CMSC 420: Fall 2022

CMSC 420 (0201) - Final Exam

This exam is closed-book and closed-notes. You may use three sheets of notes (front and back). Write all answers on the exam paper. You may use any algorithms or results given in class. If you have a question, either raise your hand or come to the front of class. Total point value is 120 points. Good luck!

The boxes here are for Gradescope. Put your primary answer in each box. We added an extra blank page to the end of the exam. If you have supporting comments, scratch work, or other, add it there. If the box for a question is small it means the answer is short, but not the reverse.

Your Name:	9-Digit StudentID:

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your Signature:

Problem 1. (50 points) Short answer questions. Unless requested, explanations are not required, but may be given to help with partial credit.

(a) (2 points) You have an inorder-threaded binary tree with n nodes. Let u be an arbitrary non-leaf node in this tree. **True or False**: There must be at least one thread that points into u.

Circle one: TRUE FALSE

(b) (4 points) Let T be an extended binary search tree (that is, one having internal and external nodes). You visit the nodes of T according to one of the standard traversals (preorder, postorder, or inorder). Which of the following statements is necessarily true? (Select all that apply.)

(1) *Preorder traversal*: All the internal nodes appear *before* any of the external nodes

(2) *Inorder traversal*: Internal and external nodes *alternate* with each other

(3) *Postorder traversal*: The *first* node visited is an *external node*

(4) *Postorder traversal*: The *last* node visited is an *internal node*

Circle all that apply: (1) (2) (3) (4)

(c) (4 points) When we delete an entry from a simple (unbalanced) binary search tree, we sometimes need to find a replacement key. Suppose that `p` is the node containing the deleted key. Which of the following statements are true? (Select all that apply.)

(1) A replacement is needed whenever `p` is the root

(2) A replacement is needed whenever `p` is a leaf

(3) A replacement is needed whenever `p` has two non-null children

(4) At most one replacement is needed for each deletion operation

Circle all that apply: (1) (2) (3) (4)

(d) (4 points) You build a union-find data structure for a set of n objects. Initially, each element is in a set by itself. You then perform k union operations, where $k < n$. Each operation merges two different sets. Can the number of union-find trees be determined from k and n alone? If not, answer "It depends". If so, give the number of trees as a function of k and n.

Answer: _____

(e) (4 points) Given a binary max-heap with n entries ($n \geq 3$), you want to return the *third largest* item in the heap (without modifying its contents). What is the minimum number of heap entries that you might need to inspect to be certain that the third largest item is among them?

Answer: _____

(f) (8 points) What are the min and max number of nodes in a 2-3 tree of height 2? (Remember, *height* is the number of edges from the root to the deepest leaf.)

Min: _____ Max: _____

(g) (4 points) You have just performed a deletion from a 2-3 tree of height h. As a function of h, what is the maximum number of key-rotations (adoptions) that might be needed as a result?

Answer: _____

(h) (4 points) You have just inserted a key into an AA tree having L levels. As a function of L, what is the maximum number of skew operations that might be needed as a result? (Here we are only counting skew operations that have an effect on the structure, in the sense that a rotation is performed.)

Answer: _____

(i) (4 points) You have just inserted n (distinct) keys into a treap. As a function of n, what is the probability that the smallest of the n keys is located at the root of the tree?

(1) 0 (That is, it cannot happen)
(2) Roughly $1/n$ (By "roughly", we mean "up to constant factors")
(3) Roughly $1/(\log n)$
(4) Roughly $1/2^n$
(5) Roughly $1/(n!)$

Circle one: (1) (2) (3) (4) (5)

(j) (2 points) The AA-tree data structure has the following constraint: "*The left child of any node must be black.*" Which of the following operations is invoked when this condition is violated?

Circle one: `skew` `split`

(k) (4 points) You have a skip list containing n keys, where n is a very large number. Suppose you perform a `find` operation. The search algorithm visits one or more nodes at each level of the structure. How many nodes do you expect to visit at level 4 of the search structure? (Select one.)

