CSE 100: Algorithm Design and Analysis: Midterm 2

Fall 2022

- This is a take-home exam with no proctoring. There are **8** questions in total covering Lecture slides 8-13. Question 8 offers 5 bonus points. If your final score is higher than 100, it will be considered 100. You can earn some partial points if you show progress even if you can't solve problems completely. You can access this exam from **12:00am on Nov. 7th to 11:59pm on Nov. 8th**. Please submit your solution by **11:59pm, 08-Nov**, through CatCourses (Midterm 2 under Assignments). You can resubmit any number of times until the deadline. If you failed uploading by 11:59pm, email them to your TA, but it will incur 1 pt penalty for you for every minute delay(e.g. If we receive your solutions at 0:03am, you will lose 4pts).

- This is an open-book exam. The **only** resources you can use during the exam are all **course materials** uploaded to CatCourses plus the **textbook**. In particular, you are NOT allowed to search for solutions on the Internet. You must take the exam by yourself. You may be asked to explain your answers to the instructor or TA later. If you cannot provide a clear explanation of your answers, it violates the UC Merced Academic Honesty Policy. This means you'll get 0 grades, and further be reported to the Office of Student Rights and Responsibilities.

- Title your work as CSE 100 Midterm 2, Fall 2022 along with your full name. **Please copy and sign the following honor pledge at the beginning** of your solution sheet(s) right below the title: *By completing this exam, I acknowledge and confirm that I will not give or receive any unauthorized assistance on this examination. I will conduct myself within the guidelines of the university academic integrity.*

- **Write your solution by hand on your solution sheet(s).** Clearly and orderly write down the question numbers. When submitting, scan or take a picture of each sheet and upload them to Catcourse. You are responsible for the clarity and completeness of your uploads to get scores.

- If you have questions, you can email the instructor and the TAs. However, we may not answer your questions on time as we can't be available all the time.

Question	Points earned	Out of
1		10
2		9
3		8
4		40
5		6
6		18
7		9
8		5(Bonus)
Sum		Max 105

1. (10 points) For each of the following claims, determine if it is true or false. Brief explanation is needed.

   (a) (2 points) The running time of (Deterministic) Quicksort is $O(n^2)$.
   Sol. True

   (b) (2 points) If A[1...n] is sorted, we can find the k th smallest element in A[1...n] in O(1) time.
   Sol. True

   (c) (2 points) The decision tree of *any* comparison based sorting algorithm has a height $O(n \log n)$.
   Sol. False

   (d) (2 points) One can sort n integers in the range between 0 and n^{10} in $O(n)$ time.
   Sol. True

   (e) (2 points) The running time of Randomized-Quicksort that picks the pivot uniformly at random is always $O(n \log n)$ for any input of size n.
   Sol. False

2. (9 points)

   (a) (3 points) Explain why the decision tree of any sorting algorithm on n elements must have at least $n!$ leaf nodes.
   Sol. There are n! possible orderings (and every ordering must appear in a leaf node to distinguish among them).

   (b) (6 points) If we sort n elements using randomized quicksort, what is the probability that the maximum element and the minimum element are compared? Explain why.
   Sol. $2/n$. Because it happens if and only if the maximum or the minimum element is chosen as the first pivot.

3. (8 points) For each of the two binary search trees shown in Fig. 1 and 2 (NILs not shown), is it possible to color it as valid red-black tree. If so, provide a red-black tree coloring. If not, write "impossible". In either case, no justification is needed. If you don't have red and black ink, write "r" to color a node red, or "b" to color it black. Each part is worth 4 points.

