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

11. November 2019

Algorithms & Data Structures — Exercise sheet 8 — HS 19

Exercise Class (Room & TA): _____

Submitted by: _____

Peer Feedback by: _____

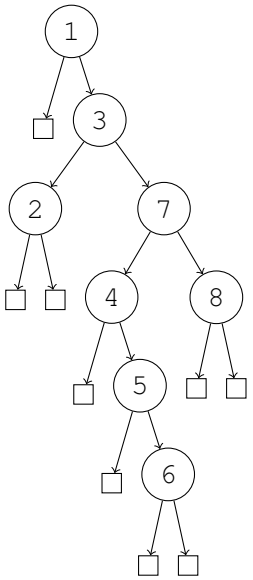
Points: ____

Submission: On Monday, 18 November 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 8.1 *Search Trees.*

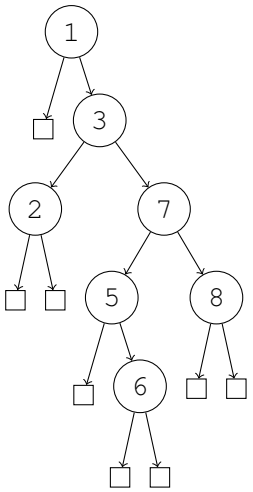
a) Draw the resulting tree when the keys 1,3,7,4,5,8,6,2 in this order are inserted into an initially empty binary (natural) search tree.

Solution:

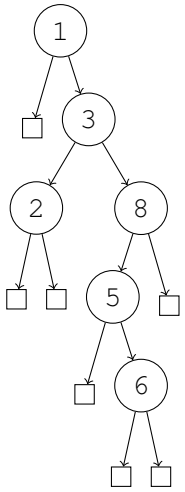


b) Delete key 4 in the above tree, and afterwards key 7 in the resulting tree.

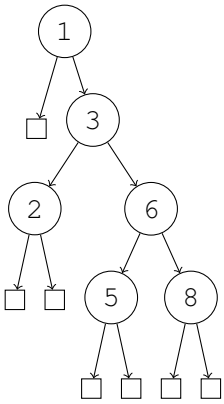
Solution: Key 4 has one child, so it can just be replaced by 5:



Key 7 must either be replaced by its sucessor key, 8, or its precesessor key, 6. If key 7 is replaced by its sucessor:



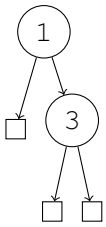
If key 7 is instead replaced by its predecessor:



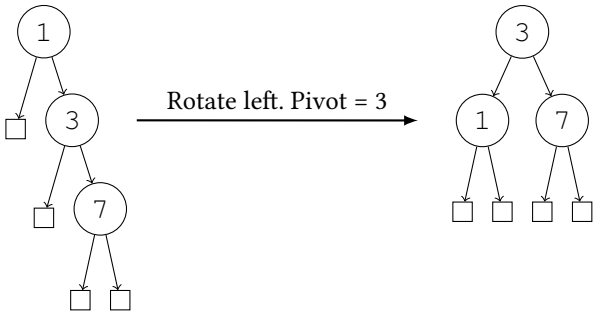
c) Draw the resulting tree when the keys are inserted into an initially empty AVL tree. Give also the intermediate states before and after each rotation that is performed during the process.

Solution:

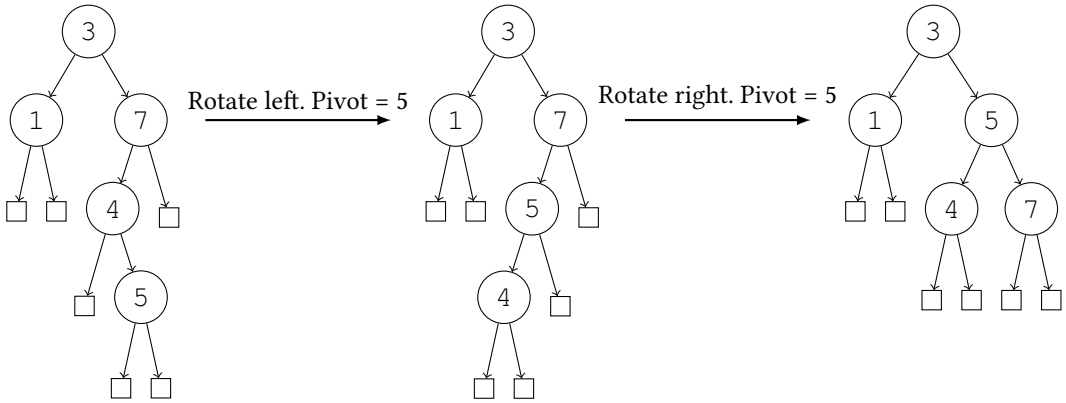
Insert 1 and then 3:



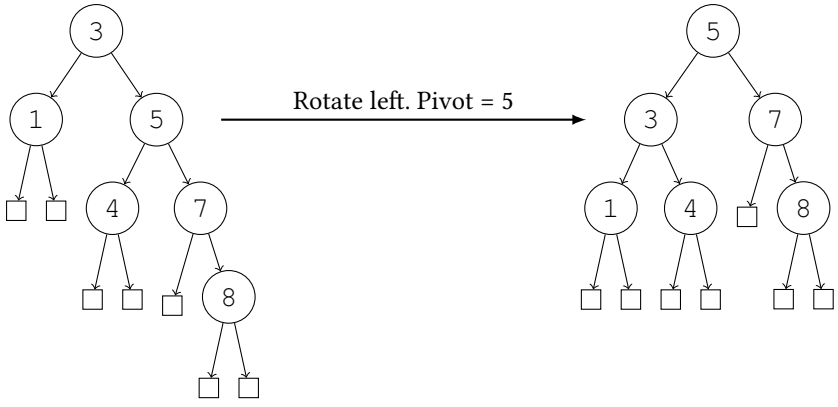
Insert 7:



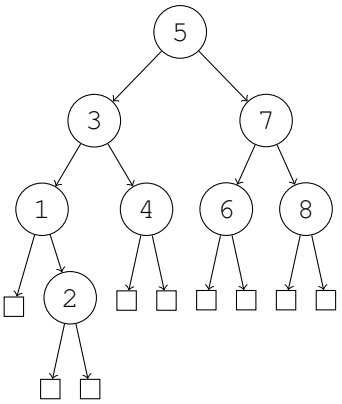
Insert 4 and then 5:



Insert 8:



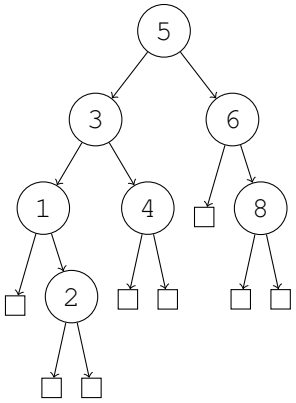
Insert 6 and 2:



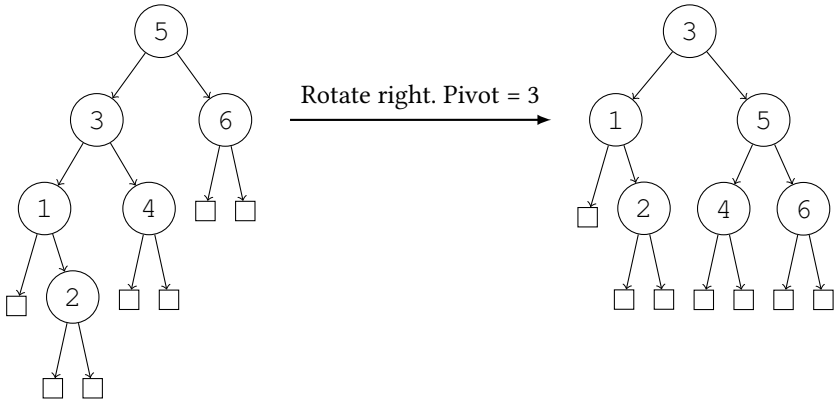
d) Delete key 7 in the above tree, and afterwards key 8 in the resulting tree.

Solution:

Delete 7:



Delete 8:



*__Exercise 8.2__ *Maximum Depth Difference of two Leaves.*

Consider an AVL tree of height h. What is the maximum possible difference of the depths of two leaves? Describe which structure such trees need to have, and draw examples of corresponding trees for every $h \in \{2, 3, 4\}$. Derive a recursive formula (depending on h), solve it and use induction to prove the correctness of your solution. Provide a detailed explanation of your considerations.

Hint: *For the proof the principle of complete induction can be used. Let $\mathcal{A}(n)$ be a statement for a number $n \in \mathbb{N}$. If, for every $n \in \mathbb{N}$, the validity of all statements $\mathcal{A}(m)$ for $m \in \{1, \ldots, n-1\}$ implies the*

validity of $\mathcal{A}(n)$, then $\mathcal{A}(n)$ is true for every $n \in \mathbb{N}$. Thus, complete induction allows multiple base cases and inductive hypotheses.