(1) $O(1)$
(2) $O(\log n)$
(3) $O(n/(2^4))$
(4) All of them
(5) None of the above

Circle one: (1) (2) (3) (4) (5)

(l) (4 points) You have a skip list with n nodes. Suppose that rather than using a fair coin to decide a node's height, you instead use a coin that comes up heads with probability 3/4 and tails with probability 1/4. All nodes start at level 0, and a node survives to the next higher level if the coin toss comes up heads. As a function of n, what is the expected number of nodes that survive to level 2?

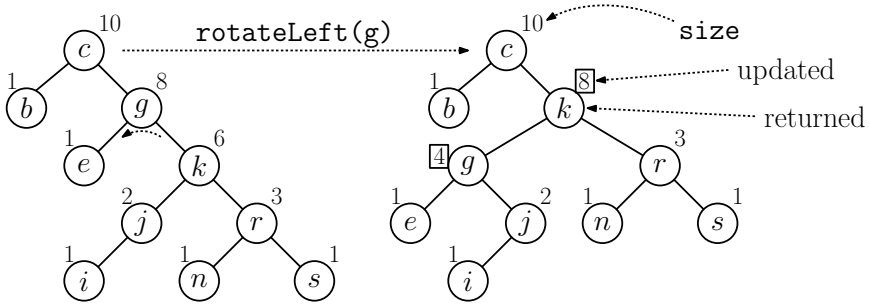
Answer: _____

(m) (2 points) A node in a B-tree has too many children. Suppose that it is possible to resolve this either by *splitting* or *key-rotation* (adoption). Which is preferred? (No explanation needed.)

Circle one: splitting key-rotation

Problem 2. (15 points) In this problem, we will assume that we have a standard (unbalanced) binary search tree. Each node stores a `key`, `left` and `right` child links, and one additional field, `size`, which indicates the number of nodes in the subtree rooted at the current node.

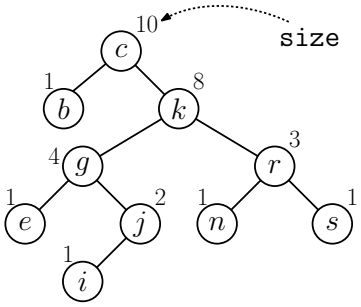
(a) (5 points) Present pseudo-code for an operation `Node rotateLeft(Node p)`, which performs a left rotation at the given node `p`, and also updates the `size` values for all nodes whose sizes are affected by the rotation. It returns the node that takes the place of `p` after the rotation. You may assume that `p.right` is not `null` and all the size values are valid prior to the rotation. For full credit this should run in $O(1)$ time.



(Continued on the next page)

Problem 2 (cont.)

(b) (10 points) Present pseudo-code for an operation `Key getKth(int k)`. Given an integer $k \geq 1$, it returns the kth smallest key in the tree. If the tree has fewer than k keys, this should return `null`. (Hint: Write a helper function `Key getKth(int k, Node p)`, which returns the value of the kth smallest key in the subtree rooted at `p`.) Briefly explain how your function works. For full credit, this should run in time proportional to the height of the tree, independent of the value of k (which may be very large).



