Exercises for Algorithms and Data Structures Lecture 7

Partitie (1,2);
Divide and conquer (3 t/m 7, 15);
Decrease and conquer (8 t/m 12, 16, 17);
Decrease by half (13 t/m 15)

Question 1. (Levitin, 5.2.8)

Reorder the elements of a given array A so that all negative values precede the positive. The algorithm must be linear (making only one pass through the array) and in situ (internal swaps only). Compare the partitioning (Partition) of the array for Quicksort.

Question 2. (Levitin, 5.2.9.a: Dutch National Flag Problem)

Given an array filled with 'R', 'W', and 'B'. Reorganize this array so that from left to right first all the 'R', then the 'W' and then the 'B' will appear. The algorithm must be linear and in situ (only internal changes). It may only pass through the array once. So no loop.

Question 3. (after Levitin, 5.1.1)

a.

Give a divide-and-conquer algorithm (in pseudocode) for finding the index of the smallest array element from an array of n elements. While doing this the array must be divided into two parts of approximately the same size.

b. (*) Show that for $n = 2^k$ the amount of comparisons that this algorithm computes is $n - 1$.

c. A brute force algorithm for this problem walks from left to right trough the array and holds the lowest value found so far. How many comparisons does this algorithm have to do and which of the two algorithms for this problem is preferable?

Question 4. (after Levitin, 5.2.11)

We have a collection of n nuts of different diameters and n corresponding screws. You may "try" a screw and a nut to see if it fits: you can find out if the screw is too big for the nut, or too small, or that it just fits. However, you can never compare 2 nuts or 2 screws,. The problem now is to find the right screw for every nut.

Give a divide and conquer algorithm (in words) that solves the problem. hint: first put some work into dividing the problem into two (smaller) versions of the problem.

Question 5. (From an old exam)

Given an array A containing n (≥ 2) integers $A[0], A[1], \ldots, A[n-1]$. It is also given that the even positions contain positive numbers (> 0), and the odd positions negative (< 0) numbers. Requested is the amount of pares (i, j) with $i < j$ for which $A[i] + A[j] = 0$.

a. Of which pairs (i, j) are you already sure that $A[i] + A[j] \neq 0$?

To avoid unnecessary comparisons, the pairs mentioned at **a.** may not be used at **b.** and **c.**.

b. Give a simple (brute force) algorithm in Python that returns the requested quantity. What is the complexity of your algorithm (as a function of n)?

c. give a divide-and-conquer algorithm in Python for the problem above. To do this, divide the array into two equal parts. Assume that n (≥ 2) is a power of 2. So a recursive Python function `int amount2(A,left,right)` must be written that solves the problem for the subarray $A[\texttt{left}], \ldots, A[\texttt{right}]$ in length of power of 2.

Question 6. (Levitin, 5.4.2)

Calculate $1201 * 2430$ by applying the divide & conquer (divide & conquer) algorithm that was descibed in college and Levitin 5.4.

Question 7. (after Levitin, 5.5.1)

a. We consider the one-dimensional closest-pair problem: given a set of n real numbers, find the two numbers that lie closest to each other. Give a divide & conquer algorithm and sort the points first.

b. Also provide a non-recursive algorithm.

Question 8. (Levitin, 4.1.1)

A group of n soldiers has to cross a river. There is no bridge, only two boys with a rowboat. The boat can only carry the 2 boys (or 1 of them) without sinking, or á single soldier. *Question:* how can all soldiers get to the come to the other side so that each time at least one of the two boys is on the same bank(side) where the boat is? Give a decrease and conquer algorithm. How often does the boat have to cross? Hint: bring the problem back from n to $n-1$.

Question 9. (Levitin, 4.1.2.a)

There are $2n$ glasses next to each other in a line. The first n of these glasses are filled with lemonade, the others are empty. Give a decrease and conquer algorithm that, by pouring over (emptying a full glass into an empty glass), ensures that the row of glasses becomes full, empty, full, empty, etc. So after each empty glass comes a full glass and after each full glass comes an empty one.

Question 10.

Write a decrease by one algorithm (in pseudocode or Python) that generates all 2^n bit strings of length n.

Question 11.

Read on page 173-174 of Levitin what Gray-codes are and make the following exercises.

a. (Levitin, 4.3.9.a) Use the decrease-by-one method to generate the Gray-code for $n = 4$.

b. (Levitin, 4.3.9.b) Apply the following non-recursive algorithm to generate a Gray code for $n = 4$:

Start with a string with n 0's.