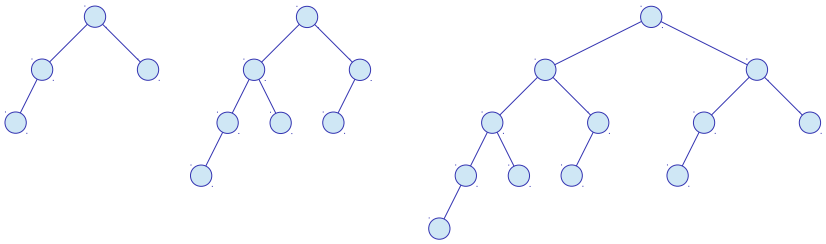
Solution:

For an AVL-tree T with a root node v and height h, we can distinguish the following 3 cases:

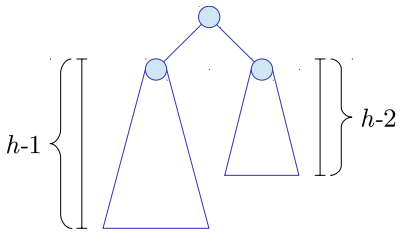
- Both sub-trees $T_l(v)$ and $T_r(v)$ have height $h-1$
- $T_l(v)$ has height $h-1$ and $T_r(v)$ has height $h-2$, or
- $T_l(v)$ has height $h-2$ and $T_r(v)$ has height $h-1$

As we are interested in the *maximum* depth difference of two leaves, we can disregard the first case, and focus on sub-trees that have heights that differ by 1. Without loss of generality, we can take the second case, assuming that the left sub-tree will have height of $h-1$, while the right sub-tree will have height of $h-2$. If the left sub-tree is an AVL-tree of height $h-1$, then the right tree must be an AVL-tree of height $h-2$. This comes from the properties of an AVL tree, because if at any time they differ by more than one, rebalancing is done to restore this property. As a result of this, the entire tree T will have a height of h and as such there will be a leaf on the left-subtree with this depth.

The figure below illustrates the AVL trees of height $h \in \{1, 2, 3\}$:



In general, we consider trees with the following structure:



The left subtree $T_l(v)$ contains a leaf of depth h (while $T_l(v)$ has height of $h-1$), the right subtree $T_r(v)$ contains a leaf of depth $h-1$ (while $T_r(v)$ has height $h-2$). The maximum possible difference of the depths of two leaves in the tree (with height h) is therefore 1 greater than the maximum difference of the depths of two leaves in the right subtree (with height $h-2$). For $h=2$ and $h=3$, the maximum depth difference is exactly 1.

As a result, we have the following recursive formula for the maximum difference of the depths of two leaves in a tree of height h:

$$D(2) = 1,\ D(3) = 1,\ D(h) = 1 + D(h-2) \text{ for all } h \geq 4. \tag{1}$$

From the above, we can assume that $D(h) = \lfloor h/2 \rfloor$. We prove this assumption using induction over h.

Base case I ($h = 2$): $D(2) = 1 = \lfloor 2/2 \rfloor$.

Base case II ($h = 3$): $D(3) = 1 = \lfloor 3/2 \rfloor$.

Induction hypothesis: Assume that the property holds for some h: $D(h-2) = \lfloor (h-2)/2 \rfloor$.

Inductive step: $((h-2) \to h)$: From the recursive definition of $D(h)$, we have:

$$D(h) = 1 + D(h-2) = 1 + \lfloor (h-2)/2 \rfloor = 1 + \lfloor h/2 - 1 \rfloor = 1 + \lfloor h/2 \rfloor - 1 = \lfloor h/2 \rfloor. \qquad (2)$$

***Exercise 8.3** *Fibonacci numbers.*

The *Fibonacci numbers* are defined by $F(0) := 0$, $F(1) := 1$, and $F(n) := F(n-1) + F(n-2)$ for $n \geq 2$. Prove that for all $n \geq 0$,

$$F(n) = \frac{1}{\sqrt{5}} \left(\phi^n - \psi^n \right),$$

where $\phi = (1 + \sqrt{5})/2$ and $\psi = (1 - \sqrt{5})/2$. Show that this formula implies $F(n) = \Theta(\phi^n)$.

