Eidgenössische
Technische Hochschule
Zürich

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

Departement of Computer Science
Markus Püschel, David Steurer
Johannes Lengler, Gleb Novikov, Chris Wendler

21. October 2019

Algorithms & Data Structures — Exercise sheet 5 — HS 19

Exercise Class (Room & TA): _____

Submitted by: _____

Peer Feedback by: _____

Points: _____

Submission: On Monday, 28 October 2019, hand in your solution to your TA *before* the exercise class starts. Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

Exercise 5.1 *Loop invariant* **(1 Point)**.

Consider the pseudocode of the bubble sort algorithm on an integer array $a[0, \ldots, n-1]$, $n \geq 1$.

```
procedure BubbleSort(a)
    for 0 ≤ i < n do
        for 0 ≤ j < n − i − 1 do
            if a[j] > a[j + 1] then
                t ← a[j]
                a[j] ← a[j + 1]
                a[j + 1] ← t
```

Bubble sort satisfies the loop invariant INV(m) for $0 \leq m \leq n$: *After m iterations of the outer for-loop, the subarray $a[n-m, \ldots, n-1]$ is sorted and each element from $a[0, \ldots, n-m-1]$ is not greater than each element from $a[n-m, \ldots, n-1]$*. Here we assume that $a[n, \ldots, n-1]$ is empty, so INV(0) trivially holds.

Prove that:

a) If INV(m) holds, then INV$(m+1)$ holds (for all $0 \leq m < n$).

b) INV(n) implies the correct solution.

Solution:

a) Assume that INV(m) holds. At the $(m+1)$-th iteration of the outer loop consider the maximal element with maximal index among $a[0, \ldots, n-m-1]$. Let's denote its value by z and its index by k. Let's prove by induction over m' that for $m' = k, \ldots, n-m-1$ after m' iterations of the inner for-loop, $a[m'] = z$.

   Base case: $m' = k$. At this iteration j is $m' - 1 = k - 1$, the condition $a[j] > a[j+1]$ is false, hence we do not swap elements and after the execution of loop $a[m'] = a[j+1] = a[k] = z$. Assume that for some m' in $k, \ldots, n-m-2$ after m' iterations of the inner for-loop, $a[m'] = z$. At the beginning of the next iteration j becomes m', so $a[j] = z$. By maximality of z and the fact that every element

from $a[k+1, \ldots, n-m-1]$ is strictly smaller than z, $a[j] > a[j+1]$ holds and we swap $a[j]$ and $a[j+1]$, so after the iteration $a[m'+1] = a[j+1] = z$. By the principle of mathematical induction, the statement is true for all $m' = k, \ldots, n-m-1$.

Hence after $m+1$ iterations of the outer loop $a[n-m-1] = z$. For all $x \in a[0, \ldots, n-m-2]$, $x \leq z$ and $a[n-m-1, \ldots, n-1]$ is sorted since for all $y \in a[n-m, \ldots, n-1]$, $z \leq y$. Therefore, INV$(m+1)$ holds.

b) INV(n) implies that the whole array $a[0, \ldots, n-1]$ is sorted.

Exercise 5.2 *Decision tree* **(1 Point)**.

In the lecture you saw a proof of the lower bound for the number of comparisons in sorting by arguing that every sorting algorithm (based on comparisons) corresponds to a decision tree. Consider the pseudocode of straight mergesort from the script:

```
procedure StraightMergesort(A[1, ..., n])
    length ← 1
    while length < n do
        right ← 0
        while right + length < n do
            left ← right + 1
            middle ← left + length − 1
            right ← min(middle + length, n)
            Merge(A, left, middle, right)
        length ← 2 · length
```

Sketch the decision tree for $n = 4$, draw enough of it to determine the number of leaves and the height (assume the root has height 0). How many comparisons are done in the worst case?

Hint: *The comparisons occur in the subroutine Merge.*

Solution: The tree is illustrated on Figure 1. For better readability, we do not draw the red and the blue subtree completely. The blue subtree can be obtained from the subtree with a blue root ($A[3] < A[4]$) by swaping $A[1]$ and $A[2]$. Similarly, the red subtree can be obtained from the subtree with a red root ($A[1] < A[3]$) by swaping $A[3]$ and $A[4]$. A tree with a root $A[i] < A[j]$ has two subtrees, the left one corresponds to the case when $A[i] < A[j]$, the right one corresponds to the case when $A[i] \geq A[j]$.

This tree has 24 leaves (each leaf corresponds to a permutation). The height of the tree is 5 and 5 comparisons should be performed in the worst case.

