CSCI 3110 Tutorial 0 Solutions

1. We have discussed the Gale-Shapely algorithm for stable matching in class at a very high level. This algorithm takes $\mathrm{O}(n^2)$ time if implemented properly—$\mathrm{O}(1)$ time for each of the $\mathrm{O}(n^2)$ iterations. The running time will be *much* worse if the algorithm is not implemented properly.

   (a) (20 pts) Discuss how to implement the algorithm using elementary data structures such as arrays, linked lists, stacks, and queues to represent preference lists, the list of unmarried men, etc. The resulting algorithm should take $\mathrm{O}(n^2)$ time.

   ANSWER: We need to choose data structures to implement the men's preference lists, the women's preference lists and the list of unmarried men. We can use a stack for the list of unmarried men, which we initially fill with every man. This allows us to implement step 1 of the algorithm, picking an unmarried man m, if one exists. We can use an array for the men's preference lists, storing the current position of each man in their preference list. This allows us to choose m's favourite woman that he has not proposed to yet, w. To determine if w is married or not, we store the current marriages in an array $M[]$ where $M[i]$ is the man that woman w_i is married to, or 0 if she is not yet married. This array gives us w's current husband m', if she is married. If w prefers m over m' or is not married then we simply update $M[w]$ with m'.

   The only difficulty is in determining whether w prefers m or m'. To implement this step, we can use an $\mathrm{O}(n^2)$ time preprocessing step that converts the woman's preference lists into *inverse preference lists*. These are arrays that store what position a man m_j is in a woman w_i's list at position j rather than having each man in order. Each man has a unique index, so these inverse preference lists can be built with a simple linear scan.

   For example, consider w_1 from the example in the first lecture slide. Her preference list is m_2, m_3, m_1, m_5, m_4. Scanning through this preference list she builds her inverse preference list:

0	0	0	0	0	initial state
0	1	0	0	0	m_2 is first
0	1	2	0	0	m_3 is second
3	1	2	0	0	m_1 is third
3	1	2	0	4	m_5 is fourth
3	1	2	5	4	m_4 is fifth

   Using an inverse preference list we can easily check whether w prefers m over m'.

(b) (15 pts) Prove that the running time of the algorithm is indeed $\mathrm{O}(n^2)$. You may use the fact that there are $\mathrm{O}(n^2)$ iterations as proved in class, but do not forgot to consider the time required to initialize data structures.

ANSWER: There are n men and n women, so we use $\mathrm{O}(n^2)$ space for the chosen data structures and $\mathrm{O}(n^2)$ time to initialize them. Each inverse preference list can be created with an $\mathrm{O}(n)$ scan for a total of $\mathrm{O}(n^2)$ time to construct the n inverse preference lists. Thus, initialization and preprocessing take $\mathrm{O}(n^2)$ time.

We proved in class that there are $\mathrm{O}(n^2)$ iterations so we simply need to show that each iteration can be carried out in constant time. This follows directly from the description of the chosen data structures in part (a), because each step of the algorithm requires a constant number of elementary operations using our chosen data structures.

2. (35 pts) Given a set S of n integers, we are interested in supporting the following operations:

$\textsc{Find}(S, x)$: Decide whether $x \in S$.

$\textsc{Range-Find}(S, a, b)$: Report all elements $x \in S$ with $a \leq x \leq b$.

$\textsc{Predecessor}(S, x)$: Given an element $x \in S$, report the next smaller element in S.

$\textsc{Insert}(S, x)$: Add the element x to S.

$\textsc{Delete}(S, x)$: Delete the element x from S. Assume you have a pointer to x (e.g. you already ran $\textsc{Find}(S, x)$).

