Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Johannes Lengler, David Steurer
Lucas Slot, Manuel Wiedmer, Hongjie Chen, Ding Jingqiu

23 October 2023

Algorithms & Data Structures — Exercise sheet 5 — HS 23

The solutions for this sheet are submitted at the beginning of the exercise class on 30 October 2023.

Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

You can use results from previous parts without solving those parts.

Sorting.

Exercise 5.1 *Sorting algorithms.*

Below you see four sequences of snapshots, each obtained in consecutive steps of the execution of one of the following algorithms: `InsertionSort`, `SelectionSort`, `QuickSort`, `MergeSort`, and `BubbleSort`. For each sequence, write down the corresponding algorithm.

3	6	5	1	2	4	8	7
3	6	5	1	2	4	8	7
3	5	6	1	2	4	8	7

3	6	5	1	2	4	8	7
3	5	1	2	4	6	7	8
3	1	2	4	5	6	7	8

3	6	5	1	2	4	8	7
3	6	1	5	2	4	7	8
1	3	5	6	2	4	7	8

3	6	5	1	2	4	8	7
3	6	5	1	2	4	7	8
3	6	5	1	2	4	7	8

Exercise 5.2 *Guessing an interval* **(1 point)**.

Alice and Bob play the following game:

- Alice selects two integers $1 \leq a < b \leq 200$, which she keeps secret.
- Then, Alice and Bob repeat the following:
  - Bob chooses two integers $a' < b'$.
  - If $a = a'$ and $b = b'$, Bob wins.
  - If $a < a'$ and $b' < b$, Alice tells Bob 'your numbers are strictly between my numbers!'.
  - Otherwise, Alice does not give any clue to Bob.

(a) Bob claims that he has a strategy to win this game in 12 attempts at most. Prove that such a strategy cannot exist.

Hint: *Represent Bob's strategy as a decision tree. Each edge of the decision tree corresponds to one of Alice's answers, while each leaf corresponds to a win for Bob.*

Hint: *After defining the decision tree, you can consider the sequence $k_0 = 1$ and $k_n = 2k_{n-1} + 2$ for $n \geq 1$, and prove that $k_n = 3 \cdot 2^n - 2$ for any $n \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$. The number of vertices in the decision tree should be related to k_n.*

(b)* Can Bob have a strategy to win the game in 13 or 14 attempts?

Hint: *Follow the same strategy as for (a). After defining the decision tree, try to analyse the number of leaves in the decision tree corresponding to Bob's strategy. The sequence $\ell_1 = 1$ and $\ell_n = 2\ell_{n-1} + 1$* for $n > 1$, for which you can prove $\ell_n = 2^n - 1$ for any $n \in \mathbb{N}$, might be helpful.

Exercise 5.3 *Building a Heap* **(1 point)**.

Recall that a binary tree is called *complete* if all of its layers are fully filled, except possibly the last layer, which should be filled from left to right. A *(max-)heap* is a complete binary tree with the extra property that for any node C with parent P,

$$\text{key}(P) \geq \text{key}(C). \qquad \text{(heap-condition)}$$

In this exercise, we formally prove the correctness of the following algorithm from the lecture, which adds a new node with key k to an existing, non-empty heap H. We will show that it performs at most $O(\log n)$ comparisons between keys, where n is the number of nodes in the heap H, and that it maintains the heap structure.

Algorithm 1 Heap insertion

function INSERT(H, k)
Add a new node N with key k to the bottom layer of H, in the left-most free position. If the bottom layer is full, instead create a new layer and add the node in the left-most position.
$P \leftarrow$ the parent of N
while $\text{key}(P) < \text{key}(N)$ **do** ▷ N violates the heap-condition
swap the keys of node N and P.
$N \leftarrow P$
if N is the root node **then**
stop
else
$P \leftarrow$ the parent of N

Let H be a heap consisting of $n \geq 1$ nodes, and let $k \in \mathbb{N}$. Let H' be the data structure that results from executing $\text{Insert}(H, k)$.

(a) Prove that at most $O(\log n)$ comparisons between keys are performed in the execution of $\text{Insert}(H, k)$.

