Exercise IV, Algorithms 2024-2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. There are many problems on this set, solve as many as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

Heaps and Heapsort

1 Illustrate the steps of HEAPSORT by filling in the arrays and the tree representations on the next page.

2 (Exercise 6.1-4 in the book) Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

3 (half a *, Problem 6-1 in the book) We can build a heap by repeatedly calling MAX-HEAP-INSERT to insert the elements into the heap. Consider the following variation on the BUILD-MAX-HEAP procedure:

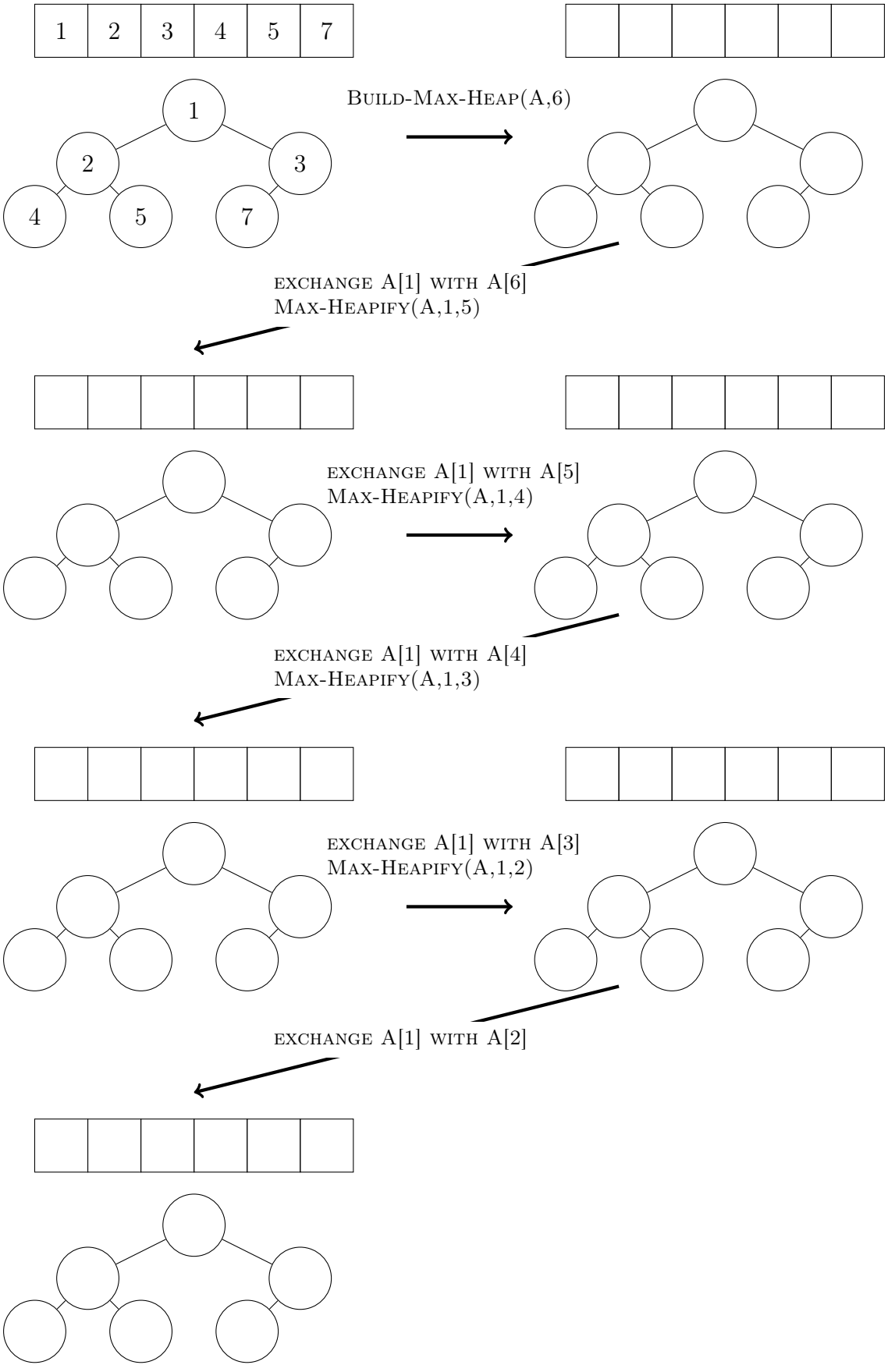
BUILD-MAX-HEAP'(A)
1. A.heap-size $= 1$
2. **for** $i = 2$ **to** A.length
3. MAX-HEAP-INSERT$(A, A[i])$

3a Do the procedures BUILD-MAX-HEAP and BUILD-MAX-HEAP' always create the same heap when run on the same input array? Prove that they do, or provide a counter example.

