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, Ulysse Schaller

19. October 2020

Algorithms & Data Structures — Exercise sheet 5 — HS 20

Exercise Class (Room & TA): ______________________

Submitted by: ______________________

Peer Feedback by: ______________________

Points: _____

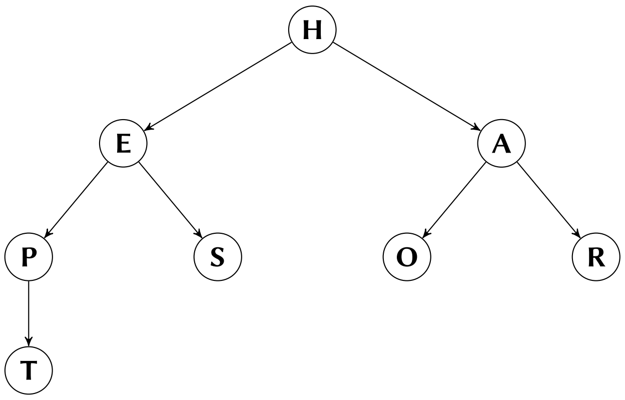
Submission: On Monday, 26 October 2020, hand in your solution to your TA *before* the exercise class starts. Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

Exercise 5.1 *Heapsort* **(1 point)**.

Given the array [H, E, A, P, S, O, R, T], we want to sort it in ascending alphabetical order using Heapsort.

a) Draw the interpretation of the array as a heap, before any call of RestoreHeapCondition.

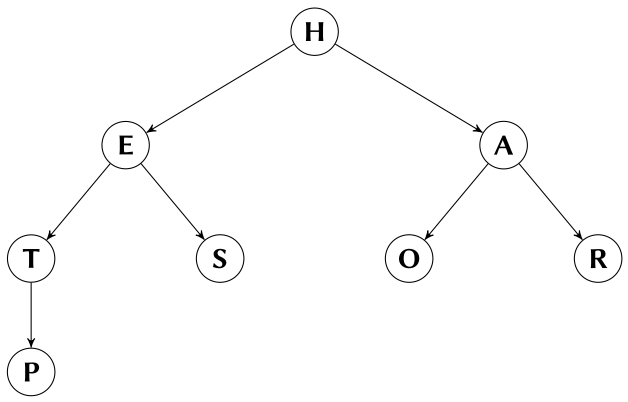
Solution:



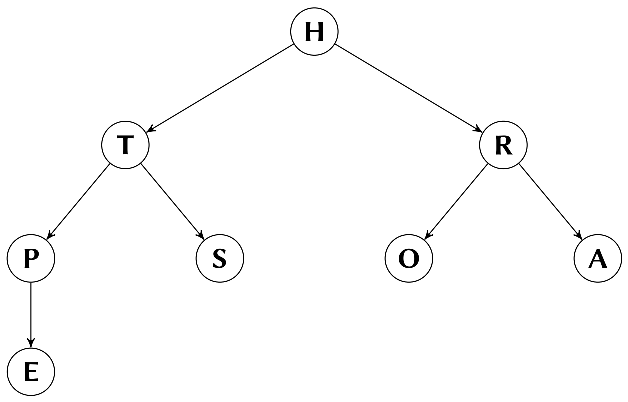
b) In the lecture you have learned a method to construct a heap from an unsorted array (see also pages 35–36 in the script). Draw the resulting max binary heap if this method is applied to the above array.

