KØBENHAVNS UNIVERSITET

Introduktion til diskret matematik og algoritmer: Problem Set 2

Due: Wednesday February 28 at 9:59 CET.
Submission: Please submit your solutions via *Absalon* as a PDF file. State your name and e-mail address close to the top of the first page. Solutions should be written in LaTeX or some other math-aware typesetting system with reasonable margins on all sides (at least 2.5 cm). Please try to be precise and to the point in your solutions and refrain from vague statements. Make sure to explain your reasoning. *Write so that a fellow student of yours can read, understand, and verify your solutions.* In addition to what is stated below, the general rules for problem sets stated on *Absalon* always apply.
Collaboration: Discussions of ideas in groups of two to three people are allowed—and indeed, encouraged—but you should always write up your solutions completely on your own, from start to finish, and you should understand all aspects of them fully. It is not allowed to compose draft solutions together and then continue editing individually, or to share any text, formulas, or pseudocode. Also, no such material may be downloaded from or generated via the internet to be used in draft or final solutions. Submitted solutions will be checked for plagiarism.
Grading: A score of 120 points is guaranteed to be enough to pass this problem set.
Questions: Please do not hesitate to ask the instructor or TAs if any problem statement is unclear, but please make sure to send private messages—sometimes specific enough questions could give away the solution to your fellow students, and we want all of you to benefit from working on, and learning from, the problems. Good luck!

1 (100 p) Suppose that we are given an array $A = [5, 6, 4, 7, 3, 8, 2, 9, 1, 10]$ to be sorted in increasing order.

1a (30 p) Run insertion sort by hand on this array (as the lecturer has been doing in class), and describe in detail in every step of the algorithm how the elements in the array are moved.

Solution: When using insertion sort we consider the array to consist of a sorted part, which starts out as the first element on the left, and an unsorted part, which is the rest. In every iteration of the insertion sort loop, we take the leftmost element in the unsorted part and place it in the correct position in the current sorted part. This goes as follows:

1. In the first iteration, we take the second element 6 and compare it with the first element 5. No swap is necessary since these elements are correctly ordered.

2. In the second iteration, we consider the third element 4. First this element is compared to 6, and since $4 < 6$ we shift 6 to the right. Then we compare to 5, and since $4 < 5$ we shift 5 to the right. Now there is nothing more to compare with, and so we place 4 in the first position. The whole array now looks like $[4, 5, 6, 7, 3, 8, 2, 9, 1, 10]$.

3. In the third iteration, we start by comparing 7 to 6. Since $7 > 6$ these two numbers are correctly sorted and we terminate immediately without changing anything.

4. In the fourth iteration, we compare 3 with the sorted part $[4, 5, 6, 7]$ of the array. Just as was the case for number 4 in the second iteration, we will shift all numbers right to make room for 3, which ends up in the leftmost position, resulting in an array looking like $[3, 4, 5, 6, 7, 8, 2, 9, 1, 10]$.

5. In the fifth iteration we terminate immediately, just as in the third, since $8 > 7$. Nothing changes except that we know that the sorted part of the array has grown by one element.

It is not hard to see that the rest of the iterations will alternate in the same way between shifting the new numbers all the way to the left (for 2 and 1) and leaving them in place (for 9 and 10). This concludes our description of how the insertion sort algorithm sorts the array.

1b (30 p) Run merge sort by hand on this array (as the lecturer has been doing in class), and show in detail in every step of the algorithm what recursive calls are made and how the results from these recursive calls are combined.

Solution: See Figure 1 for an illustration of what happens in the algorithm. (For your own solutions, please note that making legible hand-drawing and including scanned pictures in your typed up solution is perfectly fine if you do not want to typeset pictures in LaTeX. Just for your information, the pictures in this solution is done in the METAPOST programming language that is part of the LaTeX ecosystem.)

We first recursively split the list in two part, where the first part is one element larger if the list size is odd, until all lists have size 1. This leads to the split lists in the leaves of the recursion tree in Figure 1a.

In the merge phase we work bottom up from the level above the leaves. Every parent node P takes its two children lists C_1 and C_2, which have previously been sorted, and merges them. We do so by placing to pointers e_1 and e_2 at the start of C_1 and C_2, respectively, and then adding the smallest of these elements to the list under construction after which the corresponding pointer is advanced.