3b Show that in the worst case, BUILD-MAX-HEAP' requires $\Theta(n \lg n)$ time to build an n-element heap (in comparison to BUILD-MAX-HEAP that we saw in class only needs time $\Theta(n)$).

4 (*, Exercise 6.5-9 in the book) Give an $O(n \lg k)$-time algorithm to merge k sorted arrays into one sorted array, where n is the total number of elements in all the input arrays.

Hint: Use a min-heap of size k for k-way merging.


1 2 3 4 5 7
1
2 3
4 5 7
Build-Max-Heap(A,6)
exchange A[1] with A[6]
Max-Heapify(A,1,5)
exchange A[1] with A[5]
Max-Heapify(A,1,4)
exchange A[1] with A[4]
Max-Heapify(A,1,3)
exchange A[1] with A[3]
Max-Heapify(A,1,2)
exchange A[1] with A[2]

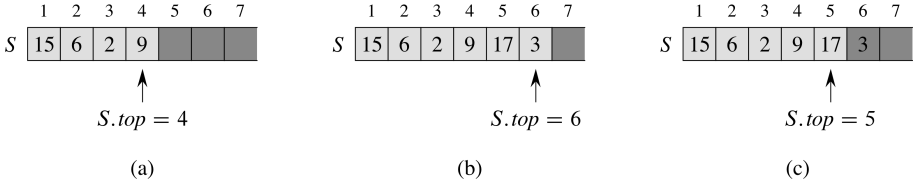
5 *(*, previous exam question)* Consider the following problem:

> **INPUT:** A positive integer k and an array $A[1 \ldots n]$ consisting of $n \geq k$ integers that satisfy the max-heap property, i.e., A is a max-heap.
>
> **OUTPUT:** An array $B[1 \ldots k]$ consisting of the k largest integers of A sorted in non-decreasing order.

Design and **analyze** an efficient algorithm for the above problem. Ideally your algorithm should run in time $O(k \log k)$ but the worse running time of $O(\min\{k \log n, k^2\})$ is also acceptable.

Queues, Stacks, Lists

6 (Exercise 10.1-1 in the book) Using Figure 10.1 in the book (see below) as a model, illustrate the result of each operation in the sequence PUSH(S,4), PUSH(S,1), PUSH(S,3), POP(S), PUSH(S,8), and POP(S) starting from the stack depicted in (a).



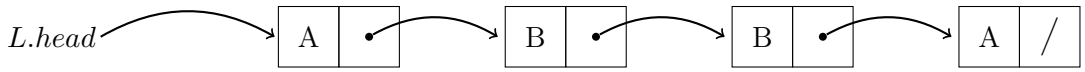
7 (Exercise 10.1-2 in the book) Explain how to implement two stacks in one array $A[1 \ldots n]$ in such a way that neither stack overflows unless the total number of elements in both stacks together is n. The PUSH and POP operations should run in $O(1)$ time.

8 (Exercises 10.2-2 and 10.2-3 in the book)

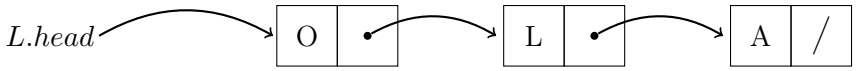
8a Show how to implement a stack by a singly linked list (the operations PUSH and POP should still take $O(1)$ time).

8b Show how to implement a queue by a singly linked list (the operations ENQUEUE and DEQUEUE should still take $O(1)$ time).

9 *(15pts)* **Palindrome.** A word is a palindrome if its reverse is equal to itself. For example, ABBA is a palindrome whereas OLA is not. One way of representing a word in a computer is to have a single-linked list where we have a list-element for each letter. For example, ABBA is represented by the single-linked list



and OLA is represented by the single-linked list



Design and **analyze** an algorithm that, given a pointer to the head of a single-linked list which represents a word, outputs YES if the word is a palindrome and NO otherwise.

For full score your algorithm should run in **linear time** and **should not use any other data structures than single-linked lists**, i.e., no arrays, stacks, queues, etc..

