

Duration: **60 minutes**
Aids Allowed: **NONE** (in particular, no calculator)

Student Number: _____
Last (Family) Name(s): _____ **SOLUTIONS** _____
First (Given) Name(s): _____ **SAMPLE** _____

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above,
*and read the instructions below **carefully**.)*

This term test consists of 3 questions on 5 pages (including this one), printed on one side of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete and write your student number where indicated at the bottom of every page (except page 1).*

Answer each question directly on the test paper, in the space provided, and use the reverse side of the pages for rough work. If you need more space for one of your solutions, use the reverse side of a page and *indicate clearly the part of your work that should be marked.*

If you are unable to answer a question (or part of a question), you will get 20% of the marks for the question (or part of the question) if you state clearly that you do not know how to answer. Note that you will *not* get those marks if your answer contains contradictory statements (such as “I don’t know” followed or preceded by parts of a solution that have not been crossed off).

MARKING GUIDE

1: _____/15

2: _____/15

3: _____/20

TOTAL: _____/50

Good Luck!

Question 1. [15 MARKS]

The array shown below stores a MIN heap. The X'ed entries of the array do not contain heap elements.

Part (a) [5 MARKS]

Show the array that results after applying an EXTRACT-MIN operation to the following heap. Show **only** your final answer. Be sure to indicate which array indices do not contain heap elements.

1	2	3	7	6	5	8	9	X	X	X
---	---	---	---	---	---	---	---	---	---	---

SAMPLE SOLUTION

2	6	3	7	9	5	8	X	X	X	X
---	---	---	---	---	---	---	---	---	---	---

Part (b) [5 MARKS]

Show the array that results after applying an INSERT(4) operation to the following heap. Show **only** your final answer. Be sure to indicate which array indices do not contain heap elements.

1	2	3	7	6	5	8	9	X	X	X
---	---	---	---	---	---	---	---	---	---	---

SAMPLE SOLUTION

1	2	3	4	6	5	8	9	7	X	X
---	---	---	---	---	---	---	---	---	---	---

Part (c) [5 MARKS]

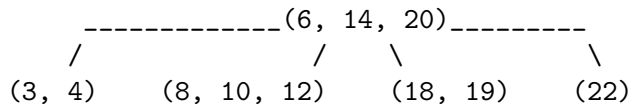
When doing an EXTRACT operation on a heap, some heap elements may be moved to new array locations and other heap elements may remain in the same place. In a heap containing n elements, what is the **exact** maximum number of elements that can end up in a different array location after performing a single EXTRACT operation (do not include the extracted element)?

SAMPLE SOLUTION

Only heap nodes along one leaf-to-root path can end up containing a different element. There are $\lceil \log_2 n \rceil$ levels in the new heap, so the answer is $\lceil \log_2 n \rceil = \lfloor \log_2(n-1) \rfloor + 1$.

Question 2. [15 MARKS]

Consider the following 2-3-4 tree T with integer keys:



Part (a) [5 MARKS]

Of all positive integers not in this tree, list *all* keys whose insertion into T would cause the tree to increase in height (if there are none, write “none”).

SAMPLE SOLUTION

Two possible answers, depending on the insert algorithm used.

1. All keys, if our insert algorithm is 1-pass. When we look at the root, it is full, so it will get split, no matter what we insert.
2. If we use the two-pass algorithm, the root is only split if one of its full children is split. The only option is the (8,10,12) node, so the root is split (and the height of the tree is increased) only if 7, 9, 11 or 13 are inserted.

Part (b) [10 MARKS]

Suppose that we search for a key k in the above tree T such that the following conditions hold: (i) $k = 2^i$ for some $i \geq 1$, and (ii) the probability that we search for a particular k is $1/k$. (Note that $\sum_{i=1}^{\infty} \frac{1}{2^i} = 1$.) Compute the *expected* number of *key comparisons* done by $\text{SEARCH}(T, k)$ under these conditions.

SAMPLE SOLUTION

The keys we may possibly search for are 2, 4, 8, 16, 32, We simply compute the number of key comparisons for each of these keys, and multiply by the probability we are searching for this key to find the expectation.

Let c_i be the number of comparisons searching for key $k = 2^i$. Note that since 32 is larger than anything in the table, $c_i = c_5$ for all $i \geq 5$.

$$\begin{aligned}
 \text{expectation} &= \sum_{i=1}^{\infty} Pr[k = 2^i] \cdot c_i \\
 &= Pr[k = 2]c_1 + Pr[k = 4]c_2 + Pr[k = 8]c_3 + Pr[k = 16]c_4 + Pr[k \geq 32]c_5 \\
 &= 1/2 \cdot 2 + 1/4 \cdot 3 + 1/8 \cdot 3 + 1/16 \cdot 4 + 1/16 \cdot 4 \\
 &= \frac{42}{16} = \frac{21}{8} = 2\frac{5}{8}
 \end{aligned}$$

Question 3. [20 MARKS]

I am hosting a fun run, and need some help designing a data structure to support queries on the results. For each runner, I have the runner's age and finishing time. There are n runners entered in my event, and some runners may take a very, very long time to finish the run.

For each of the problems below, describe concisely and precisely a data structure (based on one seen in this course) that permits an efficient implementation of the given operation within the given time bound (do not prove this bound). Inserting a new runner's result should be possible in $O(\log n)$ expected time.

Part (a) [10 MARKS]

Operation: What is the age of the oldest runner that finished in a time of t or less?

Bound on this operation: $O(\log n)$ worst-case time.

SAMPLE SOLUTION

We use an augmented AVL tree (or B-tree) to store the records. We use time as the key, and at each node add an extra field that stores the age of the oldest finisher in the subtree rooted at this node. On an INSERT, we must update oldest fields along path to root. Clearly, INSERT is $O(\log n)$.

To do the given operation, we do a BST search for the time t . We end up at a node x , with either time t or less. We return the max of $x.age$, $x.left.oldest$, and the left child oldest fields of all nodes on path from x to root. Thus, operation is $O(\log n)$ in worst case.

Part (b) [10 MARKS]

Operation: What is the average finishing time over all runners of age exactly x ?

Bound on this operation: $\Theta(1)$ expected time.

SAMPLE SOLUTION

We have two possible solutions, depending on the assumptions made.

1. Assume age is bounded (say, by 200). Use a direct access table storing number of runners of age i and their aggregate time – $\Theta(1)$ for INSERT – and compute quotient at location x to answer operation – $\Theta(1)$ worst-case time.
2. Assume age is not bounded. Use hash table of n slots. Use age as key for hashing and store only the number of runners of this age and their aggregate time. Insert and find operations take $O(1)$ expected time.