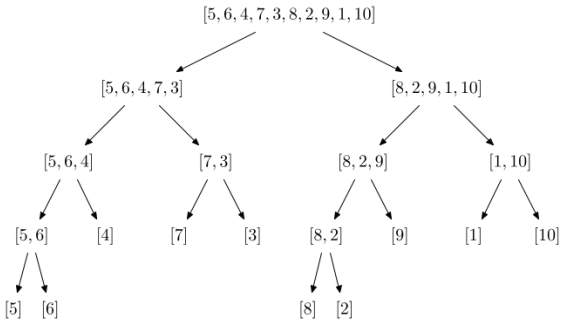
For instance, for the two leftmost leaves we have $5 < 6$, so the merge step just concatenates to get $[5, 6]$. Continuing on the left, we now want to merge $[5, 6]$ and $[4]$. We have $e_2 = 4 < 5 = e_1$, so 4 is added first, and since this empties list C_2 we then add 5 and 6 to get the list $[4, 5, 6]$. When $[4, 5, 6]$ is merged with $[3, 7]$, then 3 is the smallest element and goes first. After this, 4, 5, and 6 are all smaller than 7, which goes last. We get the list $[3, 4, 5, 6, 7]$. The recursive calls in the right subtree of the root are dealt with in an analogous fashion.

In the final step, we need to merge $[3, 4, 5, 6, 7]$ and $[1, 2, 8, 9, 10]$. Here we start with $e_2 = 1 < 3 = e_1$, so 1 goes first, updating e_2 to 2. Now we still have $e_2 = 2 < 3 = e_1$, so we add 2 to the list under construction, updating e_2 to 8. Since 8 is larger than all elements in C_2, this whole list will be added, after which 8, 9, and 10 will be added from list C_1. This produces the sorted list at the root of the tree in Figure 1b.

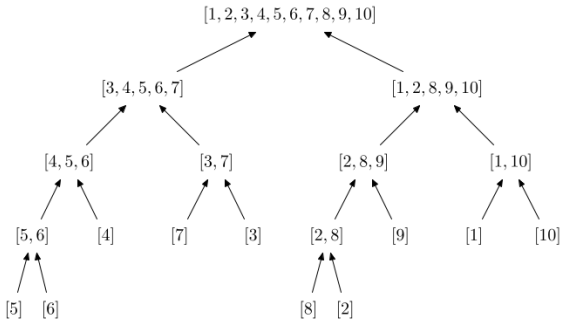
1c (20 p) Suppose that we are given another array B that is already sorted in increasing order. How fast do the insertion sort and merge sort algorithms run in this case? Is any of them asymptotically faster than the other as the size of the array B grows?

Solution: Merge sort always runs in time $\Theta(n \log n)$. One simple way of seeing this is that if we merge two lists $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$, then at least one element in every pair (a_i, b_i) will be compared to an element in the other list. We will merge lists in (roughly) $\log n$ levels from sizes 1 up to $n/2$, and by the above argument we will have $\Omega(n)$ comparisons at each level.

Insertion sort will just verify that the list is already sorted. In every iteration, the first unsorted element is found to be larger than the largest sorted element, so the iteration terminates



(a) Tree of recursive calls splitting list.



(b) Bottom-up sorting of split lists.

Figure 1: Merge sort on array $A = [5, 6, 4, 7, 3, 8, 2, 9, 1, 10]$.

immediately. The whole algorithm will thus run in linear time, which is asymptotically faster than what we get for merge sort.

1d (20 p) Suppose that we are given a third array C that is sorted in *decreasing* order, so that it needs to be reversed to be sorted in the order that we prefer, namely increasing. How fast do the insertion sort and merge sort algorithms run in this case? Is any of them asymptotically faster than the other as the size of the array C grows?

Solution: As already explained, merge sort always runs in time $\Theta(n \log n)$.

If the array C is sorted in decreasing order, then in every iteration the first unsorted element will be placed in the leftmost position, and all other elements in the sorted part will need to be shifted right. This means that the ith iteration needs to shift $i - 1$ elements, and so the total running time will be $\Theta(n^2)$, which is asymptotically slower than merge sort.

2 (60 p) Post-pandemic life in academia has meant noticeable changes for both Jakob and his family, in ways both good and bad.

2a (20 p) In the autumn of 2022, Jakob attended his first non-virtual international conference in a very long time (since February 2020, to be precise), and the conference had more than 100 participants. A colleague at the conference pointed out to Jakob that this meant that either there were participants from more than 10 different countries, or else more than 10 people from some particular country attended the conference. Jakob, who had not studied the list of participants that closely, was amazed at this claim. Can you explain to Jakob, without even having looked at the list of participants, why the claim has to be true?

Solution: If there are participants from more than 10 countries, then there is nothing to prove. Suppose therefore that the strictly more than 100 participants are coming from at most 10 different countries. Then the pigeonhole principle (with participants as pigeons and countries as pigeonholes) says that at least 11 particpants are coming from the same country.