10 (*, Exercise 10.2-7 in the book) Give a $\Theta(n)$-time nonrecursive procedure that reverses a singly linked list of n elements. The procedure should use no more than constant storage beyond that needed for the list itself.

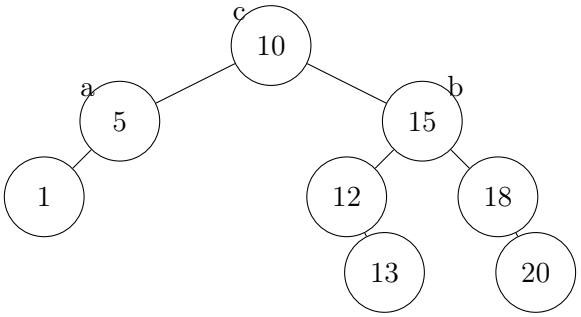
Binary Search Trees

11 Give an $O(\log n)$-time algorithm that takes as input a *sorted* array $A[1 \ldots n]$ of n numbers and a key k, and outputs "YES" if A contains the number k and "NO" otherwise.

12 What is the maximum and minimum height of a binary search tree of n elements? Which tree is better? Motivate your answers.

13 Consider the following binary search tree:

$T =$



Draw the resulting trees obtained after executing each of the following operations (each operation is executed starting from the tree T above, i.e., they are *not* executed in a sequence):

A: TREE-INSERT(T, z) where $z.key = 0$
B: TREE-INSERT(T, z) where $z.key = 17$
C: TREE-INSERT(T, z) where $z.key = 14$
D: TREE-DELETE(T, a)
E: TREE-DELETE(T, b)
F: TREE-DELETE(T, c)

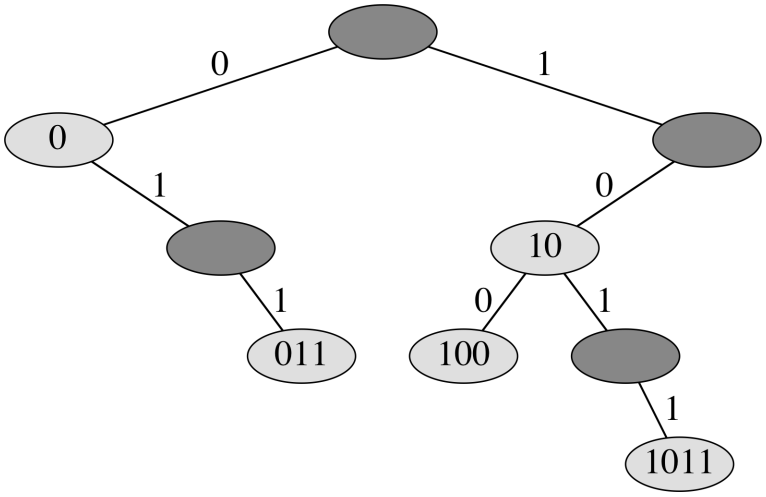


Figure 1. A radix tree storing the bit strings $1011, 10, 011, 100$, and 0. We can determine each node's key by traversing the simple path from the root to that node. There is no need, therefore, to store the keys in the nodes; the keys appear here for illustrative purposes only. Nodes are heavily shaded if the keys corresponding to them are not in the tree; such nodes are present only to establish a path to other nodes.

14 (Exercise 12.1-3)

Give a nonrecursive algorithm that performs an inorder tree walk.

Hint: An easy solution uses a stack as an auxiliary data structure. A more complicated, but elegant, solution uses no stack but assumes that we can test two pointers for equality.

15 (Exercise 12.3-3) We can sort a given set of n numbers by first building a binary search tree containing these numbers (using TREE-INSERT repeatedly to insert the numbers one by one) and then printing the numbers by an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?

16 (*, Problem 12-2) **Radix trees**

Given two strings $a = a_0a_1...a_p$ and $b = b_0b_1...b_q$, where each a_i and each b_j is in some ordered set of characters, we say that string a is ***lexicographically less than*** string b if either

1. there exists an integer j, where $0 \le j \le \min(p, q)$, such that $a_i = b_i$ for all $i = 0, 1, ..., j-1$ and $a_j < b_j$, or
2. $p < q$ and $a_i = b_i$ for all $i = 0, 1, ..., p$.

