1
   D. 0 or 1 or more (depends on the graph)

   (ii) Stack is also called as

   A. Last in first out
   B. First in last out
   C. Last in last out
   D. First in first out

   (iii) To represent hierarchical relationship between elements, which data structure is suitable?

   A. Dequeue
   B. Priority
   C. Tree
   D. Graph

   (iv) Which of the following is a good criteria to resize an open addressing hash table that uses quadratic probing for resolving collisions.

   A. $0 \leq \lambda < 0.25$
   B. $0.25 \leq \lambda < 5$
   C. $0.5 \leq \lambda < 0.75$
   D. $0.75 \leq \lambda < 1$

2. Short answer questions

   Use the provided underlined space on the right to write your answer.

   (i) *(5 points)* Give worst-case tight big-$\mathcal{O}$ bound for the following.

   a. Insertion sort

   a. $\mathcal{O}(n^2)$

   b. Merge sort

   b. $\mathcal{O}(n \log n)$

   c. Heap sort

   c. $\mathcal{O}(n \log n)$

   d. Quick sort

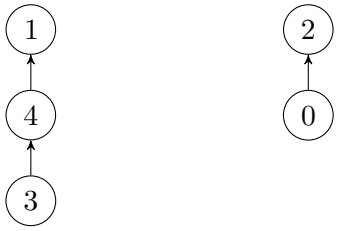
   d. $\mathcal{O}(n^2)$

   e. Selection sort

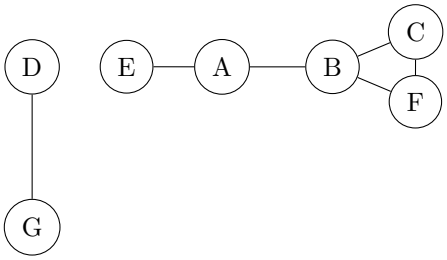
   e. $\mathcal{O}(n^2)$

(ii) *(1 point)* Consider the following disjoint set. Suppose we do `union(1, 2)` using union-by-rank optimization. In the resulting union tree, how many nodes would point directly to node 4?

(ii) **1**



(iii) Consider the *undirected, unweighted* graph below.



a. What is the maximum *degree* of the graph?

a. **3**

b. Are there any cycles? If so, where?

b. **Yes, B-C-F**

c. What is the maximum length simple path in this graph?

c. **4 (E-A-B-F-C)**

d. What is the one edge that you could add to the graph that would increase the length of the maximum length simple path of the new graph to 6?

d. **D-E**

e. What are the *connected components* of the graph?

e. **There are two.** $\{D, G\}$ **and** $\{E, A, B, C, F\}$

Basic

This part will test your ability to apply basic techniques covered in lecture and reinforced through sections and homeworks. Remember to show your work and justify your claims.

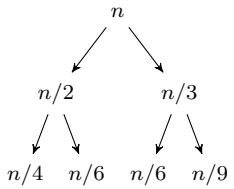
3. For each of the following questions, choose the best option (most correct).
   (i) *(1 point)* Dijkstra's algorithm will always return an incorrect path if the graph contains negative-length edges. Is this statement true or false?
   A. True
   B. False
   (ii) *(2 points)* Suppose you have an directed graph containing $|V|$ nodes. What is the maximum number of edges you can add to this graph while maintaining it acyclic? (Assume that you are not allowed to add parallel edges.)
   A. $|V|$
   B. $|V|(|V|-1)/2$
   C. $|V|(|V|+1)/2$
   D. $|V|^2$
   (iii) *(2 points)* If you insert 16 elements in an empty BST, what are the possible heights of the resulting BST?

   Solution: Anywhere between 4 and 15.

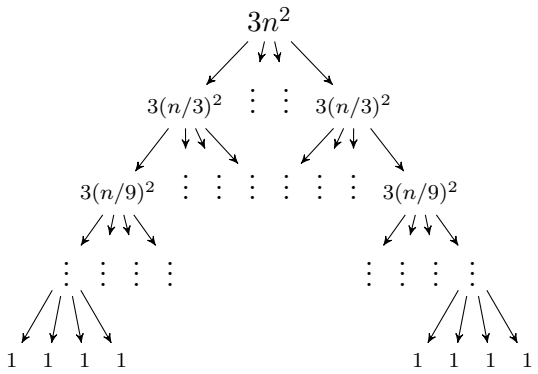
   True or False
   (iv) *(1 point)* We can always sort some list of length n in $\mathcal{O}(n)$ time.
   A. True
   B. False
   (v) *(1 point)* Iterating over a list using the iterator is always faster than iterating by repeatedly calling the `get()` method.
   A. True
   B. False
   (vi) *(1 point)* Dikjkstra's algorithm will always return the incorrect result if the graph contains negative-length edges.
   A. True
   B. False
4. Tree recurrence
   (i) *(4 points)* Draw the first three levels of the recurrence tree for this following recurrence

$$T(x) = \begin{cases} 1 & \text{if } x = 1 \\ T(n/2) + T(n/3) + n & \text{otherwise} \end{cases}$$

