

## Quiz 1

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on the top of every page of this quiz booklet.
- You have 120 minutes to earn a maximum of 120 points. Do not spend too much time on any one problem. Skim them all first, and attack them in the order that allows you to make the most progress.
- **You are allowed one double-sided letter-sized sheet with your own notes.** No calculators, cell phones, or other programmable or communication devices are permitted.
- Write your solutions in the space provided. Pages will be scanned and separated for grading. If you need more space, write “Continued on S1” (or S2, S3) and continue your solution on the referenced scratch page at the end of the exam.
- Do not spend time and paper rederiving facts that we have presented in lecture or recitation. Simply cite them.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required. Be sure to argue that your **algorithm is correct**, and analyze the **asymptotic running time of your algorithm**. Even if your algorithm does not meet a requested bound, you **may** receive partial credit for inefficient solutions that are correct.
- **Pay close attention to the instructions for each problem.** Depending on the problem, partial credit may be awarded for incomplete answers.

| Problem               | Parts | Points |
|-----------------------|-------|--------|
| 1: Information        | 2     | 2      |
| 2: Frequentest        | 2     | 8      |
| 3: Haphazard Heaps    | 2     | 10     |
| 4: Transforming Trees | 2     | 10     |
| 5: Sorting Sock       | 4     | 20     |
| 6: Triple Sum         | 1     | 15     |
| 7: Where am $i$ ?     | 1     | 15     |
| 8: Methane Menace     | 1     | 20     |
| 9: Vapor Invite       | 1     | 20     |
| Total                 |       | 120    |

Name: \_\_\_\_\_

MIT Kerberos Username: \_\_\_\_\_

**Problem 1.** [2 points] **Information** (2 parts)

(a) [1 point] Write your name and email address on the cover page.

(b) [1 point] Write your name at the top of each page.

**Problem 2.** [8 points] **Frequentest** (2 parts)

The following two Python functions correctly solve the problem: given an array  $X$  of  $n$  positive integers, where the maximum integer in  $X$  is  $k$ , return the integer that appears the most times in  $X$ . Assume: a Python `list` is implemented using a dynamic array; a Python `dict` is implemented using a hash table which randomly chooses hash functions from a universal hash family; and `max(X)` returns the maximum integer in array  $X$  in worst-case  $O(|X|)$  time. For each function, state its **worst-case** and **expected** running times **in terms of  $n$  and  $k$** .

**(a)** [4 points]

```
def frequentest_a(X):
    k = max(X)
    H = {}
    for x in X:
        H[x] = 0
    best = X[0]
    for x in X:
        H[x] += 1
        if H[x] > H[best]:
            best = x
    return best
```

**i.** Worst-case:
**ii.** Expected:
**(b)** [4 points]

```
def frequentest_b(X):
    k = max(X)
    A = []
    for i in range(k + 1):
        A.append(0)
    best = X[0]
    for x in X:
        A[x] += 1
        if A[x] > A[best]:
            best = x
    return best
```

**i.** Worst-case:
**ii.** Expected:

**Problem 3.** [10 points] **Haphazard Heap** (3 parts)

Array  $[A, B, C, D, E, F, G, H, I, J]$  represents a **binary min-heap** containing 10 items, where the key of each item is a **distinct** integer. State which item(s) in the array could have the key with:

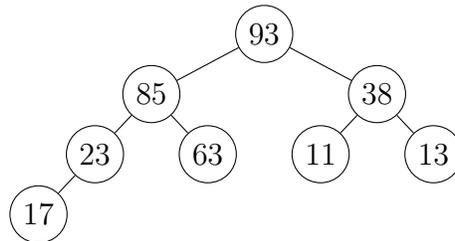
(a) the smallest integer

(b) the third smallest integer

(c) the largest integer

**Problem 4.** [10 points] **Transforming Trees** (2 parts)

The tree below contains 8 items, where each stored item is an integer which is its own key.



- (a) [6 points] Suppose the tree drawn above is the implicit tree of a binary max-heap  $H$ . State the array representation of  $H$ , **first before** and **then after** performing the operation  $H.delete\_max()$ .

- (b) [4 points] Suppose instead that the original tree drawn above is a Sequence AVL Tree  $S$  (note Sequence data structures are zero-indexed). The items in **the leaves** of  $S$  in traversal order are  $(17, 63, 11, 13)$ . Perform operation  $S.delete\_at(3)$  on  $S$  including any rotations, and then **list the items** stored in **the leaves** of  $S$  in traversal order, after the operation has completed. (You do not need to draw the tree.)

**Problem 5.** [20 points] **Sorting Sock** (4 parts)

At Wog Hearts School of Wizardry and Witchery,  $n$  incoming students are sorted into four houses by an ancient magical artifact called the Sorting Sock. The Sorting Sock first sorts the  $n$  students by each of the four houses' attributes and then uses the results to make its determinations. For each of the following parts, state and justify what type of sort would be most efficient. (By "efficient", we mean that faster correct algorithms will receive more points than slower ones.)

- (a) [5 points] For House Puffe Huff, students must be sorted by *friend number*, i.e., how many of the other  $n - 1$  incoming students they are friends with, which can be determined in  $O(1)$  time.