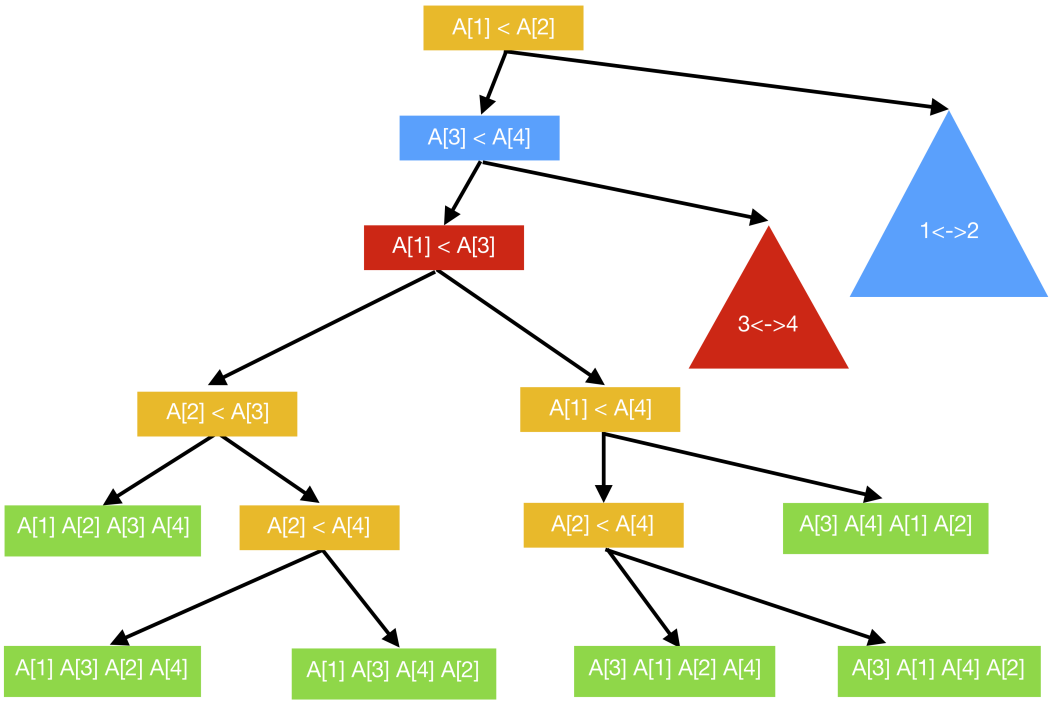

Figure 1: A decision tree of mergesort for $n = 4$.

Exercise 5.3 *Inverse questions* **(1 Point)**.

In this exercise assume that Selection Sort does not swap elements with the same index.

a) Give a sequence of 6 numbers for which Selection Sort performs exactly 3 swaps of keys in order to sort the sequence.

Solution: For example, $1, 6, 5, 2, 3, 4$.

b) For all $n > 1$ give a sequence of n numbers for which Selection Sort performs exactly 1 swap of keys in order to sort the sequence, but Bubble Sort and Insertion Sort perform at least $\Omega(n)$ swaps of keys.

Solution: For example, a sequence

$$n, 2, 3, 4, \ldots, n-1, 1.$$

c) For all $n > 1$ give a sequence of n numbers for which Bubble Sort, Selection Sort and Insertion Sort perform $\Theta(n)$ swaps of keys in order to sort the sequence.

Solution: For example, if n is even, a sequence

$$2, 1, 4, 3, 6, 5, \ldots, n, n-1,$$

and if n is odd, a sequence

$$2, 1, 4, 3, 6, 5, \ldots, n-1, n-2, n.$$

d) For all $n > 1$ give a sequence of n numbers for which Bubble Sort performs $\Theta(n^{1.9})$ swaps of keys in order to sort the sequence.

Solution: For example, a sequence

$$m, m-1, m-2, \ldots, 3, 2, 1, m+1, m+2 \ldots, n-1, n,$$

where $m = \lfloor n^{0.95} \rfloor$.

Exercise 5.4 *iPhone Drop Test.*

You just got a new job at Apple in the department of destructive testing. The first task given is to test the endurance of the new iPhone 11 series. Specifically you need to determine the highest floor that the new iPhone can withstand when dropped out of the window.

When the phone is dropped and does not break, it is undamaged and can be dropped again. For simplicity assume that subsequent drops of the phone do not affect its endurance (i.e. if it survives it will have the identical state as if it weren't dropped at all). However, once the iPhone has been broken, you can no longer use it for another test.

If the phone breaks when dropped from floor n, then it would also have broken from any floor above that. If the phone survives a fall, then it will survive any fall below that. Moreover, the phone always survives the fall from ground floor ($n = 0$), so you don't have to drop it from the ground floor.