2b (40 p) While Jakob was away, he suggested to his children that they should entertain themselves at home with the following game: Write down the numbers 1 to 20 on a sheet of paper. Erase any two distinct numbers a and b and replace them by the number $a+b-1$. Now do the same again with all numbers currently on the sheet, i.e., pick any two members

a' and b' of the multi-set $\big(\{1, 2, \ldots, 20\} \setminus \{a, b\}\big) \cup \{a + b - 1\}$ of all numbers between 1 and 20 except a and b plus the new number $a + b - 1$, and replace the two chosen numbers by their sum minus one $a' + b' - 1$. Repeat this procedure until there is only one single number left on the sheet of paper (i.e., until the multi-set has size 1). The children found this game a little bit repetitive, however. Can you describe the range of possible outcomes for this game? For a full score, provide proofs of any claims you make.

Solution: The sum of all numbers on the paper before the game starts is $\sum_{i=1}^{20} i = \frac{20 \cdot 21}{2} = 210$. After the first two numbers a and b have been erased and replaced by $a + b - 1$, the sum of all numbers is $210 - 1 = 209$, and this holds regardless of which numbers a and b were chosen. When in the second step two new numbers a' and b' are erased and replaced by $a' + b' - 1$, the sum changes to $(210 - 1) - 1 = 210 - 2 = 208$, and again this fact does not depend on the concrete choice of a' and b'. Continuing this reasoning, we see that it is an invariant that after i numbers have been erased, the sum of the numbers currently written on the sheet of paper must be $210 - i$. The game ends after 19 numbers have been erased, at which point the final remaining number (which is also the sum of all numbers currently written on the sheet of paper) is $210 - 19 = 191$.

If we would want to set up a formal induction proof, then we could pick as our induction hypothesis that "after i numbers have been erased, the sum of the numbers currently written on the sheet of paper is $210 - i$", and then prove the base case and induction step as per the reasoning above. You would certainly be more than welcome to do this, and, in general, it is a good idea to write out proofs carefully to make certain that you are not skipping any logical steps. But for this problem, as long as you explain clearly why the final answer has to be 191, which you can do by explaining clearly why it is an invariant that after i numbers have been erased the sum of the remaining numbers must be $210 - i$, then you do not have to set this proof up as an induction proof. (What we care about is that your formal *reasoning* is correct, not that your formal *format* is.)

3 (80 p) Provide formal proofs of the following claims using proof techniques that we have learned during the course.

3a (20 p) For all $s \in \mathbb{N}$ and all $k > 0$ it holds that $1 + sk \leq (1 + k)^s$.

Solution: First, we can note that the condition $k > 0$ in the problem statement is overly cautious—actually, $k > -1$ is enough, and we will see this in the proof below (although this requires some extra care in the argument which we do not need to worry about if we make k strictly positive). Second, since k could be any positive real number we cannot do inductive reasoning over k, but we can try to do induction over the integer s as follows.

Base case: For $s = 0$ we have $1 + 0 \cdot k = (1 + k)^0 = 1$ (and so the inequality $1 + sk \leq (1 + k)^s$ certainly holds for $s = 0$).

Induction step: Suppose that $1 + sk \leq (1 + k)^s$ holds and consider $s + 1$. We have

$$\begin{aligned}
(1 + k)^{s+1} &= (1 + k)(1 + k)^s \\
&\geq (1 + k)(1 + sk) && \text{[by the induction hypothesis (and since } 1 + k > 0\text{)]} \\
&= 1 + sk + k + sk^2 && \text{[multiplying out]} \\
&= 1 + (s + 1)k + sk^2 && \text{[rearranging terms]} \\
&\geq 1 + (s + 1)k && \text{[since } sk^2 \geq 0 \text{ as long as } s \geq 0\text{]}
\end{aligned}$$

which is the desired inequality.

The claim now follows by the induction principle.

Another way of establishing the inequality is to just do binomial expansion to compute

$$(1+k)^s = \binom{s}{0}1^s k^0 + \binom{s}{1}1^{s-1}k^1 + \ldots \binom{s}{s}1^0 k^s = 1 + sk + \cdots + k^s \geq 1 + ks$$

where the final inequality can be seen to hold since $k > 0$ (but note that the first proof presented above does not need this stronger assumption).