For $i = 1, 2, \ldots, 2^n - 1$, generate the ith bit string by 'flipping' the bth bit in the previous bitstring, where b is the position of the least significant (= last) 1 in the binary representation of i.

Question 12. (From an old exam)

Given an array A ($A[1], \ldots, A[n]$, with $n \geq 2$), that contains an equal amount of odd and even numbers. The odd numbers are on the odd positions and the even numbers on the even positions. The array must —via substitutions— be rearranged so that all odd numbers will be at the front, and all even numbers will be at the back.

a. Give a simple iterative algorithm for this problem that only uses one for-loop. How many exchanges does your algorithm do?

b. Give a decrease by four algorithm (in pseudocode or Python) for this problem. Assume that n is a multiple of 2. So a recursive function `shuffle(i,j)` has to be written which solves the problem for the subarray $A[i], \ldots, A[j]$ in length a multiple of 2.

Question 13. (after Levitin, exercises 4.4.4 and 4.4.5)

Linear search has the same complexity when you implement the list that has to be searched via a single-connected pointer list, as when you implement it via an array. What about binary search? (Of course we assume that the list is already sorted.)

Question 14. (From an old exam)

Given an array A with n (≥ 1) different integers $A[0], A[1], \ldots, A[n-1]$. It is also given that there is an index p with $0 \leq p \leq n-1$ so that A is going up until it reaches index p, and after p A will go down.
Example: if $A = 3\ \ 6\ \ 9$
11
$8\ \ 2$, then $p = 3$. *Edge cases*: if $A = 7\ \ 5\ \ 3\ \ 2\ \ 1$, then $p = 0$; if $A = 5$ (so consists of 1 element), then $p = 0$; if $A = 4\ \ 9$, then $p = 1$.

Give a decrease-by-half algorithm (in pseudocode or Python) for determining this index p and explain why it works.

Question 15. (From: Exam 2013)

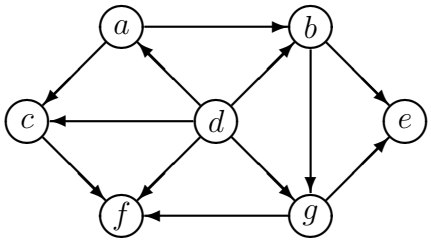
Given an array $A = A[0], A[1], \ldots, A[n-1]$ that contains n (≥ 2) different integers. Assume that n is a power of 2. We are looking for the *largest* index j for which holds that $A[j] < j$. For $9, 4, 1, 7, 11, 13, 3, 5, 8$ this index equals 7. If no such index exists, the algorithms to be written must yield -1.
We will solve the problem first with brute force and then with divide and conquer.

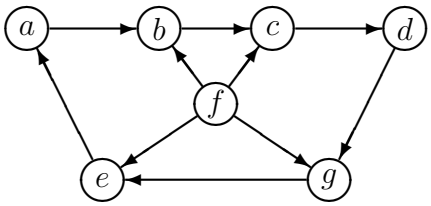
a. Write a simple brute force algorithm in Python that returns this index.

b. Give a divide-and-conquer algorithm in Python for the problem. The array has to be divided into two equal parts. To do this, write a *recursive* Python function `int largest2(int A[ ], int left, int` which solves the problem for the partial array A[`left`], ..., `A`[`right`] of the length of a power of 2.

c. We now assume that the array is sorted in ascending order. Give a decrease-by-half algorithm in Python for determining the requested index. To do this, write a *recursive* Python function `int largest3(int A[ ], int left, int right)`. Explain why your algorithm works, so why you only need to look at one of both halves at a time. Use what information is provided about the array (different integers, sorted ascending).

Question 16. **a.** (Levitin, 4.2.1) Apply the DFS-based algorithm to solve the topological sorting problem for the following digraphs.





b. Apply the source-removal algorithm to solve the topological sorting problem for the same two digraphs.

Question 17. **a.** (Levitin, 4.2.6.a) Prove that a nonempty DAG must have at least one source.

b. (Levitin, 4.2.2.a) Prove that the topological sorting problem has a solution, if and only if the digraph is a DAG.

c. (Levitin, 4.2.2.b) For a digraph with n vertices, what is the largest number of distinct solutions the topological sorting problem can have?