Solution:

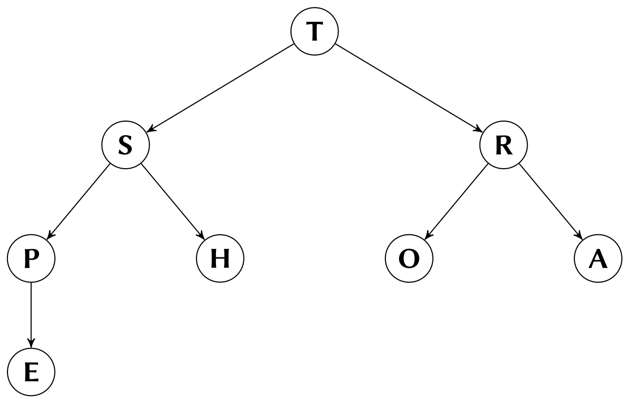
We start from the heap drawn above. The root of the heap is at level 0. Heapifying the subtree with root at level 2 yields:



Then, heapifying the subtrees with roots at level 1 yields:



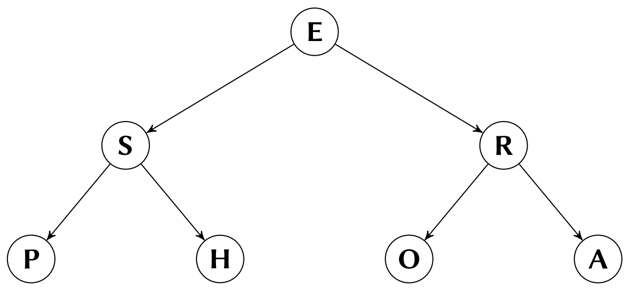
Finally, heapifying the subtree at the root node yields



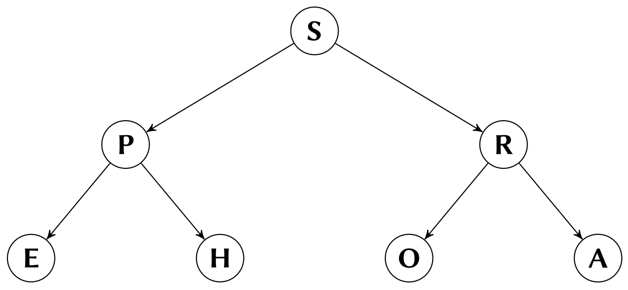
which corresponds to the array [T, S, R, P, H, O, A, E].

c) Sort the above array in ascending alphabetical order with heapsort, beginning with the heap that you obtained in (b). Draw the array after each intermediate step in which a key is moved to its final position.

Solution: We begin with the max binary heap [T, S, R, P, H, O, A, E]. We extract the root T and put it into the last position in the array, i.e., we swap T with the last element E, removing T from the heap, which yields



We then sift E downwards until the heap condition is restored:



Now, the array is [S, P, R, E, H, O, A, T] and contains the one-smaller heap in the front and the sorted entries in the end.

The array after the subsequent steps are as follows. Blue letters are at their final positions.

```
1) Swap S and A:  [A, P, R, E, H, O, S, T]
   Sift A down:   [R, P, O, E, H, A, S, T]

2) Swap R and A:  [A, P, O, E, H, R, S, T]
   Sift A down:   [P, H, O, E, A, R, S, T]

3) Swap P and A:  [A, H, O, E, P, R, S, T]
   Sift A down:   [O, H, A, E, P, R, S, T]

4) Swap O and E:  [E, H, A, O, P, R, S, T]
   Sift E down:   [H, E, A, O, P, R, S, T]

5) Swap H and A:  [A, E, H, O, P, R, S, T]
   Sift A down:   [E, A, H, O, P, R, S, T]

6) Swap E and A:  [A, E, H, O, P, R, S, T]
   done:         [A, E, H, O, P, R, S, T].
```

We are done.

Exercise 5.2 *Sorting algorithms* ***(This exercise is from Summer 2020 exam)*** .

Below you see four sequences of snapshots, each obtained during the execution of one of the following algorithms: `InsertionSort`, `SelectionSort`, `QuickSort`, `MergeSort`, and `BubbleSort`. For each sequence, write down the corresponding algorithm.