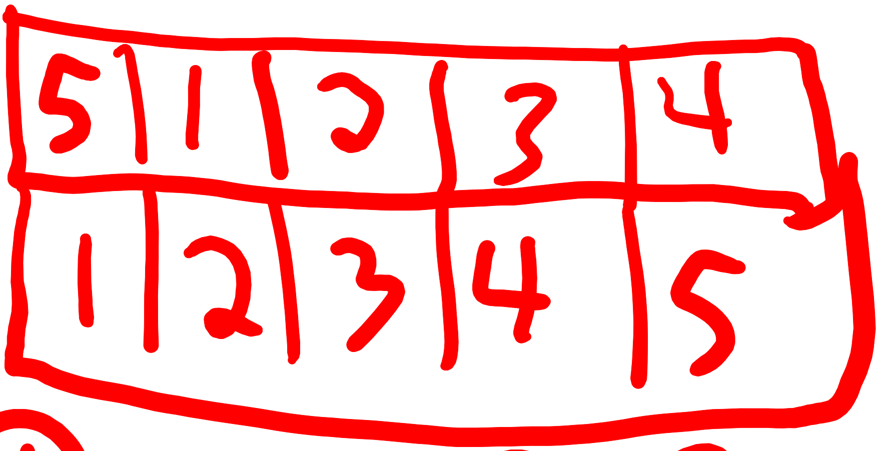
Make a table listing the costs of these operations on each of the following data structures that can be used to represent S:

- An unsorted array of elements,
- a sorted array of elements,
- a linked list,
- a doubly-linked list,
- a hash table, and
- a red-black tree.

You do not need to argue why these operations have the costs you state, but I assume that you would be able to if asked. This knowledge is an essential foundation we will build on throughout the course. You should have seen all of these data structures before (except, perhaps the red-black tree); the purpose of this question is to review what you know about these data structures. If you are not familiar with red-black trees then you may find the information on them through any means other than asking someone to answer the question for you (i.e. the internet, library, etc.) . This is the one exception to the usual rule of not using outside sources.

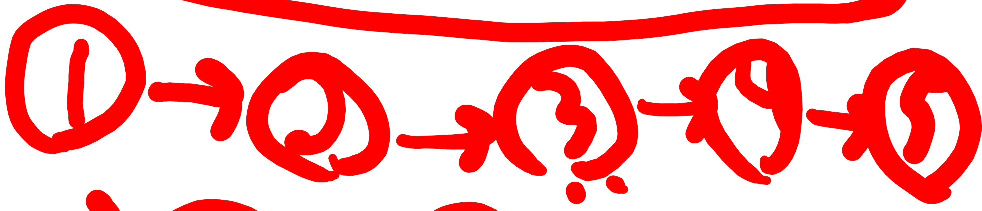
For linked and doubly-linked lists, choose whether or not to keep it sorted. Make the choice that gives you the best overall performance of the data structure. (If your choice makes one or more operations slower than if you had chosen the other option, but none of the other operations gets faster, you have made the wrong choice.)