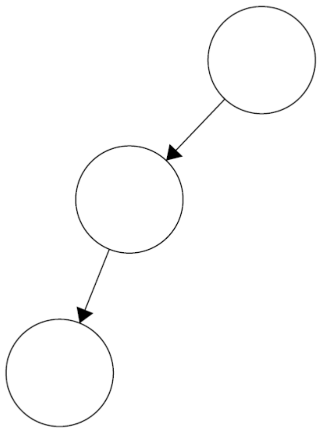


Figure 1: First Tree

Sol. It is impossible to color this BST as a valid red-black tree (You must take NILs into account)

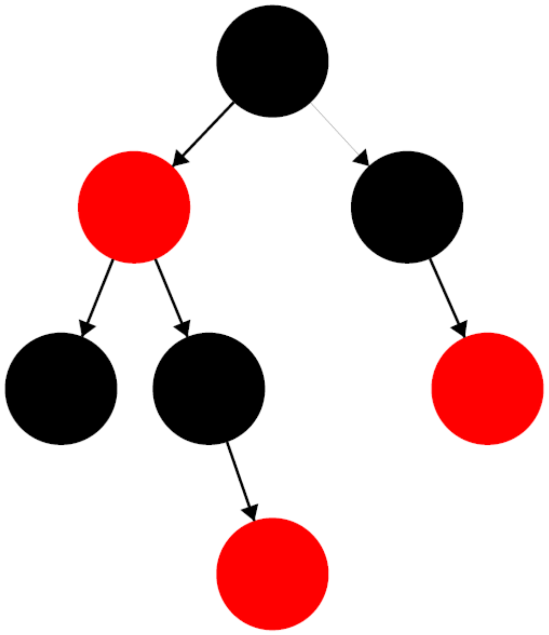
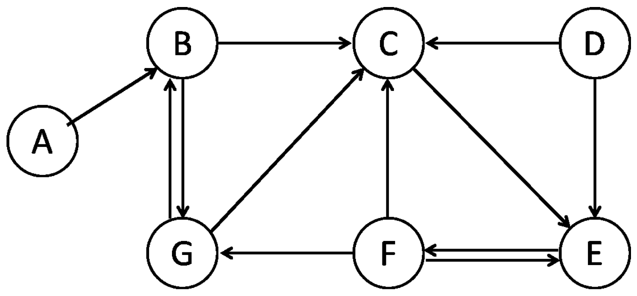


Figure 2: Second Tree

Sol. This is the only valid coloring.

4. (40 points) Graphs. Consider the following directed graph.



(a) (4 points) Draw the adjacency-list representation of G, with each list sorted in increasing alphabetical order.

Sol.
A $\rightarrow$ B
B $\rightarrow$ C $\rightarrow$ G
C $\rightarrow$ E
D $\rightarrow$ C $\rightarrow$ E
E $\rightarrow$ F
F $\rightarrow$ C $\rightarrow$ E $\rightarrow$ G
G $\rightarrow$ B $\rightarrow$ C

(b) (4 points) Give the adjacency matrix of G.

Sol.

	A	B	C	D	E	F	G
A	0	1	0	0	0	0	0
B	0	0	1	0	0	0	1
C	0	0	0	0	1	0	0
D	0	0	1	0	1	0	0
E	0	0	0	0	0	1	0
F	0	0	1	0	1	0	1
G	0	1	1	0	0	0	0

(c) (7 points) Do depth-first search in G, considering vertices in increasing alphabetical order. Show the final result, with vertices labeled with their starting and finishing times.

Sol. Here we assume that we start the DFS from A, but it was not explicitly stated in the question, so students may have decided to start from a different node, and the results might be different with different trees

Node	Start	Finish
A	00	11
B	01	10
C	02	09
E	03	08
F	04	07
G	05	06
D	12	13

(d) (15 points) Based on your results, proceed to run the SCC algorithm find the strongly connected components of G (show the result of the DFS with vertices labeled with their starting and finishing times).

Sol.
SCC1: D (12,13)
SCC2: A (0, 11)
SCC3: B,C,E,F,G (1,10)

(e) (10 points) Run BFS with B as starting vertex. Show the tree edges produced by BFS along with $v.d$ of each vertex v. More precisley, run BFS with B as starting point assuming that each adjacency list is sorted in increasing alphabetical order.

