Practice 2

Exercise levels:
(L1) Reproduce: Reproduce basic facts or check basic understanding.
(L2) Apply: Follow step-by-step instructions.
(L3) Reason: Show insight using a combination of different concepts.
(L4) Create: Prove a non-trivial statement or create an algorithm or data structure of which the objective is formally stated.

▶ Lecture 4 Sorting in linear time

Sorting properties and lower bounds

Exercise 1

(a) *(L1)* Give the definition of a stable sorting algorithm.

(b) *(L2)* Sometimes it is desirable to sort data based on several keys. For example, you may want to sort all exercises by topic, and those with the same topic by complexity. Explain why a stable sorting algorithm can simply sort the data one key at a time.

Exercise 2

(a) *(L4)* Prove, for all $n \geq 2$, that $n-1$ comparisons are sometimes necessary to test whether an array with n distinct elements is sorted in decreasing order.

(b) *(L3)* Show that there is no comparison sort whose running time is linear for at least half of the $n!$ inputs of length n.

Hint: What would the decision tree of such an algorithm look like?

Exercise 3 *(L3)* Professor F. Alsvanouds from the University of Harderwijk has made a discovery. He has figured out how to print the keys of a min-heap with n nodes in (increasing) sorted order in $O(n)$ time. Is this indeed possible? Explain how or prove why not.

Linear-time sorting

Exercise 4 *(L1)* Give the assumptions that must hold for CountingSort, RadixSort, and BucketSort respectively, to run in $O(n)$ time on an input of size n.

Exercise 5 *(L3)* Explain how to make BucketSort run in worst-case $O(n \log n)$ time.

Exercise 6 *(L2)* Let array A contain information about the students in Data Structures using the student IDs as the keys. A student ID is a k-digit integer, that is, an integer in the range $[10^{k-1}, 10^k - 1]$. Assuming that the student IDs are uniformly distributed in the given range, propose an average-case $O(n)$ running time algorithm to sort the array A. (*The running time of your algorithm needs to be independent of k.*)

Exercise 7 *(L4)* Prove using a loop invariant that RadixSort works. Where does your proof need the assumption that the intermediate sort is stable?

Exercise 8

(a) *(L1)* Give a step-by-step description of the working of RadixSort using CountingSort as a subroutine.

(b) *(L2)* Illustrate the execution of RadixSort, using CountingSort as a subroutine, on the following input:

$$\langle 286, 125, 623, 916, 435, 522, 111, 736 \rangle .$$

For the first digit that is sorted, also show the steps of the execution of CountingSort.

(c) *(L2)* RadixSort is normally used to sort integers in a given range (as each value has at most d digits). What if we simply use CountingSort instead? Analyze the running time of CountingSort on an input of n integers with at most d digits each.

Exercise 9 *(L3)* Design an $O(n)$ running time algorithm to sort n words in lexicographic order. Each word has exactly l letters and $l = O(1)$. (*You may assume only basic characters occur.*)

▶ Lecture 5 QuickSort and selection

Exercise 10

(a) *(L2)* Illustrate the execution of Quicksort step by step on the following input:

$$[18, 67, 21, 92, 70, 9, 12, 14, 54, 21]$$

One call to Partition counts as one step.

(b) *(L3)* In general, how many times are two arbitrary keys x and y compared during the execution of Quicksort?

Exercise 11 *(L3)* In the algorithm Select, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7?

Exercise 12

(a) *(L1)* Give a step-by-step description of Quicksort using linear-time median finding as a subroutine to find a good pivot.

(b) *(L2)* Assuming that all the elements of the input array are distinct, analyze the running time of your algorithm. What changes if the elements need not be distinct?

(c) *(L4)* Modify the algorithm to also handle duplicate elements efficiently.

Exercise 13 *(L4)* Let $A[1 : n]$ and $B[1 : n]$ be two sorted arrays, each containing n numbers. Describe an algorithm that finds the median of all $2n$ elements in arrays A and B in $O(\log n)$ time. Don't forget to analyze the running time and prove the correctness of your algorithm.

▶ Lecture 6 Hash tables

Exercise 14 *(L4)* (This is exercise 11.1-1 from the book.) A dynamic set S is represented by a direct-address table T of length m. Describe a procedure that finds the maximum element of S. What is the worst-case performance of your procedure?

Exercise 15

(a) *(L3)* You have a universe of 30 numbers $\{0, 1, \ldots, 29\}$ and a hash-table of size 10. Which hash-function is better, $h_1(k) = k \mod 10$, or $h_2(k) = \lfloor k/10 \rfloor$?

(b) *(L3)* How many different hash functions exist that map from a universe of size n to the integers 1 to m? Assuming that $n = a \cdot m$, how many different uniform hash functions exist?

Exercise 16 *(L3)* Consider the family of hash functions defined by

$$h_{ab}(k) = ((ak + b) \mod p) \mod m,$$

where m is the table size, p is a large prime and $a \geq 1$ and $b \geq 0$ are integers that are at most $p - 1$. As explained in the book, this family of hashing functions is universal. The values p and m are chosen in the design of the algorithm and are fixed. The values a and b are chosen at random when the hash table is initialized. We could instead pick these values either earlier or later, but this may result in changes to the behaviour or analysis of the algorithm.

(a) How does the behavior and analysis of the hashing function change if we pick a and b when designing the algorithm?

(b) How does the behavior and analysis of the hashing function change if we pick a and b later? That is, if we do not fix them when the table is initialized, but pick new random a and b each time the hash function is used?

Exercise 17

(a) *(L2)* Draw the hash table of length $m = 16$ resulting from hashing the keys 8, 12, 40, 13, 88, 45, 29, 20, 23, and 77, using the hash function $h(k) = (3k + 1) \mod 16$ and assuming collisions are handled by chaining. Is $m = 16$ a good choice for the size of a table? Why or why not?

(b) *(L2)* Repeat the previous exercise for a table of size $m = 13$ using the hash function $h(k) = (3k+1) \mod 13$. Why is $m = 13$ a better choice for the size of a hash table than $m = 16$?

(c) *(L2)* What is the result of the previous exercise, assuming collisions are handled by linear probing?

(d) *(L2)* What is the result of (b) assuming collisions are handled by double hashing with a primary hash function $h'(k) = h(k)$ and a secondary hash function $h''(k) = k \mod 15$?

For sub-problems (c) and (d), note for every key how often you had to probe unsuccessfully before you could insert the key, and for each hashing scheme the total number of unsuccessful probes.

Exercise 18 *(L3)* Describe algorithms for Hash-Chain-Insert, Hash-Chain-Delete, and Hash-Chain-Search for a hash table with chaining, where the lists are stored in a sorted order. Analyze the running time of the algorithms.

Exercise 19 *(L4)* Consider two sets of integers, $S = \{s_1, s_2, ..., s_m\}$ and $T = \{t_1, t_2, ..., t_n\}$, $m \leq n$. Describe an $O(n)$ expected time algorithm that uses a hash table of size m to test whether S is a subset of T.

You may assume the existence of a suitable hash function that satisfies the assumption of simple uniform hashing.

Exercise 20 *(L3)* (This is Exercise 11.3-1 from the book.) You wish to search a linked list of length n, where each element contains a key k along with a hash value $h(k)$. Each key is a long character string. How might you take advantage of the hash values when searching the list for an element with a given key?