As this is your first responsibility at your new job, you want to impress your new boss, and deliver results as soon as possible. To achieve that, you devise a strategy to minimize the number of drop tests required to find the solution.

a) What strategy would you use if only one phone is given and you perform the drop test on a building with n floors? What are the maximum number of drop tests that you have to perform?

Solution: If we have one phone on disposal, we really have no other choice but to start at floor 1. If it survives, great, we go up to floor 2 and try again, then floor 3 and all the way up the building; one floor at a time. As a result, it will take us n trials in the worst case scenario.

b) What if we are given unlimited amount of identical phones and a building with $n = 2^k$ floors, where $k \in \mathbb{N}$?

Solution: In that case scenario, we can start at floor $n/2$. If the phone breaks, we repeat this procedure with the block of floors $1, \ldots, n/2 - 1$, if not, we repeat this procedure with the block of floors $n/2 + 1, \ldots, n$, essentially performing a binary search. Assuming worse case scenario, it will take us $\log_2 n + 1$ trials.

c) Assume that you are given exactly 2 identical phones and a building with n floors. Devise a search strategy for the floor where the phone breaks that requires at most $\mathcal{O}(\sqrt{n})$ drops. Prove the runtime bound.

Hint: *Consider the inverse problem: Assume you want to spend at most $\sqrt{n}$ rounds with the first phone (first phase), and afterwards at most $\sqrt{n}$ rounds with the second phone (second phase). When the first phone breaks, then you are left with the problem of determining the correct floor among the remaining possibilities with only one more phone. Note that you have already analysed this problem in part (a). Now keep in mind that your first phone might break with your first try. How does this limit your choice for the very first step with the first phone? What about subsequent steps? Can you cover all n floors with this strategy?*

Solution: We can divide the building into $\mathcal{O}(\sqrt{n})$ blocks (sequences of consecutive floors) of size $\lfloor\sqrt{n}\rfloor$. Starting from the first block we drop the first phone from the last floor of each block until the phone breaks. If the phone didn't break after drop from the n-th floor, the lowest floor where the phone breaks is higher than n (so we cannot determine it using a building with n floors). If the phone broke after drop from the last floor of some block, the correct floor (lowest floor where the phone breaks) is in this block. Since there are $\mathcal{O}(\sqrt{n})$ blocks, the number of drops in this phase is $\mathcal{O}(\sqrt{n})$.

Then, starting from the first floor of this block we drop the second phone from each floor of this block until the phone breaks, and correctly determine the floor. Since we have at most $\sqrt{n}$ floors in the block, the number of drops in this second phase is $\mathcal{O}(\sqrt{n})$. Hence, the total number of drops is $\mathcal{O}(\sqrt{n})$.

d)* Assume that you are given exactly t identical phones and a building with n floors (where t is some constant that doesn't depend on n). Determine an efficient search strategy for the floor where the phone breaks and give the number of drops in $\mathcal{O}$-Notation.

Solution: We can divide the building into $\mathcal{O}(\sqrt[t]{n})$ blocks (sequences of consecutive floors) of size $\mathcal{O}(n^{1-1/t})$. Starting from the first block we drop the first phone from the last floor of each block until the phone breaks. If the phone didn't break after drop from the n-th floor, the lowest floor where the phone breaks is higher than n (so we cannot determine it using a building with n floors). If the phone broke after drop from the last floor of some block, the correct floor (lowest floor where the phone breaks) is in this block.

Then we can divide this block of size $\mathcal{O}(n^{1-1/t})$ into $\mathcal{O}(\sqrt[t]{n})$ sub-blocks of size $\mathcal{O}(n^{1-2/t})$. Starting from the first sub-block we drop the second phone from the last floor of each sub-block until the phone breaks. If the phone broke after drop from the last floor of some sub-block, the correct floor is in this sub-block.

We repeat this procedure $t-1$ times and in the end we get a sub-block of size $\mathcal{O}(\sqrt[t]{n})$. Then, starting from the first floor of this sub-block we drop the d-th phone from each floor of this sub-block until the phone breaks, and correctly determine the floor. Since we have each block of sub-blocks contains $\mathcal{O}(\sqrt[t]{n})$ subblocks and we have at most t such blocks, the number of drops is $\mathcal{O}(t \cdot \sqrt[t]{n}) = \mathcal{O}(\sqrt[t]{n})$.

Remark. Assume now that t is a function of n. It can be shown that $t \cdot \sqrt[t]{n} \geq \Omega(\log n)$ for all $t \geq 1$. If $t = \Theta(\log n)$, then $\sqrt[t]{n} = \Theta(1)$ and $t \cdot \sqrt[t]{n} = \Theta(\log n)$. So if we take logarithmic number of phones t, the procedure described above becomes very similar to binary search.