Sol. Here we assume that we start the BFS from B. As this was explicitly stated in the question, this is the only valid solution

Level	Vertices
1	B
2	C,G
3	E
4	F

5. (6 points) Consider a hash table with $m = 10$ slots and using the hash function $h(k) = k$ mod 10. Say we insert (elements of) keys $k = 33, 25, 53, 3, 55, 39$ in this order. Show the final table when chaining is used to resolve collisions. Insert the element at the *beginning* of the linked list.

   Sol.

   slot 3 has a linked list storing 3, 53, 33 in this order.
   slot 5 has a linked list storing 55, 25, in this order.
   slot 9 has a linked list storing 39. All other slots have NIL.

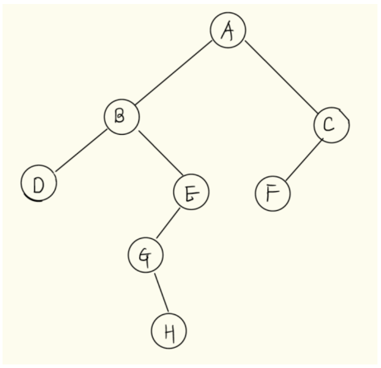
   Any ordering of elements with the same hash value is acceptable. Rubric: if you has an integer to a wrong slot, -1. Wrong order in each LL, -1.

6. (18 points)

   (a) (5 points) You're given an array A[1 ···8] = ⟨2,6,5,4,1,2,4,3⟩. Run Max-heapify on the root. What is A[1 ···8]?
   Sol. Sol. 6, 4, 5, 3, 1, 2, 4, 2.

   (b) (13 points) Consider the following binary search tree where nodes are labeled by alphabet. Here the keys are *not* shown in the picture; $a, b, c, \cdots$ are node labels, not keys. Assume that all keys have distinct values.

      i. (2 points) What is the node with the min key value?
      ii. (2 points) What is the node with the max key value?
      iii. (2 points) What is G's successor?
      iv. (3 points) Output node labels according to in-order-traversal.
      v. (4 points) Delete node B and show the resulting BST.



Sol.

i. D.
ii. C.
iii. H.
iv. D, B, G, H, E, A, F, C.
v. .

7. (9 points) You are given two sequences, $A = \langle a_1, a_2, ..., a_n \rangle$ and $B = \langle b_1, b_2, ..., b_n \rangle$, where each sequence consists of distinct integers. Describe a linear time algorithm (in the average case) that tests if a sequence is a permutation of the other. We say that A is a permutation of B, if we can find every item of A in B. Assume that the simple uniform hashing assumption holds. Explain the running time of your algorithm.

   Sol. We create a hash table of size $\Theta(n)$ and use chaining to resolve collisions. Recall that under the simple uniform hashing assumption, a search, either successful or unsuccessful, take $O(1)$ time on average. We first insert $a_1, a_2, ..., a_n$ into the hash table. This is done in $O(n)$ time in the average case. Then, we search each b_i in the hash table, which is done in $O(1)$ time. If we can find every b_i in the hash table, it, together with the fact that each of the two sequences consists of distinct integers, one is a permutation of the other. The running time is clearly $O(n)$.

8. (5 points) (Bonus) Quicksort implementation. Give a *pseudo-code* of Quicksort(A, p, r) that sorts $A[p...r]$ via quick-sort. You can assume that all elements in A have distinct values. Please add a comment for each line of your code to explain your code. You can use the following helper function, Partition(A, p, r):

```
Partition(A, p, r)
1. x = A[r]
2. i = p-1
3. for j = p to r-1
4.    if A[j] <= x
5.       i = i+1
6.       exchange A[i] with A[j]
7. exchange A[i+1] with A[r]
8. return i+1
```

Sol. See CLRS.