   Solution:



(ii) *(4 points)* Write the recurrence for this following recurrence tree



Solution:

$$T(n) = \begin{cases} 1 & \text{if } x = 1 \\ 4T(n/3) + 3n^2 & \text{otherwise} \end{cases}$$

Applied

This section tests your ability to think a little bit more insightfully. The approaches necessary to solve these problems may not be immediately obvious. Remember to show your work and justify your claims.
For each of the answers, unless specified in the question, describe your algorithm in English or pseudocode. DO NOT write java code. For all questions, the expected answers are at most 4-5 sentences. Longer than necessary answers will not get full credit.

5. *(5 points)* Recall that in insertion sort, the algorithm does a linear search to find the position in the sorted subarray where the current element should be inserted. Suppose you use a binary search instead of the linear search to find that position, what would be the worst-case tight big-$\mathcal{O}$ time complexity of the sort? Briefly justify your answer.

   Solution: Even with binary search, the worst-case time complexity of the insertion sort is $\mathcal{O}(n^2)$. Because even if we find the insertion index quickly (in $\mathcal{O}(\log n)$ time), we still need to shift all the elements, which takes $\mathcal{O}(n)$ time, so the time complexity still remains as $\mathcal{O}(n^2)$.

6. *(6 points)* Suppose you are given a source code and you need to figure out the order in which to compile the files. Explain how you would solve this problem? State the runtime of your solution.

   Solution: Represent the source code files as a directed unweighted graph, where each vertex is a file and an edge represents dependency, i.e., if file A imports file B, there is an edge from B to A.
   Run topological sort, which gives the order in which files can compiled.
   Runtime: $\mathcal{O}(|V| + |E|)$

7. Frodo and Sam are on their way to Mordor to destroy the ring, and along their path they have to pass through several human villages on their way, some of which are now deserted and likely Orc territory. They learn that there are some paths that are being heavily guarded by Orcs, and they want to avoid those paths at all costs. Friendly spies tell Frodo and Sam that they should avoid the road between two deserted villages, as it is more likely be monitored by Orcs.
   (i) *(4 points)* How would you represent a graph to capture this problem.
   - What are your vertices and edges?
   - What information would you store for each vertex and each edge?
   - Is the graph directed/undirected, weighted/unweighted?
   - Are there any self-loops? Parallel edges?

   Solution: Undirected unweighted graph, where vertex is a village and edge is a road between villages. Each vertex stores the information whether it is deserted or occupied.
   Graph contains parallel edges, as there can be different roads connects two villages.
   No self-loops, since adding those does not affect the solution.

   (ii) *(4 points)* How should Frodo and Sam find the shortest and safest path to Mordor? State the runtime of your solution.

Solution: While traversing the graph we can identify unsafe edges by checking whether both vertices are deserted. Traverse the graph with BFS, identify all unsafe edges, and then remove them. Run Dijkstra's to find the shortest path.

(NOTE: A slightly more efficient solution would be to ignore the unsafe edges in Dijkstra's.)

Runtime: $\mathcal{O}(|E| \log |V| + |V| \log |V|)$ (same as Dijkstra's).

8. *(6 points)* Suppose you have a bunch of computers networked together (haphazardly) using wires. You want to send a message to every other computer as fast as possible. Unfortunately, some wires are being monitored by some shadowy organization that wants to intercept your messages.

   After doing some reconnaissance, you were able to assign each wire a "risk factor" indicating the likelihood that the wire is being monitored. For example, if a wire has a risk factor of zero, it is extremely unlikely to be monitored; if a wire has a risk factor of 10, it is more likely to be monitored. The smallest possible risk factor is 0; there is no largest possible risk factor.

   Implement an algorithm that selects wires to send your message such that (a) every computer receives the message and (b) you minimize the total risk factor. The total risk factor is defined as the sum of the risks of all of the wires you use.

   Solution: This problem basically boils down to finding the MST of the graph.

   Graph: We make each computer a node and each wire (with the risk factor) a weighted, undirected edge.

   Algorithm: Once we form the graph, we can use either Prim's or Kruskal's algorithm as we implemented them in lecture, with no further modifications.