Hint: *You can solve the problem inductively by direct (and nasty) calculation. If you want to understand where the formula comes from, you should proceed by the following steps.*

1. *Show that ϕ and ψ are the roots of the equation $x^2 = x + 1$.*
2. *Show that if a real number x satisfies $x^2 = x + 1$, then for all $n \geq 2$ it satisfies $x^n = x^{n-1} + x^{n-2}$.* Hint: This is trivial!
3. *Consider the set $V := \{f : \mathbb{N}_0 \to \mathbb{R} \mid f(n) = f(n-1) + f(n-2)$ for all $n \geq 2\}$. Show that V forms a vector space, and that the two functions $G(n) := \phi^n$ and $H(n) := \psi^n$ are in V. In fact, it is easy to see (you don't need to formally prove this, but give an intuitive argument) that V has dimension 2, and that G and H are independent. So they form a basis.*
4. *Conclude that there must be two constants $a, b \in \mathbb{R}$ such that $F = a \cdot G + b \cdot H$.*
5. *Compute a and b by looking at the special cases $F(0) = a \cdot G(0) + b \cdot H(0)$ and $F(1) = a \cdot G(1) + b \cdot H(1)$.*
6. *Conclude that $F_n = \frac{1}{\sqrt{5}}(\phi^n - \psi^n)$ for all $n \geq 0$.*

Solution:

We solve the problem as in the hint.

1. By the p-q-formula that you learned at school, the solutions of $x^2 + px + q = 0$ are $-\frac{p}{2} \pm \frac{1}{2}\sqrt{p^2 - 4q}$. Here we have $p = q = -1$.
2. We just multiply both sides of $x^2 = x + 1$ with x^{n-2}.
3. Let $f, g \in V$ and $\alpha \in \mathbb{R}$. Then $f + g \in V$ because $(f+g)(n) = f(n) + g(n) \overset{f,g \in V}{=} f(n-1) + f(n-2) + g(n-1) + g(n-2) = (f+g)(n-1) + (f+g)(n-2)$.

   Likewise, $\alpha f \in V$ because $\alpha \cdot f(n) \overset{f \in V}{=} \alpha \cdot (f(n-1) + f(n-2)) = \alpha \cdot f(n-1) + \alpha \cdot f(n-2)$.

   Therefore, V is a vector space. It has dimension 2 because (being slightly sloppy) a function in V is determined by the first 2 values.[1] G and H are in V by items 1 and 2, and they are independent

[1] A rigorous mathematical argument would be to consider the mapping $P : V \to \mathbb{R}^2; f \mapsto (f(0), f(1))$. It is straightforward to check that this is a homomorphism. Moreover, it is injective (any two functions in V which coincide on 0 and 1 agree everywhere, since they satisfy the same recurrence relation) and surjective (for every $x, y \in \mathbb{R}$ we can construct a function in V with $f(0) = x$ and $f(1) = y$). Hence, P is an isomorphism, and thus V has the same dimension as $\mathbb{R}^2$.

because they are not multiples from each other. This can be argued in many ways, for example: if they were multiples of each other, then $G = \Theta(H)$ would have to hold. However, this does not hold since $G(n) \to \infty$ and $H(n) \to 0$ (since $|\psi| < 1$) for $n \to \infty$.

4. This follows since any two independent elements of a vector space of dimension 2 form a basis.

5. By definition of F, G, H, we have $F(0) = 0$, $F(1) = 1$, $G(0) = 1$, $G(1) = \phi$, $H(0) = 1$, $H(1) = \psi$. Hence, a and b must satisfy

$$\begin{aligned} a + b &= 0, \\ a \cdot \phi + b \cdot \psi &= 1. \end{aligned}$$

   This is a linear system of equation, and you have learned in linear algebra how to solve them systematically. Here, we can shortcut by rewriting the first equation as $b = -a$, and plugging this into the second equation. This gives $a \cdot (\phi - \psi) = 1$, or $a = 1/(\phi - \psi) = 1/\sqrt{5}$, and thus $b = -a = -1/\sqrt{5}$.

6. This is just a summary of what we have shown: we have $F = a \cdot G + b \cdot H$, which is in other words that for all $n \geq 0$ we have

$$F(n) = a \cdot G(n) + b \cdot H(n) = \frac{1}{\sqrt{5}} \left(\phi^n - \psi^n\right).$$

Finally, since $H(n) \to 0$ and $G(n) \to \infty$ for $n \to \infty$, we have $F = \Theta(G) = \Theta(\phi^n)$.

Exercise 8.4 *Online supermarket* **(1 point)**.