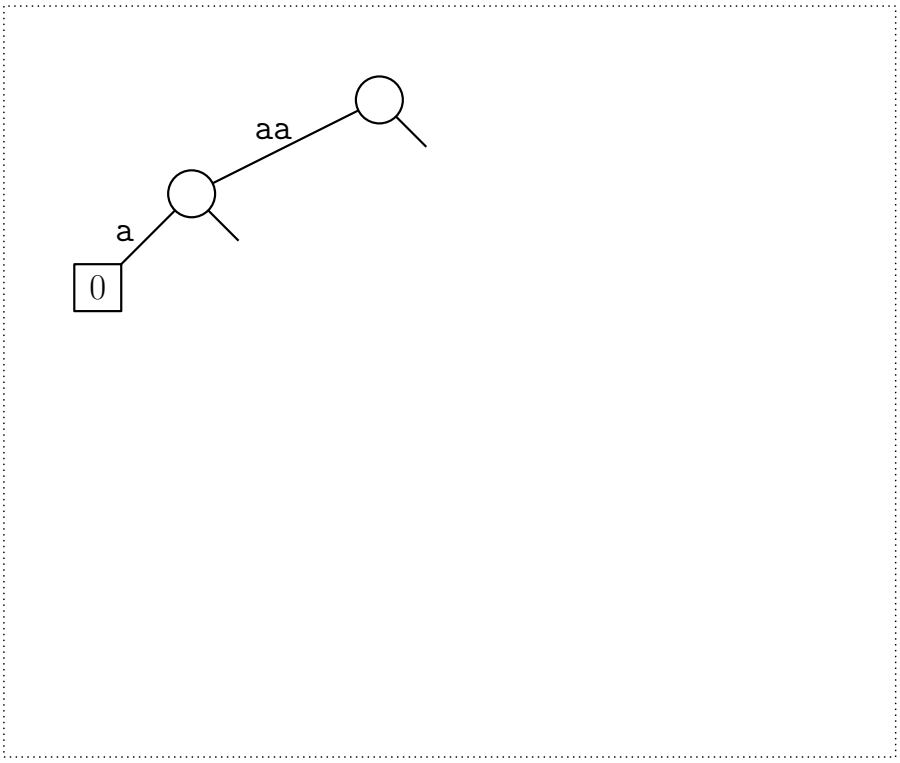

```
getKth(1) = "b"
getKth(3) = "e"
getKth(8) = "n"
getKth(0) = null
getKth(11) = null
```

Problem 3. (10 points) Consider the following set of strings over the alphabet {a, b}.

$$\begin{aligned} S_0 &= \texttt{aaa} \\ S_1 &= \texttt{aabaabaab} \\ S_2 &= \texttt{aabaabb} \\ S_3 &= \texttt{aabab} \\ S_4 &= \texttt{bab} \\ S_5 &= \texttt{bba} \end{aligned}$$

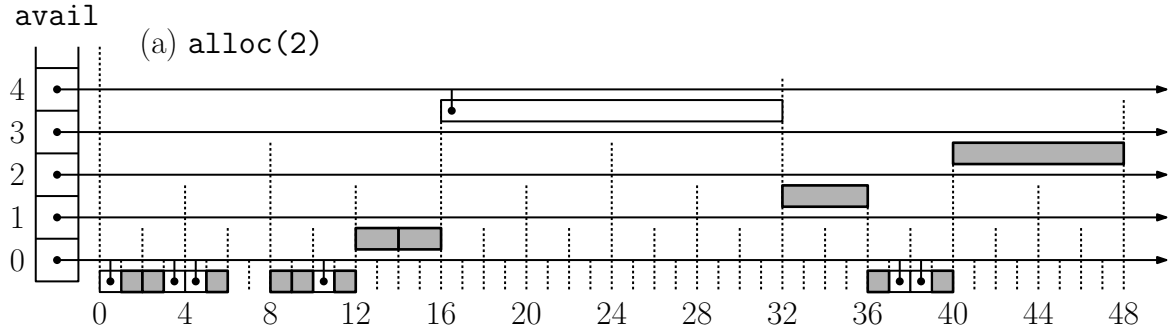
Draw the *Patricia trie* for $\{S_0, \ldots, S_5\}$. (Remember that such a trie uses path compression whenever possible.) Draw your tree using the convention given in class, where each edge is labeled with the substring it matches. For consistency, order the children of each node in alphabetical order from left to right. Label the external nodes with integers 0 to 5 according to which string this is. (We've started you off by drawing the path for S_0 in the figure. Fill in the rest.)