For example, if a and b are bit strings, then $10100 < 10110$ by rule 1 (letting $j = 3$) and $10100 < 101000$ by rule 2. This ordering is similar to that used in English-language dictionaries.

The ***radix tree*** data structure shown in Figure 1 stores the bit strings $1011, 10, 011, 100$, and 0. When searching for a key $a = a_0a_1...a_p$, we go left at a node of depth i if $a_i = 0$ and right if $a_i = 1$. Let S be a set of distinct bit strings whose lengths sum to n. Show how to use a radix tree to sort S lexicographically in $\Theta(n)$ time. For the example in Figure 1, the output of the sort should be the sequence $0, 011, 10, 100, 1011$.

17 *(old exam question)* **Quadtrees.** A quadtree is a search tree data structure in which each internal node has exactly four children. Quadtrees are usually used to partition the two-dimensional plane by recursively subdividing it into four quadrants or regions. In this exercise, we shall use quadtrees to represent cities according to their geographical position.

Figure 2 shows an example of 4 cities in western Switzerland. It also shows the 4 quadrants induced by the city Fribourg: the first, named NW (for *north-west*), contains the city Neuchâtel; the two following (NE for *north-east* and SE for *south-east*) are empty; the last (SW for *south-west*) contains the two remaining cities, Yverdon and Lausanne.

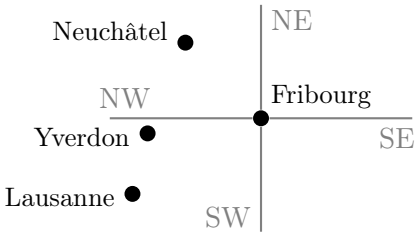


Figure 2. Geographical location of 4 cities in Switzerland

Each node v in the quadtree will store the name ($v.name$) and the coordinates ($v.x$ and $v.y$) of the cities. In addition, the node v will contain a pointer $v.p$ to its parent (or NIL if it's the root) and pointers $v.NW, v.NE, v.SE, v.SW$ to its four children. Similarly as with binary search trees, the key properties that make quadtrees useful are the following:

- if u is in the quadtree rooted by $v.NW$ then $u.x < v.x$ and $u.y \geq v.y$;
- if u is in the quadtree rooted by $v.NE$ then $u.x \geq v.x$ and $u.y > v.y$;
- if u is in the quadtree rooted by $v.SE$ then $u.x > v.x$ and $u.y \leq v.y$;
- if u is in the quadtree rooted by $v.SW$ then $u.x \leq v.x$ and $u.y < v.y$.

Figure 3 shows a possible quadtree representation of the cities in Figure 2. The root is the node corresponding to Fribourg that has x and y coordinates equal to 578461 and 183802, respectively.

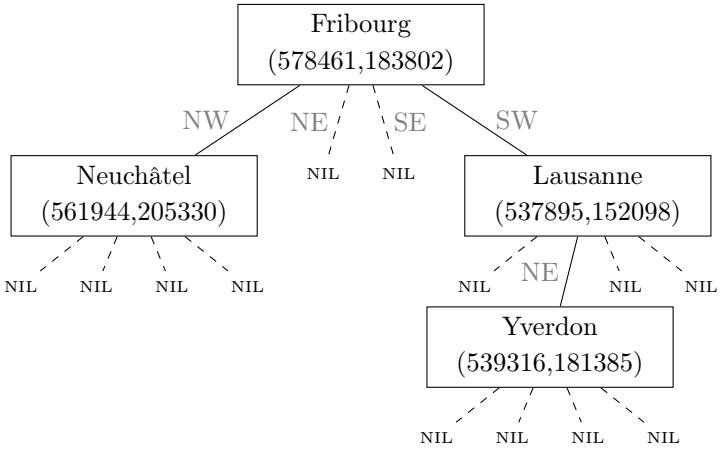


Figure 3. A possible quadtree representation of the cities in Figure 2

Design (give the pseudocode) and **analyze** the running times of the following procedures for the quadtree data structure:

> SEARCH(Q.root, x, y) — prints the name of the city located at (x, y) if such a city exists in the quadtree rooted at $Q.root$.
>
> PRINTSOUTHMOST(Q.root) — prints the name of the southmost city (the city with the smallest y coordinate) in the quadtree rooted at $Q.root$.

Your runtime analyses should be tight for the typical case when the height of the quadtree is logarithmic in the number of cities that it contains. In addition, for full score, the running times of your implementations should not be unnecessarily large.