8	6	4	2	5	1	3	7
6	4	2	5	1	3	7	8
4	2	5	1	3	6	7	8

Bubblesort

8	6	4	2	5	1	3	7
1	6	4	2	5	8	3	7
1	2	4	6	5	8	3	7

Selectionsort

8	6	4	2	5	1	3	7
6	8	2	4	1	5	3	7
2	4	6	8	1	3	5	7

Mergesort

8	6	4	2	5	1	3	7
6	8	4	2	5	1	3	7
4	6	8	2	5	1	3	7

Insertionsort

Exercise 5.3 *Counting Operations in Loops II.*

For the following code fragments count how many times the function f is called. Report the number of calls as nested sum, and then simplify your expression in simplified Θ-notation and prove your result.

Hint: *Note that in order to justify your Θ-notation you are required to show two parts: an upper bound on your nested sum as well as a lower bound.*

a) Consider the snippet:

Algorithm 1

```
for $j = 1, \dots, n$ do
    $k \leftarrow 1$
    while $k \le j$ do
        $m \leftarrow 1$
        while $m \le j$ do
            $f()$
            $m \leftarrow 2 \cdot m$
        $k \leftarrow 2 \cdot k$
```

Solution: f is called

$$\sum_{j=1}^{n} \sum_{l=0}^{\lfloor \log_2 j \rfloor} \sum_{i=0}^{\lfloor \log_2 j \rfloor} 1 = \sum_{j=1}^{n} (\lfloor \log_2 j \rfloor + 1)^2 \le \sum_{j=1}^{n} (\lfloor \log_2 n \rfloor + 1)^2 \le \mathcal{O}(n \log^2 n)$$

times. Notice that, when $n \ge 4$, then $\log_2(n/2) = \log_2 n - 1 \ge (\log_2 n)/2$. Therefore, for all $n \ge 4$ we have

$$\sum_{j=1}^{n} \sum_{l=0}^{\lfloor \log_2 j \rfloor} \sum_{i=0}^{\lfloor \log_2 j \rfloor} 1 = \sum_{j=1}^{n} (\lfloor \log_2 j \rfloor + 1)^2 \ge \sum_{j=\lceil n/2 \rceil}^{n} (\lfloor \log_2 (n/2) \rfloor + 1)^2$$

$$\ge \sum_{j=\lceil n/2 \rceil}^{n} \log_2 (n/2)^2 \ge n/2 \cdot ((\log_2 n)/2)^2 \ge \Omega(n \log^2 n),$$

so actually we have

$$\sum_{j=1}^{n}\sum_{l=0}^{\lfloor\log_2 j\rfloor}\sum_{i=0}^{\lfloor\log_2 j\rfloor} 1 = \Theta(n\log^2 n).$$

b) Consider the snippet:

Algorithm 2

```
for $j = 1, \dots, n$ do
    for $l = 1, \dots, 100$ do
        $k \leftarrow 1$
        while $k^2 \le j$ do
            $f()$
            $f()$
            $k \leftarrow k + 1$
```

Solution: f is called

$$\sum_{j=1}^{n}\sum_{l=1}^{100}\sum_{k=1}^{\lfloor\sqrt{j}\rfloor} 2 = \sum_{j=1}^{n} 100\cdot\lfloor\sqrt{j}\rfloor\cdot 2 \le 200\sum_{j=1}^{n}\sqrt{n} = 200n^{3/2} \le \mathcal{O}(n^{3/2})$$

times. Notice that, when $n \ge 24$, then $\lfloor\sqrt{n/2}\rfloor \ge \sqrt{n/4}$. Therefore, for all $n \ge 24$ we have

$$\begin{aligned}\sum_{j=1}^{n}\sum_{l=1}^{100}\sum_{k=1}^{\lfloor\sqrt{j}\rfloor} 2 &\ge \sum_{j=\lceil n/2\rceil}^{n}\lfloor\sqrt{j}\rfloor \ge \sum_{j=\lceil n/2\rceil}^{n}\lfloor\sqrt{n/2}\rfloor\\ &\ge \sum_{j=\lceil n/2\rceil}^{n}\sqrt{n/4} \ge n/2\cdot\sqrt{n/4} = n^{3/2}/4 \ge \Omega(n^{3/2}),\end{aligned}$$

so actually we have

$$\sum_{j=1}^{n}\sum_{l=1}^{100}\sum_{k=1}^{\lfloor\sqrt{j}\rfloor} 2 = \Theta(n^{3/2}).$$

Exercise 5.4 *Mastermind* **(1 point)**.