Assume that you work in a large online supermarket that offers different types of goods. At every moment you have to know the number of goods of each type that the supermarket currently offers. Let us denote the number of goods of type t by S_t. At any moment S_t can either be decreased (if someone has bought some goods of type t) or increased (if some goods of type t have been delivered from the manufacturer). Also your boss can ask you how many goods of type t does the supermarket currently offer. So you can receive three types of queries: to decrease S_t by $0 < x \leq S_t$, to increase S_t by $x > 0$ or to return S_t.

Assume that at each moment number of different types of goods that the supermarket offers at that moment is bounded by $n > 0$, but the number of types of goods that the supermarket can potentially offer might be much larger than n. Consider the following example: $n = 3$, at 14:00 the supermarket can offer 5 balls, 1 doll and 4 phones and at 14:15 it can offer 6 balls, 3 chairs and 12 pencils.

Provide an algorithm that can handle each query in time $\mathcal{O}(\log n)$. You may assume that initially all S_t are zero.

Solution:

We store the goods in an AVL-tree. That is, for good t we use t as the key that determines the position in the AVL-tree, and we store the value S_t in the node of t. We only store a good t in the AVL-tree if $S_t > 0$.

For all three types of queries, we first search the key t in the AVL-tree, which takes time $O(\log n)$. If t does not exist in the tree, then we give out 0 if we are asked for the number of elements; we give an error for a decrease query; and we insert the key t into the AVL-tree with value $S_t := x$ if we get an increase query. The latter operation takes time $O(\log n)$, the other two take constant time. However, since we first need to search for the key, all queries need total time $O(\log n)$.

If the key t exists, we proceed similarly. Depending on the query, we return S_t, or in-/decrease S_t by x. Moreover, if we decrease S_t to zero then we delete the key from the AVL-tree, which takes time $O(\log n)$. Again, all queries take a total time of $O(\log n)$, as required.

Exercise 8.5 *Nim game* **(2 points)**.

Consider the following game in which two players move alternately: at the beginning there are 2 piles of stones with n_1 and n_2 stones, respectively. During a move a player chooses a non-empty pile and an integer $k > 0$, and removes $s = k^2$ stones from the chose pile. (Obviously, s cannot be greater than the number of stones in the chosen pile). Player 1 makes the first move. A player who cannot move loses. For example, if all piles are empty at the very beginning, Player 1 loses.

a) Provide a *dynamic programming* algorithm that, given n_1 and n_2, determines which player wins the game if both play optimally. You can assume that arithmetic operations take unit time. You can obtain full points with a dynamic programming with a DP table of dimension 2.

   Address the following aspects in your solution:

   (a) *Dimension of the DP table: What is the dimension of the table $DP[\ldots]$*? What range do you have in each dimension?

   (b) *Definition of the DP table*: What is the meaning of each entry?

   (c) *Computation of an entry*: How can an entry be computed from the values of other entries? Specify the base cases, i.e., the entries that do not depend on others.

   (d) *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?

   (e) *Extracting the solution*: How can the final solution be extracted once the table has been filled?

   (f) *Running time*: What is the runtime of your solution?[2] Does the algorithm have polynomial runtime?

   Solution:

   Dimensions of the table:

   The table is 2-dimensional, the first dimension ranges from 0 to n_1, and the second dimension ranges from 0 to n_2.

   Meaning of a table entry (in words): For $0 \leq i \leq n_1$ and $0 \leq j \leq n_2$, consider the position in which the first pile has i stones, the second pile has j stones, and player α moves next (where α is 1 or 2). The entry $DP[i, j]$ is 1 if player α has a winning strategy for this position, and 0 otherwise.

   Computation of an entry (initialization and recursion): We initialize the DP-table by $DP[0, 0] = 0$.

   For all other positions (i, j), there are still valid moves. Assume without loss of generality that player 1 moves at some position (i, j), where $i > 0$ or $j > 0$. Player 1 wins if and only if she can make a move that leads to a *losing position* for player 2. Therefore, $DP[i, j] = 1$ if and only if there exists a k such that

[2] By this formulation, we expect you to specify the worst case runtime in Θ-notation.

- $k^2 \leq i$ and $DP[i-k^2, j] = 0$, or
- $k^2 \leq j$ and $DP[i, j-k^2] = 0$.

We could write this concisely, e.g., by the formula

$$DP[i,j] = 1 - \min\left(\min_{1\leq k\leq\sqrt{i}}(DP[i-k^2,j]), \min_{1\leq k\leq\sqrt{j}}(DP[i,j-k^2])\right).$$

However, this formula is harder to understand than the other description, so the other description is preferable.