3b (30 p) For the sequence (T_i) defined by

$$T_i = \begin{cases} 0 & \text{for } i = 1, \\ 1 & \text{for } i = 2, \\ T_{i-1} + T_{i-2} & \text{for } i \geq 3, \end{cases}$$

it holds for all $i \geq 2$ that

$$1 \leq \frac{T_{i+1}}{T_i} \leq 2 \; .$$

Solution: We prove this by induction over the numbers in the sequence, i.e., by induction over i. As a side note, the inequalities are actually sharp (i.e., $<$ rather than $\leq$) for $i \geq 4$, and this is easily seen from the proof that will follow below, but since the problem statement does not say anything about this, you absolutely do not have to point this out in your solution (though you are certainly welcome to do so).

In order to make the proof clearer and avoid hidden assumption, we start by spelling out in full detail what induction hypotheses we will need.

Induction hypotheses: It holds for i that

1. $T_i > 0$ and $T_{i+1} > 0$;
2. $T_{i+1} \geq T_i$;
3. $T_i \leq T_{i+1} \leq 2 \cdot T_i$.

Note that we get the statement we need from this by dividing the inequalities in item 3 by T_i, which is in order since $T_i > 0$ by item 1.

Base case: For the base case $i = 2$ it is straightforward to verify for $T_2 = 1$ and $T_3 = T_2 + T_1 = 1$ that all of the induction hypotheses hold. (And yes, just writing like this in your solutions is perfectly in order, because you have said exactly what needs to be done, and the verification that what you say is true is fully mechanical and thus trivial.)

Induction step: Suppose that the induction hypotheses hold for i and consider $i+1$. Let us consider the hypotheses one by one.

1. Since T_i and T_{i+1} are both strictly positive by the induction hypotheses, it holds that $T_{i+2} = T_{i+1} + T_i$ is the sum of two strictly positive numbers and hence also strictly positive.
2. Since $T_i > 0$ by the induction hypotheses, we have $T_{i+2} = T_{i+1} + T_i > T_{i+1}$.
3. Since $T_{i+1} \geq T_i$ by the induction hypotheses, we can plug this into the recurrence relation to get $T_{i+2} = T_{i+1} + T_i \leq 2 \cdot T_{i+1}$. (And note that since $T_4 > T_3$ we could get a strict inequality here if we wanted to.)

The inequalities follow by the induction principle.

3c (30 p) For the sequence (a_n) defined by

$$a_n = \begin{cases} 5 & \text{if } n = 1, \\ 13 & \text{if } n = 2, \\ 5a_{n-1} - 6a_{n-2} & \text{if } n > 2, \end{cases}$$

it holds for all positive integers n that $a_n = 2^n + 3^n$.

Solution: We prove the claim by induction. Note that in the induction step we will need to talk about both a_{n-1} and a_{n-2}, and therefore our induction hypothesis will need to assume that both a_{n-1} and a_{n-2} are on the right form. For the same reason, we need two base cases.

Base case ***(*** $n = 1$ ***and*** $n = 2$***):*** We have $a_1 = 5 = 2^1 + 3^1$ and $a_2 = 13 = 2^2 + 3^2$.

Induction step: Suppose that $a_{n-2} = 2^{n-2} + 3^{n-2}$ and $a_{n-1} = 2^{n-1} + 3^{n-1}$, and consider $a_n = 5a_{n-1} - 6a_{n-2}$. Calculating, we get

$$\begin{aligned} a_n &= 5a_{n-1} - 6a_{n-2} && \text{[by definition]} \\ &= 5\big(2^{n-1} + 3^{n-1}\big) - 6\big(2^{n-2} + 3^{n-2}\big) && \text{[by the induction hypothesis]} \\ &= 2 \cdot 2^{n-1} + 3 \cdot 2^{n-1} + 3 \cdot 3^{n-1} + 2 \cdot 3^{n-1} \\ &\qquad - 3 \cdot 2 \cdot 2^{n-2} - 2 \cdot 3 \cdot 3^{n-2} && \text{[regrouping]} \\ &= 2^n + 3^n + 3 \cdot 2^{n-1} + 2 \cdot 3^{n-1} - 3 \cdot 2^{n-1} - 2 \cdot 3^{n-1} \\ &= 2^n + 3^n \end{aligned}$$

as desired, and so the induction step goes through. The claimed equality now follows by the induction principle.

One remark is that instead of having an induction hypothesis with claims about both a_{n-1} and a_{n-2}, we could simply use strong induction. We would still need to establish two base cases for the strong induction to go through, though.

Another comment is that it is also possible to solve this problem by using material from Section 3.5 on recurrence relations in the KBR textbook (more specifically, Theorem 1 on page 115). Section 3.5 in KBR is not part of the required reading for the course, but a fully correct solution using this approach will yield a full score.