Anna and Ben are playing *Mastermind*. The game consists of pins of 6 different colors taken from the set $C = \{1,2,3,4,5,6\}$. Anna secretly chooses a combination of four of these pins (not necessarily of different colors), i.e. a tuple $a = (a_1,a_2,a_3,a_4) \in C^4$. Ben's goal is to discover this tuple. For every guess $b = (b_1,b_2,b_3,b_4) \in C^4$ that Ben does, Anna tells him how many correct pins there are in b, i.e. for how many indices $i \in \{1,2,3,4\}$ one has $a_i = b_i$. For example, if $a = (2,1,2,1)$ and $b = (5,1,1,1)$, Anna tells Ben that his guess contains 2 correct pins (but she does not tell him which positions are correct).

a) How many different combinations of pins can Anna make ?

Solution: There are $6^4 = 1296$ possible combinations (for each of the 4 positions Anna can choose among 6 different colors).

b) Suppose that, after a few guesses, Ben has reduced to k the number of possible combinations (that is, the number of tuples a that are compatible with all of Anna's answers). Show that it is impossible for Ben to reduce *for sure* the number of possible combinations to strictly less than $\lceil \frac{k-1}{4} \rceil$ with his next guess.

   Hint: *How many possible answers can Anna give to the Ben's next guess? How many combinations correspond to the the answer that the Ben's guess contains* 4 *correct pins?*

   Solution:

   Denote by b the next guess of Ben, and let T be the set of possible combinations just before Ben makes the guess b (so $|T| = k$ by assumption). There are 5 possible answers that Anna can give to b, namely 0, 1, 2, 3 or 4 (corresponding to the number of correct pins in b). So T is partitioned in 5 sets $T_0, \ldots, T_4$, where T_i is the set of combinations in T that are still possible if the answer to Ben's guess is i. Note that Anna answers 4 if and only if the guess b is correct, in other words $T_4 = \{b\}$. In particular $|T_4| = 1$. Since $T_0, \ldots, T_4$ form a partition of T (i.e. they are disjoint and their union is T), this means that the other 4 subsets $T_0, \ldots, T_3$ contain $k-1$ elements together. In particular, one of them must contain at least $\lceil \frac{k-1}{4} \rceil$ elements.

   Let $i \in \{0, 1, 2, 3\}$ be an index such that $|T_i| \geq \lceil \frac{k-1}{4} \rceil$. If Anna's secret combination a is in T_i, then Ben gets the answer i and can therefore only reduce the possibilities to the set T_i. In this case, the number of remaining possibilities is at least $\lceil \frac{k-1}{4} \rceil$.

c) Use parts (a) and (b) to show that, for any strategy that Ben uses to make his guesses, there always exists a tuple $a \in C^4$ that Ben cannot attain in strictly less than 6 guesses. In other words, there is no algorithm that Ben can implement which has a worst-case runtime (in terms of number of guesses) strictly less than 6.

   Solution:

   At the beginning, before Ben makes any guess, there are 1296 possible combinations by part **a**. By part **b**, after one guess, in the worst case there can be $\lceil \frac{1296-1}{4} \rceil = 324$ remaining possibilities, and after the second guess there can remain $\lceil \frac{324-1}{4} \rceil = 81$ of them. Continuing with the same reasoning, we get that after 3 guesses there can be 20 remaining possibilities, and after 4 guesses there can still be 5 possibilities. Then with its fifth guess, Ben can only try one of these 5 combinations, so in the worst case he then needs a sixth guess to output the correct combination.

Exercise 5.5 *Wine tasting* (1 point).