Hint: *After each iteration of the while-loop, what can you say about the* depth *of the node N?*

(b) Let N_{stop} be the final node considered by the algorithm. Prove that all nodes in H' with depth less than or equal to $\text{depth}(N_{\text{stop}})$ satisfy the heap-condition. (A node N satisfies the heap-condition if it is the root node, or otherwise if $\text{key}(N) \leq \text{key}(\text{parent}(\text{N}))$.)

Hint: *Use the fact that H was a heap before we inserted the new node. Consider separately the two different reasons for the algorithm to terminate.*

(c) Let N_{stop} be the final node considered by the algorithm. Prove that all nodes in H' with depth strictly greater than $\text{depth}(N_{\text{stop}})$ satisfy the heap-condition. Using (b), conclude that H' is a heap.

Hint: *Let T be the depth of H'. Use induction to show that after t iterations of the while-loop, the heap-condition is satisfied by all nodes with depth strictly greater than $T - t$.*

Hint: *After swapping the keys of nodes N and P in an iteration of the while-loop, which nodes might potentially no longer satisfy the heap-condition?*

Data structures.

Exercise 5.4 *Implementing abstract data types.*

In the lecture, you saw how we can implement the abstract data type list with operations insert, get, delete and insertAfter. In this exercise, the goal is to see how we can implement two other abstract data types, namely the stack (german "Stapel") and the queue (german "Schlange" or "Warteschlange"). The abstract data type stack is, as the name suggests, a stack of elements. For a stack S, we want to implement the two following operations; see also Figure 1.

- $\text{push}(x, S)$: Add x on top of the stack S.
- $\text{pop}(S)$: Remove (and return) the top element of the stack S.

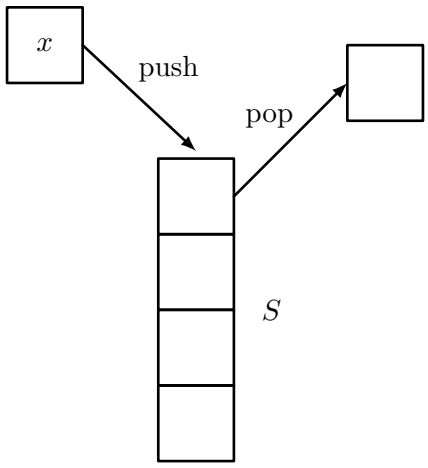


Figure 1: Abstract data type stack

The abstract data type queue is a queue of elements. For a queue Q, we want to implement the following two operations; see also Figure 2.

- $\text{enqueue}(x, Q)$: Add x to the end of Q.
- $\text{dequeue}(Q)$: Remove (and return) the first element of Q.

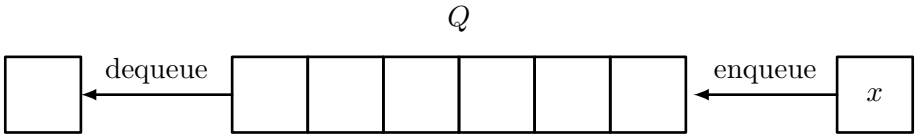


Figure 2: Abstract data type queue

(a) Which data structure from the lecture can be used to implement the abstract data type stack efficiently? Describe for the operations push and pop how they would be implemented with this data structure and what the run time would be.

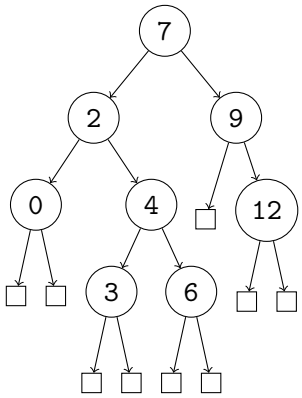
(b) Which data structure from the lecture can be used to implement the abstract data type queue efficiently? Describe for the operations enqueue and dequeue how they would be implemented with this data structure and what the run time would be.

Remark: The following exercise 5.5 is related to the content of the lecture on Tuesday, October 24.

Exercise 5.5 *AVL trees* **(1 point)**.

(a) Draw the tree obtained by inserting the keys 3, 8, 6, 5, 2, 9, 1 and 0 in this order into an initially empty AVL tree. Give also all the intermediate states after every insertion and before and after each rotation that is performed during the process.

(b) Consider the following AVL tree.



Draw the tree obtained by deleting 6, 12, 7 and 4 in this order from this tree. Give also all the intermediate states after every deletion and before and after each rotation that is performed during the process.