- (b) [5 points] For House Craven Law, students must be sorted by the weight of their books. Book weights cannot be measured precisely, but the Sorting Sock has a *scale* that can determine in  $O(1)$  time whether one set of books has total weight greater than, less than, or equal to another set of books.

(c) [5 points] For House Driven Gore, students must be sorted by *bravery*, which can't be directly measured or quantified, but for any set of students, the Sorting Sock can determine the bravest among them in  $O(1)$  time, e.g., by presenting the students with a scary situation.

(d) [5 points] For House Leather Skin, students must be sorted by their *magical lineage*: how many of a student's ancestors within the previous  $3\lceil \log n \rceil + 4$  generations were magical. Recall that humans, magical or not, always have two parents in the previous generation, unlike binary tree nodes which have at most one. Assume the Sorting Sock can compute the magical lineage of a student in  $O(1)$  time.

**Problem 6.** [15 points] **Triple Sum**

Given three arrays  $A, B, C$ , each containing  $n$  integers, give an  $O(n^2)$ -time algorithm to find whether some  $a \in A$ , some  $b \in B$ , and some  $c \in C$  have zero sum, i.e.,  $a + b + c = 0$ . State whether your running time is worst-case, expected, and/or amortized.

**Problem 7.** [15 points] **Where Am  $i$ ?**

Given a Sequence AVL Tree  $T$  containing  $n$  nodes, and a pointer to a node  $v$  from  $T$ , describe an  $O(\log n)$ -time algorithm to return the (zero-indexed) index  $i$  of node  $v$  in the traversal order of  $T$ . (Recall that every node  $u$  in a Sequence AVL Tree  $T$  stores an item  $u.item$ , parent  $u.parent$ , left child  $u.left$ , right child  $u.right$ , subtree height  $u.height$ , and subtree size  $u.size$ .)

**Problem 8.** [20 points] **Methane Menace**

FearBird is a supervillian who has been making small holes in the methane gas pipe network of **mahtoG City**. The network consists of  $n$  pipes, each labeled with a distinct positive integer. A hole  $i$  is designated by a pair of positive integers  $(p_i, d_i)$ , where  $p_i$  denotes the label of the pipe containing the hole, and  $d_i$  is a positive integer representing the *distance* of the hole from the *front* of pipe  $p_i$ . Assume any two holes in the same pipe  $p_i$  will be at different distances from the front of  $p_i$ . When a new hole  $(p_i, d_i)$  is spotted, the city receives a *report* of the hole to keep track of. The city will periodically patch holes using the following priority scheme:

- if each pipe contains at most one hole, patch any hole (if one exists);
- otherwise, among pairs of holes  $(p_i, d_i)$  and  $(p_j, d_j)$  appearing on the **same pipe**, i.e.,  $p_i = p_j$ , identify any pair with smallest distance  $|d_i - d_j|$  between them, and patch one of them.

Describe a database supporting the following operations, where  $k$  is the number of recorded but unpatched holes in the network at the time of the operation. State whether your running times are worst-case, expected, and/or amortized.

|  |   |
|--|---|
| <code>initialize(<math>H</math>)</code>    | Initialize the database with $n$ holes $H = \{(p_0, d_0), \dots, (p_{n-1}, d_{n-1})\}$ , with one hole on each pipe, in $O(n)$ time |
| <code>report(<math>p_i, d_i</math>)</code> | Record existence of a hole in pipe $p_i$ at distance $d_i$ in $O(\log k)$ time  |
| <code>patch()</code>                       | Patch any hole that follows the priority scheme above in $O(\log k)$ time   |

**Problem 9.** [20 points] **Vapor Invite**

Vapor is an online gaming platform with  $n$  users. Each user has a unique positive integer **ID**  $d_i$  and an updatable **status**, which can be either active or inactive. Every day, Vapor will post online an **active range**: a pair of positive integers  $(a, b)$  with the property that **every user** having an ID  $d_i$  contained in the range (i.e., with  $a \leq d_i \leq b$ ) **must be active**. Vapor wants to post an active range containing as many active users as possible, and invite them to play in a special tournament. Describe a database supporting the following **worst-case** operations:

|  |  |
|--|--|
| <code>build(<math>D</math>)</code>           | Initialize the database with user IDs $D = \{d_0, \dots, d_{n-1}\}$ , setting all user statuses initially to active, in $O(n \log n)$ time |
| <code>toggle_status(<math>d_i</math>)</code> | Toggle the status of the user with ID $d_i$ , e.g., from active to inactive or vice versa, in $O(\log n)$ time                             |
| <code>big_active_range()</code>              | Return an active range $(a, b)$ containing the largest number of active users possible in $O(1)$ time                                      |

**SCRATCH PAPER 1. DO NOT REMOVE FROM THE EXAM.**

You can use this paper to write a longer solution if you run out of space, but be sure to write "Continued on S1" on the problem statement's page.

**SCRATCH PAPER 2. DO NOT REMOVE FROM THE EXAM.**

You can use this paper to write a longer solution if you run out of space, but be sure to write "Continued on S2" on the problem statement's page.

**SCRATCH PAPER 3. DO NOT REMOVE FROM THE EXAM.**

You can use this paper to write a longer solution if you run out of space, but be sure to write "Continued on S3" on the problem statement's page.

MIT OpenCourseWare  
<https://ocw.mit.edu>

6.006 Introduction to Algorithms  
Spring 2020

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>