You have n barrels of different wines and you want to organize a wine tasting event. However, you learn that exactly one of the wine barrels is poisoned. Your friend Céline is really into wine and agrees to taste the wines in advance in order to help you find the poisoned barrel. On the i-th day, Céline can try as many different wines as you want. However, the effect of the poisoning is slow, so you only learn on the morning of day $i+1$ whether one of the drunken wines was poisoned: if yes, she is sick for the whole day and cannot drink any wine during this day, but she will be fully recovered for day $i+2$; if not, she can again drink as many wines on day $i+1$ as she wants. In any case, you learn whether one of the wines from the i-th day was poisoned, but you don't know which one it might be. You want to know how many days in the worst case you will need to discover which barrel is poisoned.

a) Draw a decision tree that shows how many different cases (where one case corresponds to one specific barrel being poisoned) you can distinguish after k days, for $k = 4$ (in other words your tree should have depth 4).

Below you can see the decision tree which corresponds to the case $k = 3$:

$$\begin{tikzcd}
\textbf{Day 1} & & & \mathbf{H} \arrow[lld] \arrow[rrd] & & & \\
\textbf{Day 2} & \mathbf{S} \arrow[d] & & & & \mathbf{H} \arrow[ld] \arrow[rd] & \\
\textbf{Day 3} & \mathbf{H} & & & \mathbf{S} & & \mathbf{H}
\end{tikzcd}$$

An **S** node indicates that Céline is sick on that day, i.e. she is not able to drink any wine, while an **H** node indicates that Céline is healthy on that day and can try some wines. The leaves of this tree correspond to 3 different cases which is possible to distinguish in 3 days.

Solution:

Below, a decision tree of the possible outcomes of the first four days of wine tasting is drawn. Note that an **S** node on day i is always followed by an **H** node on day $i + 1$ by assumption, while an **H** node on day i can be followed by either another **H** node or an **S** node on day $i + 1$, depending on whether Céline drank the poisoned wine on this day.

$$\begin{tikzcd}
\textbf{Day 1} & & & & \mathbf{H} \arrow[lld] \arrow[rrd] & & & & \\
\textbf{Day 2} & & \mathbf{S} \arrow[d] & & & & \mathbf{H} \arrow[ld] \arrow[rd] & & \\
\textbf{Day 3} & & \mathbf{H} \arrow[ld] \arrow[rd] & & & \mathbf{S} \arrow[d] & & \mathbf{H} \arrow[ld] \arrow[rd] & \\
\textbf{Day 4} & \mathbf{S} & & \mathbf{H} & & \mathbf{H} & \mathbf{S} & & \mathbf{H}
\end{tikzcd}$$

In particular, there are 5 leaves at the level of the fourth day, which means that one can distinguish 5 different cases within four days.

b) Denote by C_k the number of cases that you can distinguish after k days. Find a recurrence relation of the form $C_k = f(C_{k-1}, C_{k-2})$ and justify why it holds. The decision tree from part (a) could be useful.

Hint: *C_k corresponds to the number of leaves at depth k in the decision tree.*

Solution:

Let $k \geq 3$. Note that C_k is simply the number of leaves in the decision tree of depth k. If we denote by $C_k(\mathbf{H})$ the number of **H** leaves of depth k, and similarly for $C_k(\mathbf{S})$, we see that

$$C_k = C_{k-1}(\mathbf{S}) + 2C_{k-1}(\mathbf{H}),$$

since every **S** node has exactly one children and every **H** node has exactly two children. Using the fact that $C_{k-1} = C_{k-1}(\mathbf{S}) + C_{k-1}(\mathbf{H})$, we get $C_k = C_{k-1} + C_{k-1}(\mathbf{H})$. Since every node on level $k - 2$ has exactly one **H** children on level $k - 1$, we actually have $C_{k-1}(\mathbf{H}) = C_{k-2}$, so we deduce the recurrence relation

$$C_k = C_{k-1} + C_{k-2}. \tag{1}$$

Alternatively, one can argue that a decision tree with root **H** and depth at least 2 has one subtree in which the next **H** node is one level below the root, and one subtree in which the next **H** node is two levels below the root. This argument can also be given in the form that in case of staying healthy, Céline still has $k-1$ days and can thus distinguish between C_{k-1} wines, and in case of getting sick, she has $k-2$ days left, and can thus distinguish between C_{k-2} wines.

c) Use the recurrence relation found in part (b) to show by induction that $C_k \leq 2^k$.