Order of computation:

There are several ways for the order of computation. One possibility is to first compute all values $DP[i,0]$ in increasing order of i, then all values $DP[i,1]$ in increasing order of i, and so on. Another way would be to compute the fields by increasing order in $i+j$, i.e, first compute the field with $i+j=0$, then all fields with $i+j=1$ (in arbitrary order), then all fields with $i+j=2$ (in arbitrary order), and so on. This works since in order to compute $DP[i,j]$, we only need to look up entries for which $i+j$ is strictly smaller.

Computing the output:

The output is in the field $DP[n_1, n_2]$.

Running time in $\mathcal{O}$-notation in terms of n_1 and n_2:

The table has size $n_1 \cdot n_2$. To compute the entry $DP[i,j]$, we need to take a minimum over $\Theta(\sqrt{i}+\sqrt{j})$ values, which needs time $\Theta(\sqrt{i}+\sqrt{j})$. (Except for $i=j=0$, but this will only contribute $O(1)$ and is negligible.) Therefore the total runtime is

$$\Theta(\underbrace{\sum_{i=0}^{n_1}\sum_{j=0}^{n_2}\sqrt{i}+\sqrt{j}}_{=:T}).$$

To estimate T, we first compute an upper bound by observing that $\sqrt{i} \leq \sqrt{n_1}$ and $\sqrt{j} \leq \sqrt{n_2}$. Hence,

$$T \leq \sum_{i=0}^{n_1}\sum_{j=0}^{n_2}(\sqrt{n_1}+\sqrt{n_2}) = (\sqrt{n_1}+\sqrt{n_2}) \cdot \sum_{i=0}^{n_1}\sum_{j=0}^{n_2} 1 \leq O((\sqrt{n_1}+\sqrt{n_2}) \cdot n_1 \cdot n_2)$$

For the lower bound, we make the sum smaller by restricting to the range $n_1/2 \leq i \leq \sqrt{n_1}$ and $n_2/2 \leq j \leq \sqrt{n_2}$. In this range, we have $\sqrt{i} \geq \sqrt{n_1/2}$ and $\sqrt{j} \geq \sqrt{n_2/2}$. Hence,

$$T \geq \sum_{i=n_1/2}^{n_1}\sum_{j=n_2/2}^{n_2}\sqrt{\frac{n_1}{2}}+\sqrt{\frac{n_2}{2}} = \frac{1}{\sqrt{2}}(\sqrt{n_1}+\sqrt{n_2}) \cdot \underbrace{\sum_{i=n_1/2}^{n_1}\sum_{j=n_2/2}^{n_2} 1}_{=n_1\cdot n_2/4 \text{ up to rounding}} \geq \Omega\left((\sqrt{n_1}+\sqrt{n_2}) \cdot n_1 \cdot n_2\right).$$

Since the upper and lower bound match, we obtain $T = \Theta((\sqrt{n_1}+\sqrt{n_2}) \cdot n_1 \cdot n_2)$. We remark that this can be equivalently written as $T = \Theta(\max(\sqrt{n_1}, \sqrt{n_2}) \cdot n_1 \cdot n_2)$

This time is not polynomial in the input size, since the input size is only $\log_2 n_1 + \log_2 n_2$. However, it is polynomial in the quantities n_1 and n_2, and thus the algorithm has *pseudopolynomial* runtime.

b) Finally, assume you have already filled out the DP table, and that the current player (say, it is Player 1) is in a winning position. Describe how you can find a winning strategy, i.e., how you can determine a move such that after her turn, Player 2 is in a *losing* position.

Solution: Once the DP table is filled and we are in a winning position (i,j), we obtain a winning move by tracing back the "reason" for the one in $DP[i, j]$. I.e, we find a k such that $DP[i - k^2, j] = 0$ or $DP[i, j - k^2] = 0$ and this defines us the winning move. This needs time $O(\sqrt{i} + \sqrt{j})$, i.e., linear time in the number of moves in that position. If we invest slightly more work when we fill out the table, we can decrease this time: when we fill the table, we compute this move anyway. So when we fill out the table, then for each cell with a one-entry we can also store the winning move in this cell with additional time $O(1)$ per cell. This allows us to trace back the winning strategy faster (in time $O(1)$ for each move, since we only need to look up the stored entry), but it does not decrease the overall asymptotic runtime when we consider all parts of the algorithm (creation of the table and tracing back) together.