i	S_i
0	aaa
1	aabaabaab
2	aabaabb
3	aabab
4	bab
5	bba

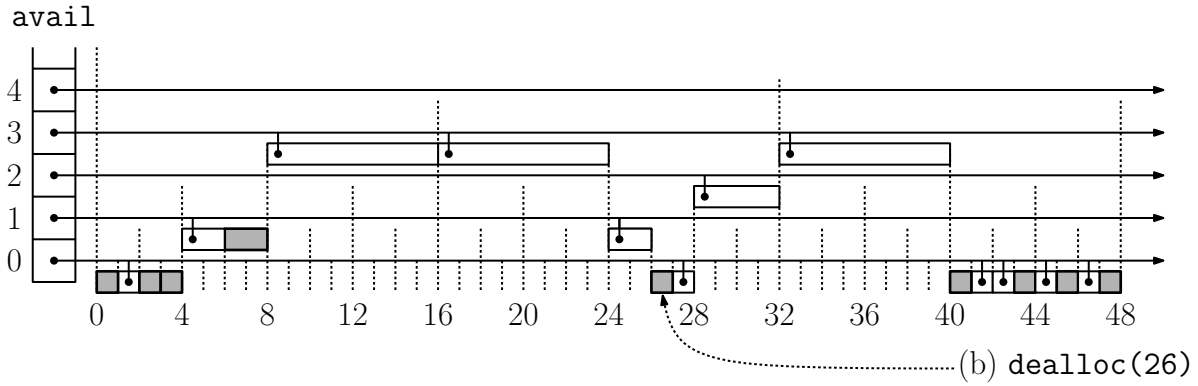


Problem 4. (10 points, 5 points each) This problem involves performing operations using the buddy system for memory allocation.

(a) Consider the buddy allocation shown in the figure below. Explain which blocks are split in order to perform the operation `alloc(2)`. Show the final blocks and indicate what level of the structure they reside. Assume that we always split the leftmost block of sufficient size. You may assume that the size of the final block is exactly 2, there is no need to round the size up for the sake of adding header information. (You can draw your changes directly on top of the figure.)

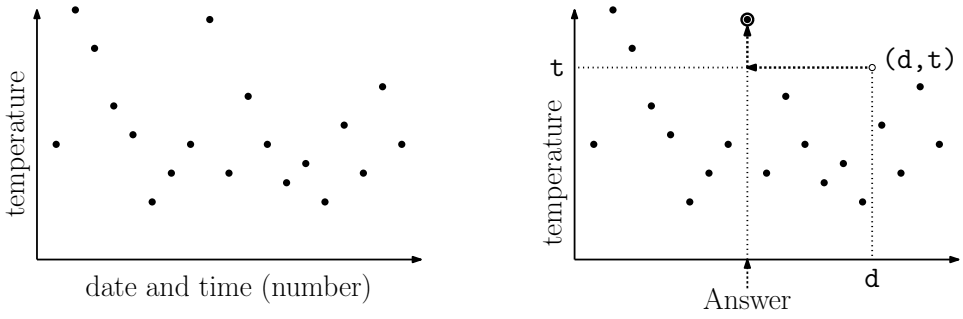


(b) Consider the buddy allocation shown in the figure below. Explain which blocks are merged in order to perform the operation `dealloc(26)`, which deallocates the shaded block of size 1 at address 26 as shown in the figure. Show the final merged block and indicate which level it resides at. (You can draw your changes directly on top of the figure.)



Problem 5. (15 points) The local weather service measures and stores temperatures throughout the day over many decades. (This is a lot of data!) Many people are interested in climate change, and they are often asked *temperature queries* of the form: "When was the last time prior to 11:00am, June 16, 2022 when the temperature exceeded 105°?"

This can be modeled as a 2-dimensional retrieval problem. You are given a set of n points (x, y), where x represents the date and time of the measurement (encoded as a real number) and y denotes the temperature at that date and time (see the figure below).



Let us assume that these points are stored in a standard 2-dimensional kd-tree.

(a) (10 points) Present pseudocode for an efficient function that answers *temperature queries*. Given a pair (`d`, `t`), where `d` is a date/time (encoded as a real) and `t` is a temperature, the query returns the date/time on or before d such that the temperature at that time was greater than or equal to t (see the figure, right).