Solution:

We have $C_1 = 1$, $C_2 = 2$, and (1) holds for all $k \geq 3$.

Base case. We have $C_1 = 1 \leq 2^1$ and $C_2 = 2 \leq 2^2$.

Induction Hypothesis. We assume that for some $k \geq 2$ it holds that

$$C_k \leq 2^k \quad \text{and} \quad C_{k-1} \leq 2^{k-1}. \tag{2}$$

Inductive step ($k \to k+1$). Let $k \geq 2$. By (1), we have $C_{k+1} = C_k + C_{k-1}$. Using the induction hypothesis (2), we get

$$C_{k+1} \leq 2^k + 2^{k-1} \leq 2^k + 2^k = 2^{k+1}$$

as desired.

*c') Alternatively to (c), let $\varphi := \frac{1+\sqrt{5}}{2}$, and show that $C_k \leq \varphi^k$.

Solution:

First of all, note that $\varphi < 2$ and thus this is a stricly stronger statement than (c). We have $C_1 = 1$, $C_2 = 2$, and (1) holds for all $k \geq 3$. We will now show by indution that $C_k \leq \varphi^k$.

Base case. We have $C_1 = 1 \leq \frac{1+\sqrt{5}}{2}$ and $C_2 = 2 \leq \left(\frac{1+\sqrt{5}}{2}\right)^2$.

Induction Hypothesis. We assume that for some $k \geq 2$ it holds that

$$C_k \leq \varphi^k \quad \text{and} \quad C_{k-1} \leq \varphi^{k-1}. \tag{3}$$

Inductive step ($k \to k+1$). Let $k \geq 2$. By (1), we have $C_{k+1} = C_k + C_{k-1}$. Using the induction hypothesis (3), we get

$$C_{k+1} \leq \varphi^k + \varphi^{k-1} = \varphi^k(1+\varphi^{-1}) = \varphi^k \frac{3+\sqrt{5}}{\sqrt{5}+1} = \varphi^k \frac{2\sqrt{5}+2}{6} \leq \varphi^k \frac{\sqrt{5}+1}{2} = \varphi^{k+1}$$

as desired.

d) Deduce that, no matter what the drinking strategy of Céline is, you will need at least $\Omega(\log n)$ days to find the poisoned wine in the worst case.

Solution:

Part (c) tells us that after k days, one can distinguish at most 2^k different cases. If you have n different barrels, you need to make sure that you can distinguish at least n different cases (the poisoned wine could be in any of the n barrels). In particular, the number k of days needed must satisfy $2^k \geq n$. This is equivalent to

$$k \geq \log_2(n) \geq \Omega(\log n),$$

which concludes the proof.

e) Is there a strategy that is guaranteed to succeed in $\mathcal{O}(\log n)$ days? Justify your answer.

Solution:

The naive binary search approach actually always succed in $\mathcal{O}(\log n)$ days. Indeed, notice that we can divide the number of potentially poisoned barrels by two every two days. More precisely, if on day i we know that the poisoned barrel is among m given barrels, by making Céline drink a subset of $\lfloor m/2 \rfloor$ of these wines, we will have reduced the number of potentially poisoned wines to $\lceil m/2 \rceil$ by day $i + 1$, and we can continue this procedure on day $i + 2$ (in case Céline was sick on day $i + 1$). The number of days required for this strategy is simply the double of the number of binary search steps that needs to be done. Since binary search among n elements requires $\mathcal{O}(\log n)$ steps, the number of days needed is also $\mathcal{O}(\log n)$.