unsorted A

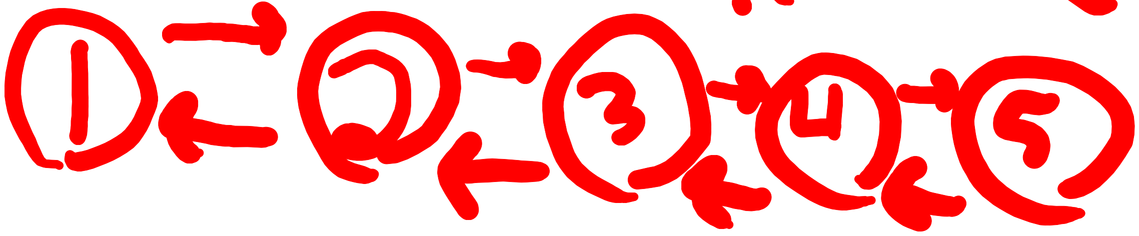


sorted A

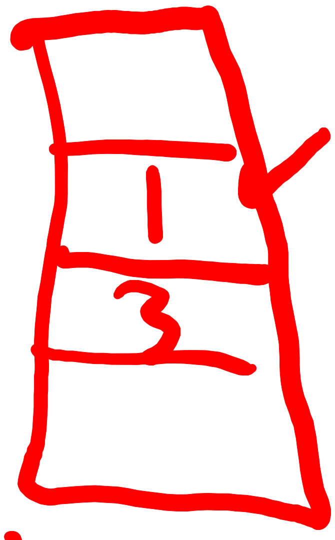
linked list



DLL



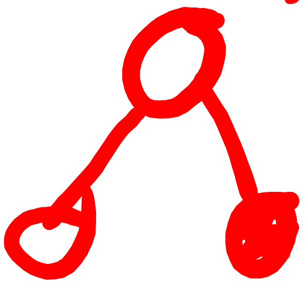
hashtables



$1 \rightarrow f(1)$

$f(1) = f(3)$

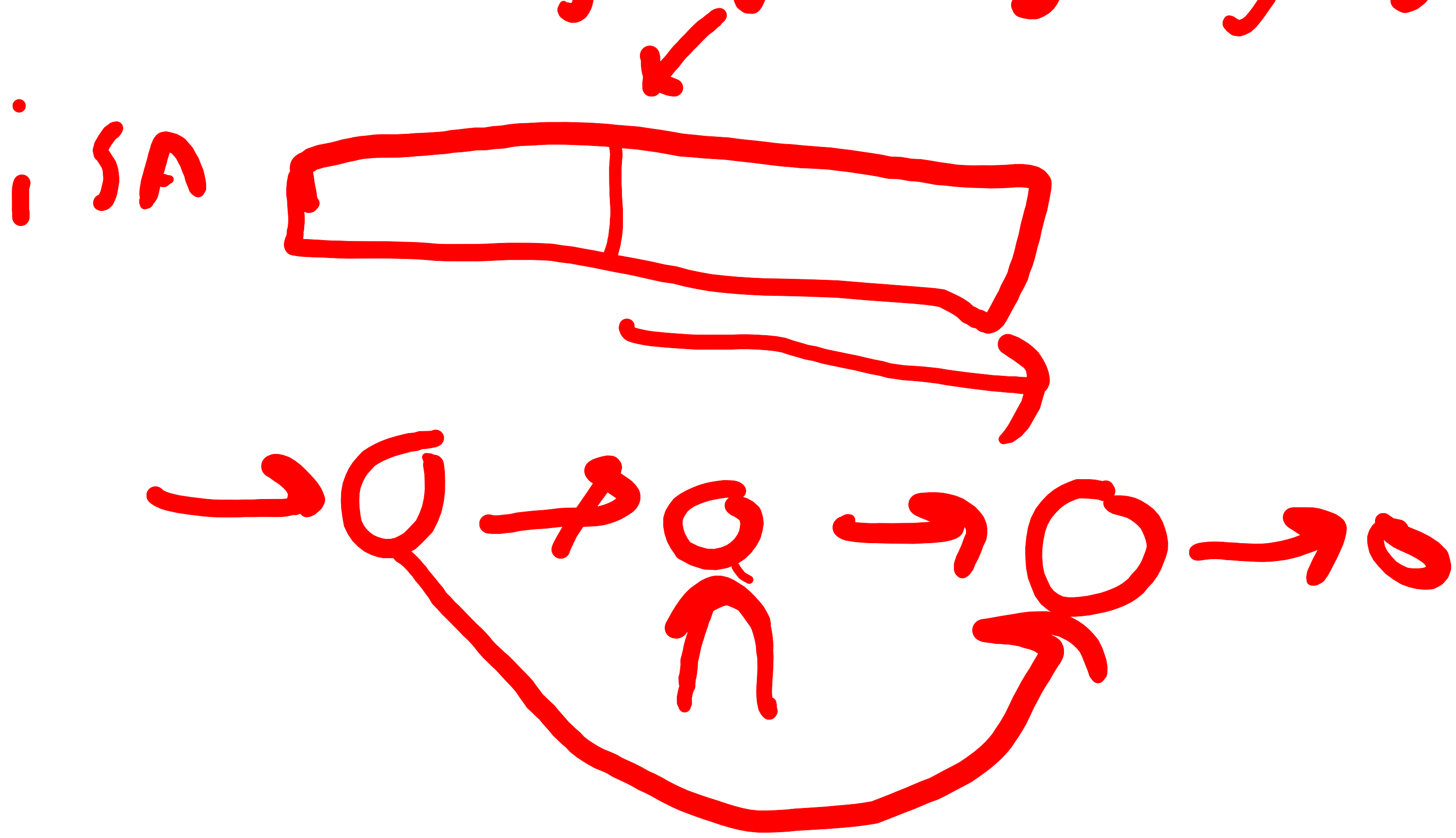
red black tree

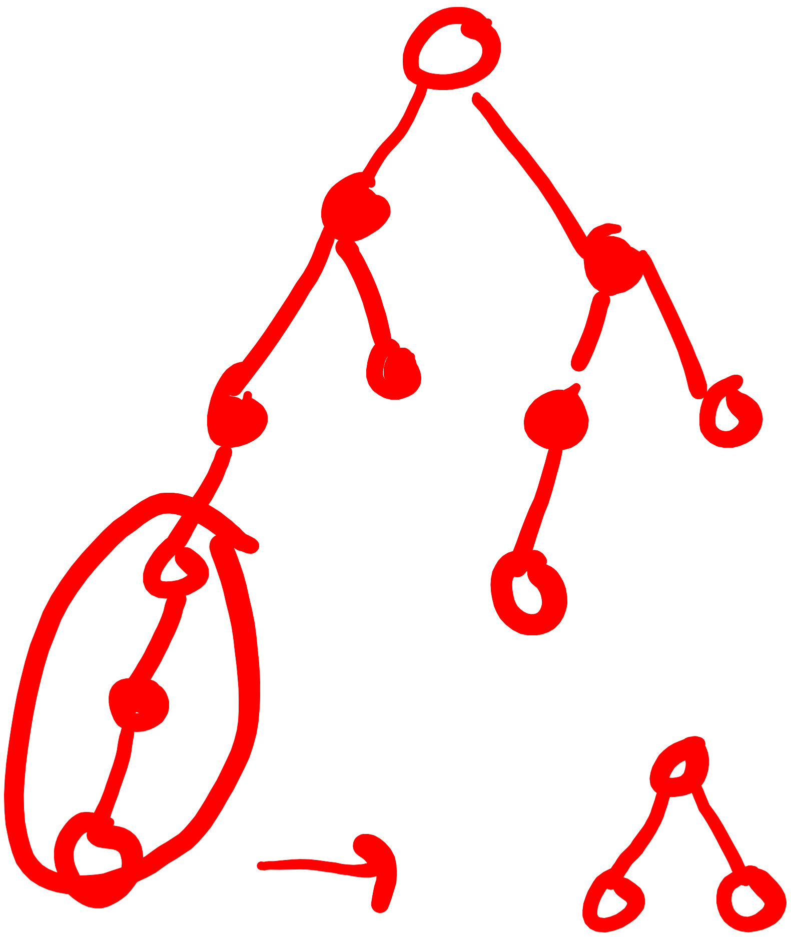


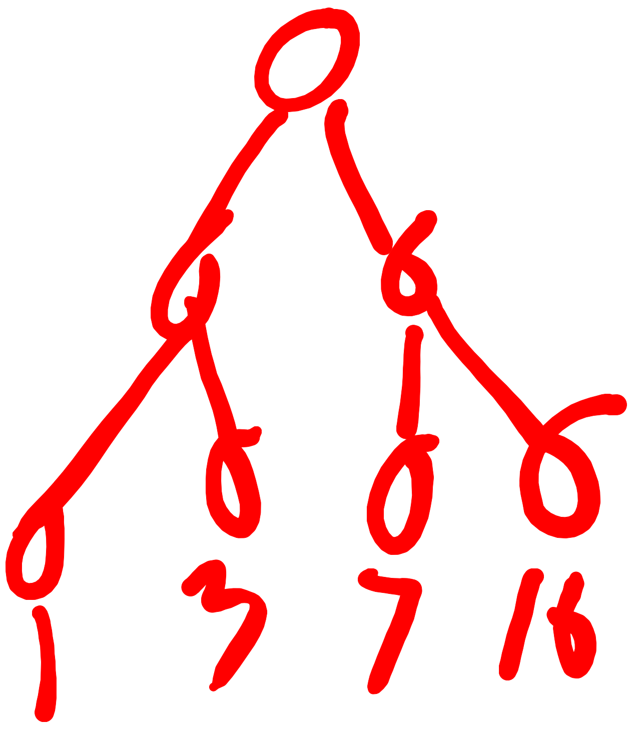
For hash tables, state the expected and the worst-case cost of each operation and indicate clearly which is which.

ANSWER:

	FIND	RANGE-FIND	PREDECESSOR	INSERT	DELETE
Unsorted array	n	n	n	1	n
Sorted array	lg n	lg n + k	lg n	n	n
Unsorted Linked list	n	n	n	1	n
Unsorted D-linked list	n	n	n	1	1
Hash table	n	n	n	1	n
Hash table (expected)	1	n	n	1	1
Red-black tree	lg n	lg n + k	lg n	lg n	lg n






1
3
7
16