Assume a standard kd-tree (cutting dimensions alternate and the tree is well-balanced). For full credit, your algorithm should not visit any nodes that obviously cannot contribute to the final answer. You may assume any geometric primitive operations you like.

Hint: You can use any helper, but here is a suggestion:

```
double tempQuery(double d, double t, KDNode p, Rect cell, double best)
```

where `d` is the query date/time, `t` is the query temperature, `p` is the current node of the kd-tree, `cell` is the rectangular cell associated with `p`, and `best` is the best answer so far.

Checklist:

- Handle the case when `p == null`?
- Check whether the cell is relevant?
- Process `p.point`?
- Order of recursive calls?
- Initial call to your helper?

Give your answer on the next page

Answer to Problem 5(a)

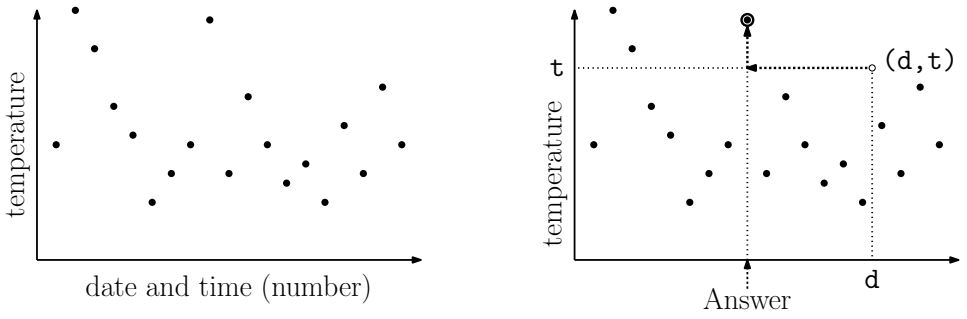
(Suggested helper)

```
double tempQuery(double d, double t, KDNode p, Rect cell, double best)
```

(b) (5 points) What is the running time of your algorithm? (No explanation needed.)

Answer: _____

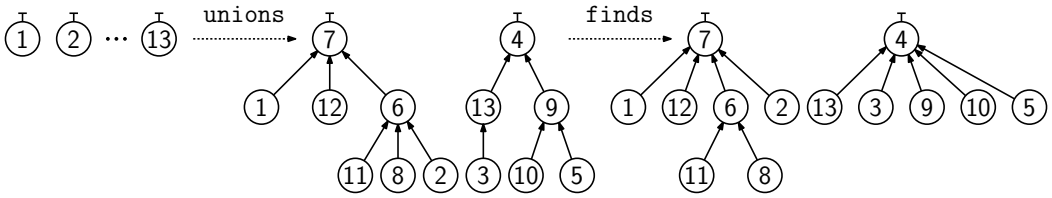
Problem 6. (10 points, 5 points each) This is a repeat of Problem 5 (Temperature Queries), but this time, explain how to solve it using *range trees*.



(a) Briefly describe your data structure and derive its space bound. (What layers are used? How is each sorted? What auxiliary data, if any, is stored in each node?)

(b) Briefly explain how queries are answered and derive the query time.

Problem 7. (10 points) Suppose that we are given a set of n items (initially each item in its own set), and we perform a sequence of m `unions` and `finds` (using height-balanced `unions` and path-compression `finds` as given in class). *Further, suppose that all the* ***`unions`*** *occur before any of the* ***`finds`****.* Prove that after initialization, the resulting sequence takes $O(m)$ time to execute (rather than the $O(m \cdot \alpha(m, n))$ time given in class).



Hint: Classify the links of the Union-Find tree as being of two types: (1) those that point directly to a root node and (2) those that point to a non-root node. Start by proving that for any $k \geq 1$, if a `find` traverses k links, then $k - 1$ links in the tree switch from type-(2) to type-(1).

Scratch Work Area:

Scratch Work Area: