Exercise Session 06 – Trees

Data Structures and Algorithms

These slides are based on those of the lecture, but were adapted and extended by the teaching assistant Adel Gavranović

Today's Schedule

- Intro
- Follow-up
- Feedback for **code** expert
- Learning Objectives
- Repetition theory
  - Binary Trees and Heaps
  - Binary Trees
  - 2-3 Trees
  - Red-Black Trees
- Code-Example
- Old Exam Question
- Outro

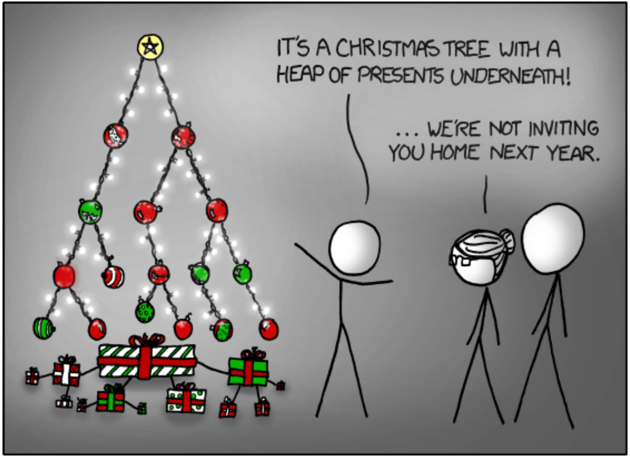
`n.ethz.ch/~agavranovic`

▸ Exercise Session Material

▸ Adel's Webpage

▸ Mail to Adel

Comic of the Week



▸ xkcd

1. Intro

Intro

Intro

- Lots of exercises today so get your tablets and styli ready!

2. Follow-up

Follow-up from last exercise session

Follow-up from last exercise session

- None(?)

3. Feedback for **code** expert

General things regarding **code** expert

General things regarding **code** expert

- I’m working on the corrections with highest priority

General things regarding **code** expert

- I’m working on the corrections with highest priority
  - If there’s anyone still waiting for the **code** expert text task corrections for unlocking the Bonus Exercise: **Send me an email ASAP**

General things regarding **code** expert

- I’m working on the corrections with highest priority
  - If there’s anyone still waiting for the **code** expert text task corrections for unlocking the Bonus Exercise: **Send me an email ASAP**
- Please send me questions before Thursday so I can prepare for them properly and give more useful answers during/after the session (they also might be relevant for others!)

General things regarding **code** expert

- I'm working on the corrections with highest priority
  - If there's anyone still waiting for the **code** expert text task corrections for unlocking the Bonus Exercise: **Send me an email ASAP**
- Please send me questions before Thursday so I can prepare for them properly and give more useful answers during/after the session (they also might be relevant for others!)
- Keep your **code** expert answers brief

General things regarding **code** expert

- I’m working on the corrections with highest priority
  - If there’s anyone still waiting for the **code** expert text task corrections for unlocking the Bonus Exercise: **Send me an email ASAP**
- Please send me questions before Thursday so I can prepare for them properly and give more useful answers during/after the session (they also might be relevant for others!)
- Keep your **code** expert answers brief
- You can answer in german too if that is easier for you

Exercise Review: "The Master Method"

There was an error in the task description!

Exercise Review: "The Master Method"

There was an error in the task description!

- Written:
  - $a \geq 1$ *and* $b > 1$ *are* ***integer*** *constants*

Exercise Review: "The Master Method"

There was an error in the task description!

- Written:
  - $a \geq 1$ *and* $b > 1$ *are* ***integer*** *constants*
- But should be:
  - $a \geq 1$ *and* $b > 1$ *are real constants*
  - i.e. a, b don't have to be integers

$\frac{2n}{3}$ $\frac{n}{b}$

$b = \cancel{\frac{2}{3} < 1}$ ↯

$= \frac{3}{2}$

Exercise Review: "Comparing Sorting Algorithms"

Bubblesort	min	max
Comparisons	$\Theta(n^2)$	$\Theta(n^2)$
Sequence	any	any
Swaps	0	$\Theta(n^2)$
Sequence	$1, 2, \ldots, n$	$n, n-1, \ldots, 1$

Exercise Review: "Comparing Sorting Algorithms"

InsertionSort	min	max
Comparisons	$\Theta(n)$	$\Theta(n^2)$
Sequence	$1, 2, \ldots, n$	$n, n-1, \ldots, 1$
Swaps	0	$\Theta(n^2)$
Sequence	$1, 2, \ldots, n$	$n, n-1, \ldots, 1$

Exercise Review: "Comparing Sorting Algorithms"

SelectionSort	min	max
Comparisons	$\Theta(n^2)$	$\Theta(n^2)$
Sequence	any	any
Swaps	0	$\Theta(n)$
Sequence	$1, 2, \ldots, n$	$n, n-1, \ldots, 1$

Exercise Review: "Comparing Sorting Algorithms"

QuickSort	min	max
Comparisons	$\Theta(n \log n)$	rare $\Theta(n^2)$
Sequence	complex	$1, 2, \ldots, n$
Swaps	$\Theta(n)$	$\Theta(n \log n)$
Sequence	$1, 2, \ldots, n$	complex

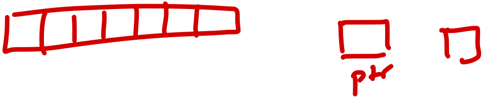
complex: Sequence must be made such that the pivot halves the sorting range in each step. For example ($n = 7$): $4, 5, 7, 6, 2, 1, 3$

"Require a constant amount of (additional) memory"

---

[1] i.e. independent of the size n of the data

"Require a constant amount of (additional) memory"



Basically, whenever an algorithm's memory footprint is only a **constant**[1] amount more than the data size n.

---

[1] i.e. independent of the size n of the data

"Require a constant amount of (additional) memory"

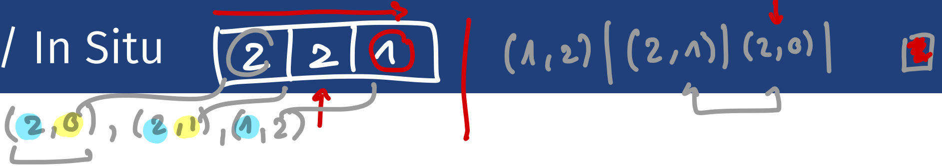
Basically, whenever an algorithm's memory footprint is only a **constant**[1] amount more than the data size n.

e.g. only storing a "highest so far"-variable (in addition to the data size n) would entail a memory cost of 1 (constant)

---

[1] i.e. independent of the size n of the data

Stable / In Situ



In-Situ

- QuickSort uses between $\Omega(\log n)$ and $\mathcal{O}(n)$ extra space to keep track of the recursive calls.
- MergeSort has to merge repeatedly parts of the array. There are complicated modifications to make MergeSort in-situ, but none that can be achieved by simple modifications of the standard algorithm.

Stable

- Stability of a sorting algorithm only refers to the order of elements with the same value. Attribute each element with its original position and sort by value plus position for elements with equal values. Maximally one additional comparison per element (factor of 2), hence the asymptotic running time stays the same.

Questions regarding **code** expert from your side?

Questions regarding **code** expert from your side?

- "How to write answers in **code** expert that are legible"?

Questions regarding **code** expert from your side?

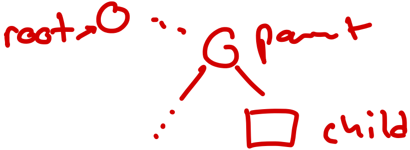
- "How to write answers in **code** expert that are legible"?
- "How to upload PDF solutions to **code** expert correctly"?

4. Learning Objectives

Learning Objectives

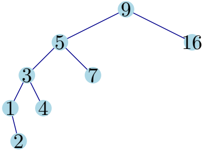
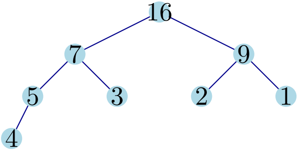
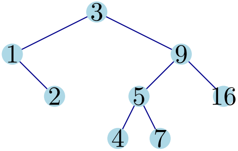
☐ Be able to perform basic operations on the most common trees

5. Repetition theory

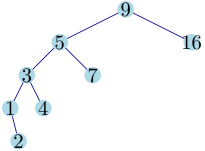
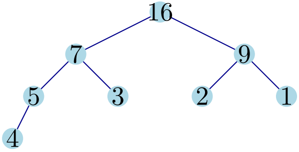
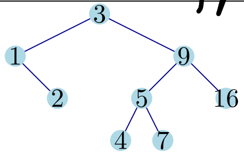


5.1 Binary Trees and Heaps

Comparison of binary Trees

	Search trees	**Heaps** Min- / Max- Heap	**Balanced trees** AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
			
Insertion	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Search	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Deletion	$\Theta(h(T))$	Search + $\Theta(\log n)$	$\Theta(\log n)$
Min/Max	$\Theta(h(T))$	$\Theta(1)$/search	$\Theta(\log n)$

Comparison of binary Trees

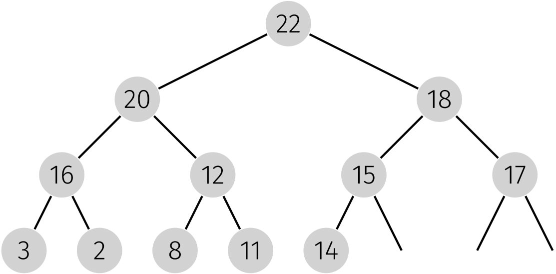
	Search trees	**Heaps** Min- / Max- Heap	**Balanced trees** AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
operations			
Insertion	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Search	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Deletion	$\Theta(h(T))$	Search + $\Theta(\log n)$	$\Theta(\log n)$
Min/Max	$\Theta(h(T))$	$\Theta(1)$/search	$\Theta(\log n)$

cppreference

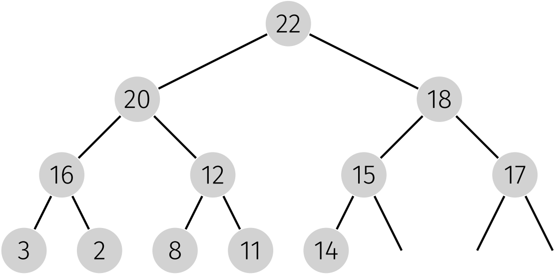
hard way

Remark: $\Theta(\log n) \leq \Theta(h(T)) \leq \Theta(n)$

Recall: Binary Tree as Array

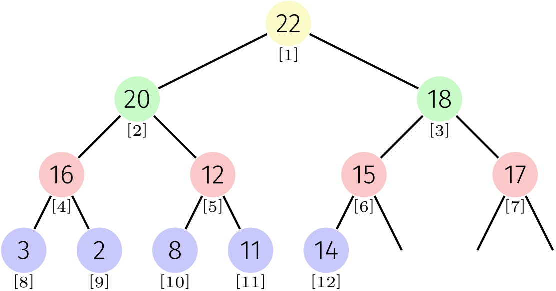


Recall: Binary Tree as Array



22	20	18	16	12	15	17	3	2	8	11	14
1	2	3	4	5	6	7	8	9	10	11	12

Recall: Binary Tree as Array



22	20	18	16	12	15	17	3	2	8	11	14
1	2	3	4	5	6	7	8	9	10	11	12

Repetition: Binary Trees, Inserting a Key

Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).

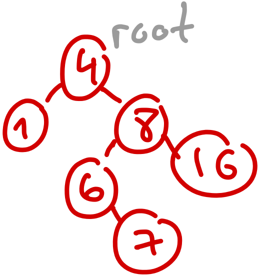
MinHeap

- Insert at the very next free spot (back of the array).
- Restore Heap-Condition: `siftUp` (climb successively).

Repetition: Binary Trees, Inserting a Key

Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).



MinHeap " min on top "

- Insert at the very next free spot (back of the array).
- Restore Heap-Condition: `siftUp` (climb successively).

Exercise: Insert $4, 8, 16, 1, 6, 7$ into empty Search Tree/Min-Heap.

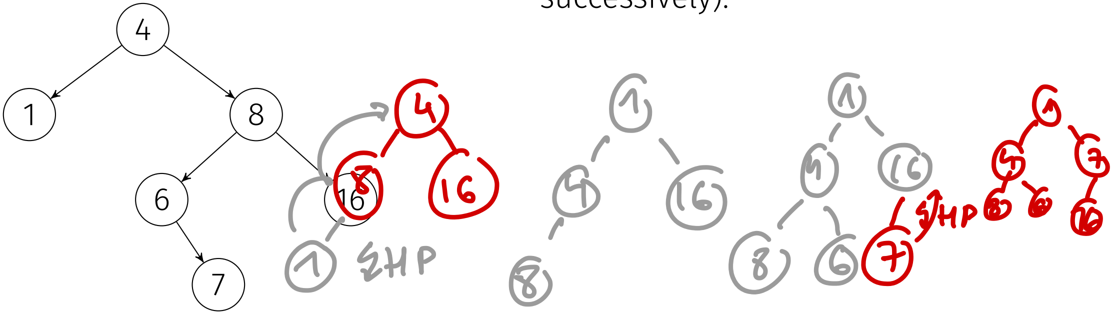
Repetition: Binary Trees, Inserting a Key

Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).

MinHeap

- Insert at the very next free spot (back of the array).
- Restore Heap-Condition: `siftUp` (climb successively).

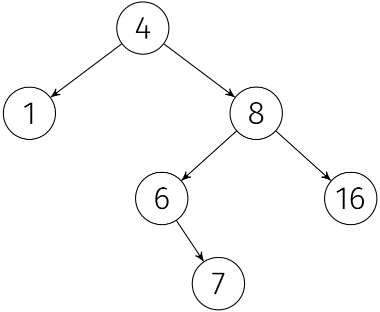


Exercise: Insert ~~4~~, ~~8~~, ~~16~~, ~~1~~, ~~6~~, 7 into empty Search Tree/Min-Heap.

Repetition: Binary Trees, Inserting a Key

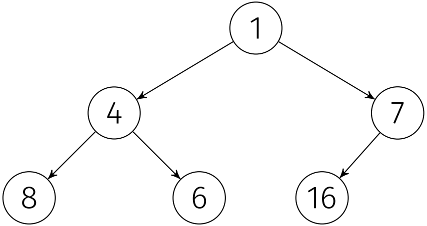
Binary Search Trees

- Search for Key.
- Insert at the reached empty leaf (`null`).



MinHeap

- Insert at the very next free spot (back of the array).
- Restore Heap-Condition: `siftUp` (climb successively).



Exercise: Insert $4, 8, 16, 1, 6, 7$ into empty Search Tree/Min-Heap.

Repetition: Binary Trees, Deleting a Key

Binary Search Trees

- Replace key k by symmetric successor n.
- Careful: What about right child of n?

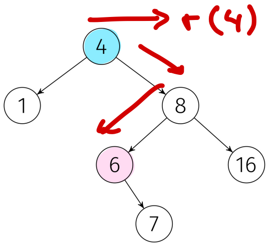
MinHeap

- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` *or* `siftUp`.

Repetition: Binary Trees, Deleting a Key

Binary Search Trees

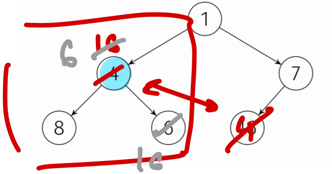
- Replace key k by symmetric successor n.
- Careful: What about right child of n?



MinHeap

delete (4)

- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` *or* `siftUp`.

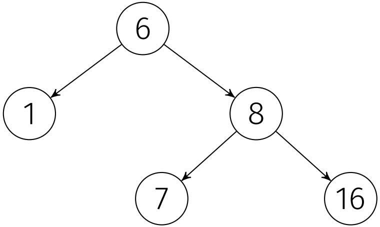


Exercise: Delete 4 from Search Tree/Min-Heap.

Repetition: Binary Trees, Deleting a Key

Binary Search Trees

- Replace key k by symmetric successor n.
- Careful: What about right child of n?



MinHeap

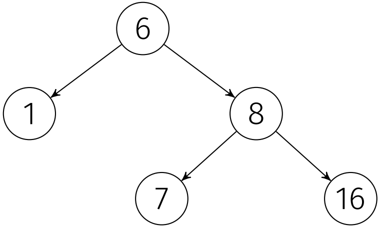
- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` or `siftUp`.

Exercise: Delete 4 from Search Tree/Min-Heap.

Repetition: Binary Trees, Deleting a Key

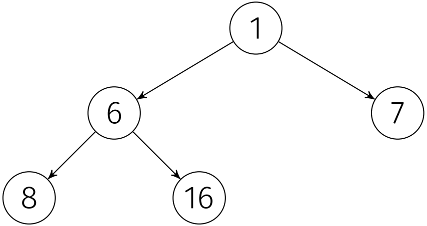
Binary Search Trees

- Replace key k by symmetric successor n.
- Careful: What about right child of n?



MinHeap

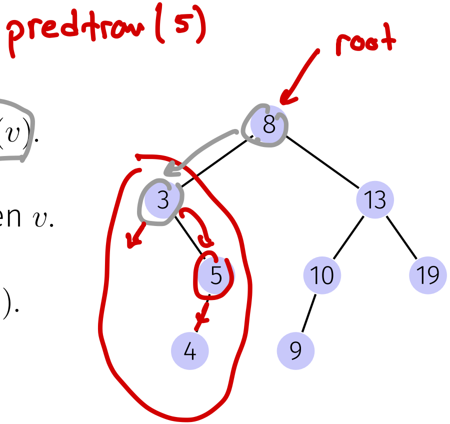
- Replace key by last element of the array.
- Restore Heap-Condition: `siftDown` or `siftUp`.



Exercise: Delete 4 from Search Tree/Min-Heap.

Traversal possibilities

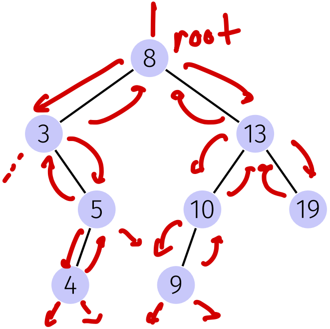
- preorder: v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
- postorder: $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
- inorder: $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.



Traversal possibilities

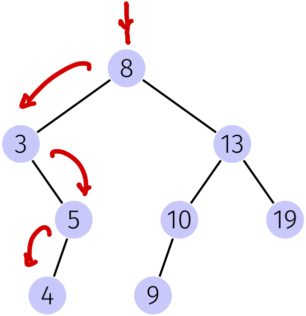
8, 3, 5, 4, 13, 10, 9, 19

- preorder: v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- postorder: $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
- inorder: $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.



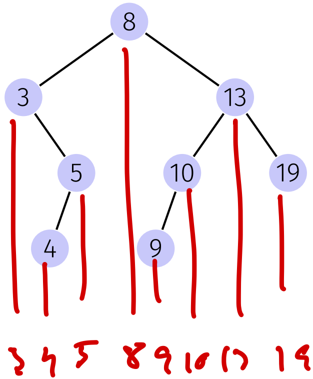
Traversal possibilities

- preorder: v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- postorder: $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
  4, 5, 3, 9, 10, 19, 13, 8
- inorder: $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.



Traversal possibilities

- preorder: v, then $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- postorder: $T_{\text{left}}(v)$, then $T_{\text{right}}(v)$, then v.
  4, 5, 3, 9, 10, 19, 13, 8
- inorder: $T_{\text{left}}(v)$, then v, then $T_{\text{right}}(v)$.
  3, 4, 5, 8, 9, 10, 13, 19



Quiz

Draw a binary search tree each that represents the following traversals. Is the tree unique?

inorder	1 2 3 4 5 6 7 8
preorder	4 3 1 2 8 6 5 7
postorder	1 3 2 5 6 8 7 4

Provide for each order a sequence of numbers from $\{1, \ldots, 4\}$ such that it cannot result from a valid binary search tree

Answers

inorder: any binary search tree with numbers $\{1, \ldots, 8\}$ is valid.

The tree is not unique

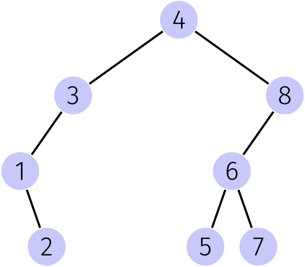
There is no search tree for any non-sorted sequence. Counterexample 1 2 4 3

Answers

< ≥

preorder 4 | 3 1 2 | 8 6 5 7

always root

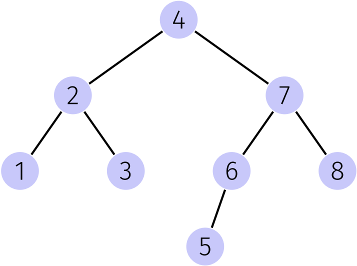


Tree is unique

It must hold recursively that first there is a group of numbers with lower and then with higher number than the first value. Counterexample: 3 1 4 2

Answers

postorder 1 3 2 5 6 8 7 4



Tree is unique

Construction here: `https://www.techiedelight.com/build-binary-search-tree-from-postorder-sequence/`, similar argument as before, but backwards. Counterexample 4 2 1 3

Quiz

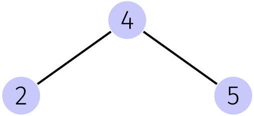
True or false:

1. The preorder is the reversed postorder.
2. The first node in the preorder is always the root.
3. The first node in the inorder is never the root.
4. Inserting the nodes in preorder into an empty tree leads to the same tree.
5. Inserting the nodes in postorder into an empty tree leads to the same tree.
6. Inserting the nodes in inorder into an empty tree leads to the same tree.

Quiz: Solution

True or false:

1. The preorder is the reversed postorder.
   False Preorder: $4, 2, 5$. Postorder: $2, 5, 4$.



2. The first node in the preorder is always the root.
   true (by definition!)
3. The first node in the inorder is never the root.
   False. When the left subtree is empty, the root is the first node inorder.

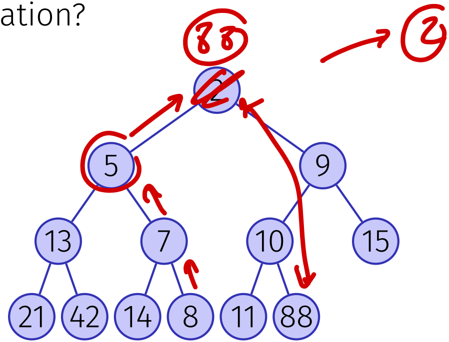
Quiz: Solution

True or false:

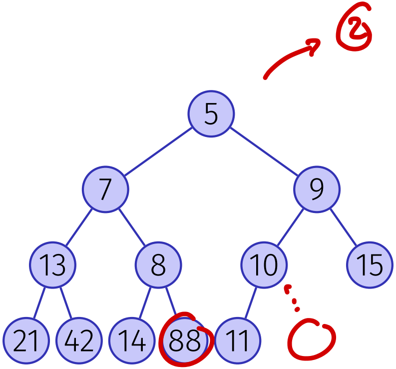
4. Inserting the nodes of a tree in preorder into a new empty tree leads to the same tree.
   True. Since first the root is inserted and then its children, we will get the same tree.
5. Inserting the nodes in postorder into an empty tree leads to the same tree.
   False. But it is true for the reversed postorder!
6. Inserting the nodes in inorder into an empty tree leads to the same tree.
   False. There are many different trees with the same inorder!

Heap

On the following Min-Heap, perform an extract-min operation, including re-establishing the heap-condition, as shown in class. What does the heap look like after the operation?

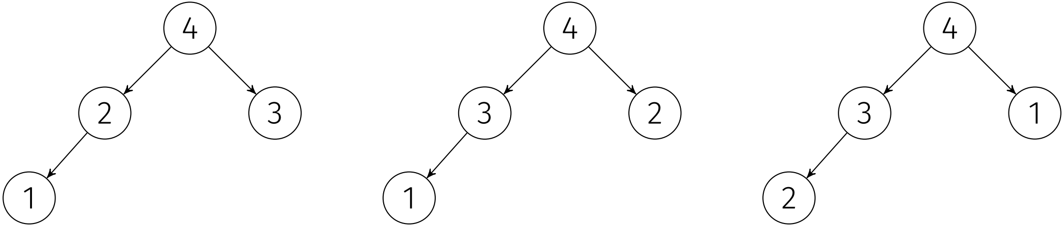


Solution



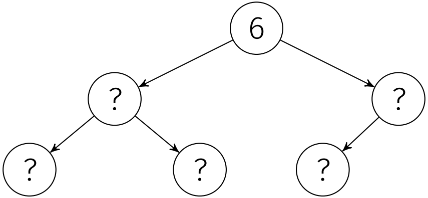
Quiz: Number of MaxHeaps on n keys

Let $N(n)$ denote the number of distinct Max-Heaps which can be built from all the keys $1, 2, \ldots, n$. For example we have
$N(1) = 1,\ N(2) = 1,\ N(3) = 2,\ N(4) = 3$ und $N(5) = 8$.
Find the values $N(6)$ and $N(7)$.



Number of MaxHeaps on n distinct keys

A MaxHeap containing the elements $1, 2, 3, 4, 5, 6$ has the structure:



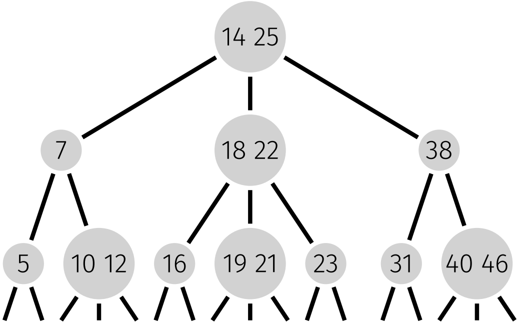
Number of combinations to choose elements for the left subtree: $\binom{5}{3}$.

$$\Rightarrow N(6) = \binom{5}{3} \cdot N(3) \cdot N(2) = 10 \cdot 2 \cdot 1 = 20.$$

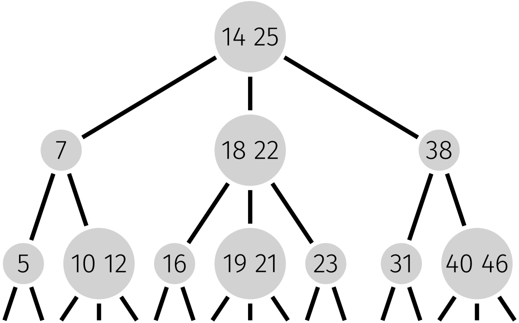
and $N(7) = \binom{6}{3} \cdot N(3) \cdot N(3) = 20 \cdot 2 \cdot 2 = 80.$

5.3 2-3 Trees

Searching

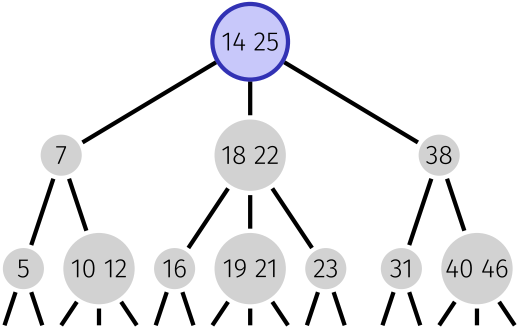


Searching



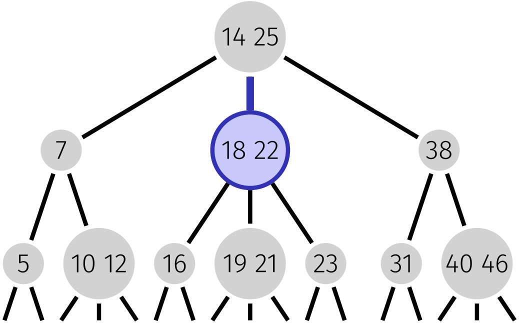
`search`(23)

Searching



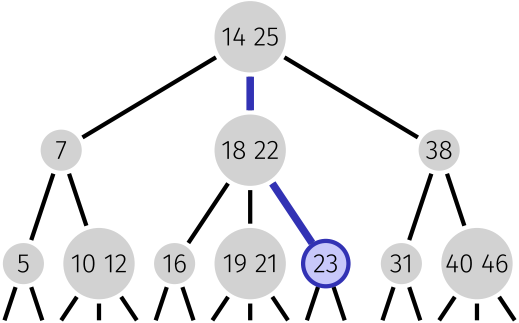
$\texttt{search}(23)$

Searching



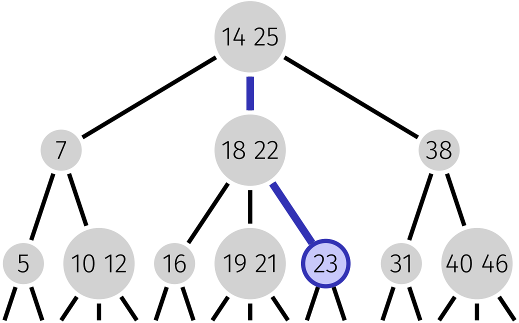
$\texttt{search}(23)$

Searching



`search`(23)

Searching

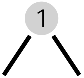


$\texttt{search}(23) \rightarrow \texttt{found}$

2-3 Tree: Insertion

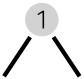
Insert the keys $1, \ldots, 7$ into an (initially empty) 2-3-tree. Draw the tree after every step (`split`/`propagate`, `join`, $\ldots$).

2-3 Tree: Insertion

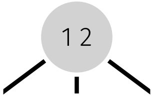


`insert(1)`:
new node

2-3 Tree: Insertion

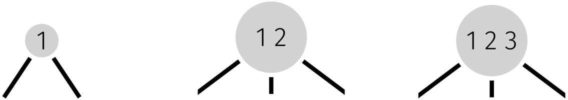


`insert(1)`:
new node



`insert(2)`:
`join`

2-3 Tree: Insertion

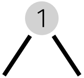


`insert(1)`:
new node

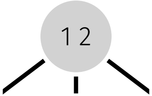
`insert(2)`:
`join`

`insert(3)`:
4-node

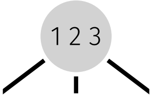
2-3 Tree: Insertion



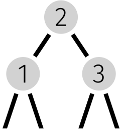
`insert(1)`:
new node



`insert(2)`:
`join`

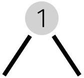


`insert(3)`:
4-node

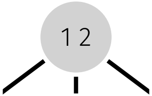


`insert(3)`:
`split`/`propagate`

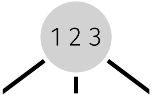
2-3 Tree: Insertion



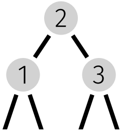
`insert(1)`:
new node



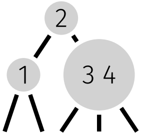
`insert(2)`:
`join`



`insert(3)`:
4-node

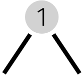


`insert(3)`:
`split/propagate`

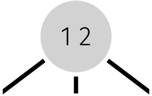


`insert(4)`:
`join`

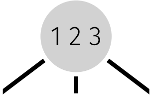
2-3 Tree: Insertion



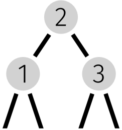
`insert(1):`
new node



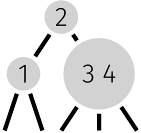
`insert(2):`
`join`



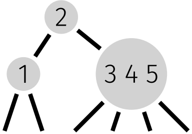
`insert(3):`
4-node



`insert(3):`
`split/propagate`

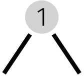


`insert(4):`
`join`

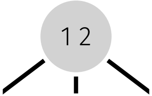


`insert(5):`
4-node

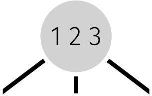
2-3 Tree: Insertion



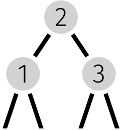
`insert(1):`
new node



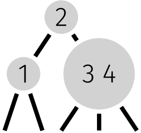
`insert(2):`
`join`



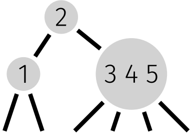
`insert(3):`
4-node



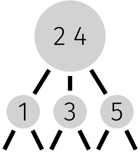
`insert(3):`
`split/propagate`



`insert(4):`
`join`

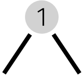


`insert(5):`
4-node

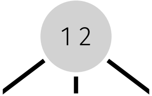


`insert(5):`
`split/propagate`

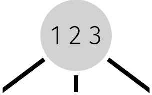
2-3 Tree: Insertion



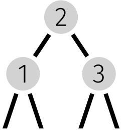
`insert(1)`:
new node



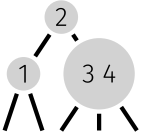
`insert(2)`:
`join`



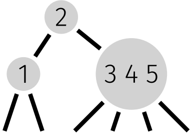
`insert(3)`:
4-node



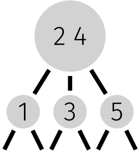
`insert(3)`:
`split/propagate`



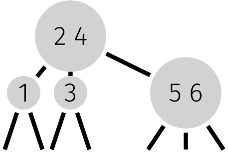
`insert(4)`:
`join`



`insert(5)`:
4-node

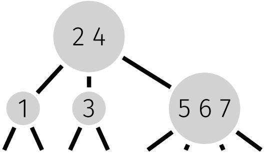


`insert(5)`:
`split/propagate`



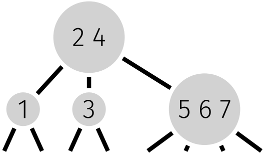
`insert(6)`:
`join`

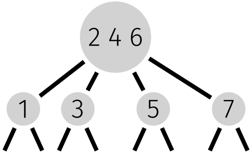
2-3 Tree: Insertion



`insert(7)`:
4-node

2-3 Tree: Insertion

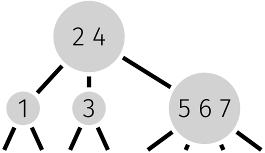




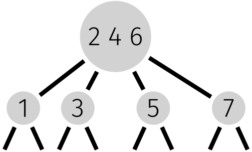
`insert(7)`:
4-node

`insert(7)`:
`split/propagate`

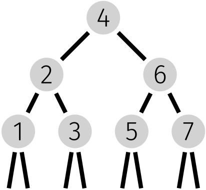
2-3 Tree: Insertion



`insert(7)`:
4-node

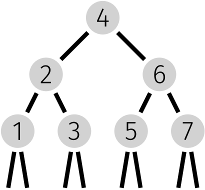


`insert(7)`:
`split/propagate`



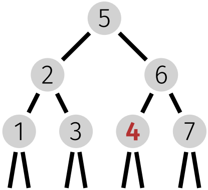
`insert(7)`:
`split/propagate`

2-4 Tree: Deletion

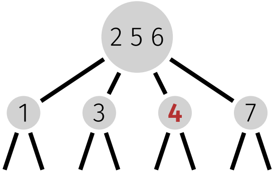


Delete key 4 from the resulting tree.

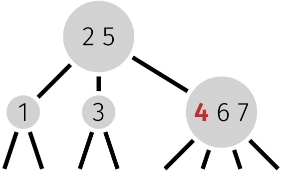
2-4 Tree: Deletion



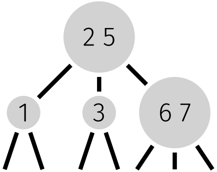
1. swap



2. create 4-node at root



3. combine with sibling

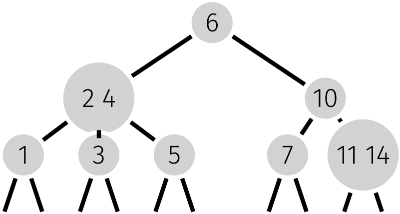


4. delete key

5.4 Red-Black Trees

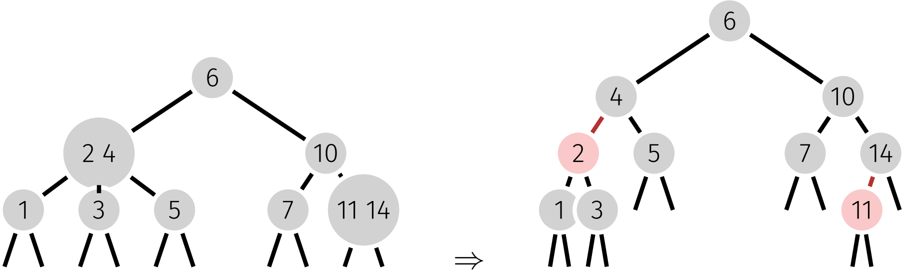
Red-Black Trees

Draw the following 2-3 tree as a red-black tree.



Red-Black Trees

Draw the following 2-3 tree as a red-black tree.



Red-Black Trees: True or False?

1. right spine (path going right from root) has length $\lceil \log_2(n+1) \rceil$.

Red-Black Trees: True or False?

1. right spine (path going right from root) has length $\lceil \log_2(n+1) \rceil$.
   Correct, since there are no right-leaning red edges and we have perfect black balance.

Red-Black Trees: True or False?

1. right spine (path going right from root) has length $\lceil \log_2(n+1) \rceil$.
   Correct, since there are no right-leaning red edges and we have perfect black balance.
2. the number of red edges is at most the number of black edges.

Red-Black Trees: True or False?

1. right spine (path going right from root) has length $\lceil \log_2(n+1) \rceil$.
   Correct, since there are no right-leaning red edges and we have perfect black balance.
2. the number of red edges is at most the number of black edges.
   Wrong, a tree with 2 nodes and one edge must have a red edge but not black edge.

Red-Black Trees: True or False?

1. right spine (path going right from root) has length $\lceil \log_2(n+1) \rceil$.
   Correct, since there are no right-leaning red edges and we have perfect black balance.
2. the number of red edges is at most the number of black edges.
   Wrong, a tree with 2 nodes and one edge must have a red edge but not black edge.
3. All nodes in the left subtree of a node are smaller than the node.

Red-Black Trees: True or False?

1. right spine (path going right from root) has length $\lceil \log_2(n+1) \rceil$.
   Correct, since there are no right-leaning red edges and we have perfect black balance.
2. the number of red edges is at most the number of black edges.
   Wrong, a tree with 2 nodes and one edge must have a red edge but not black edge.
3. All nodes in the left subtree of a node are smaller than the node.
   Correct, since a red-black tree is a search tree.

Red-Black Trees: Insertion

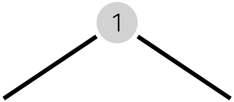
Insert the numbers $1, \ldots, 7$ into an (initially empty) red-black tree and draw the tree after every step.

Red-Black Trees: Insertion

Insert the numbers $1, \ldots, 7$ into an (initially empty) red-black tree and draw the tree after every step.

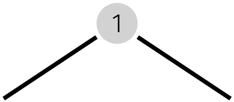
Compare your steps with your result for the 2-3 tree before.

Red-Black Trees: Insertion

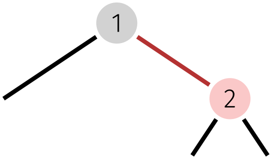


`insert(1)`

Red-Black Trees: Insertion

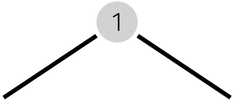


`insert(1)`

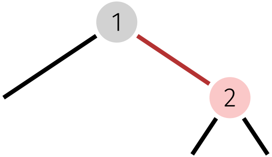


`insert(2): add`

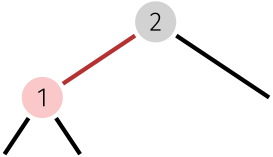
Red-Black Trees: Insertion



`insert(1)`

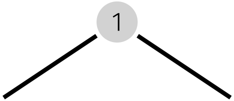


`insert(2): add`

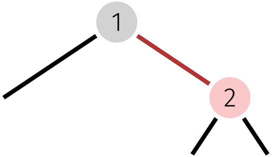


`insert(2): rotate_left`
(since right child red)

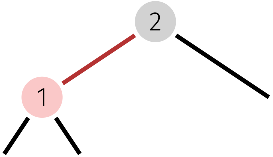
Red-Black Trees: Insertion



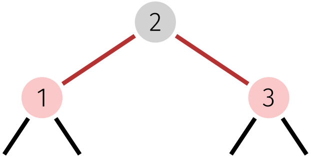
`insert(1)`



`insert(2): add`

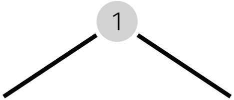


`insert(2): rotate_left`
(since right child red)

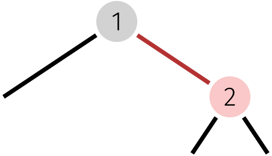


`insert(3): add`

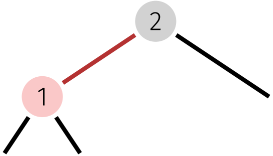
Red-Black Trees: Insertion



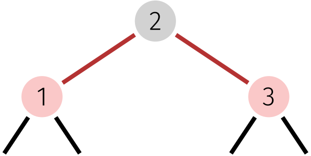
`insert(1)`



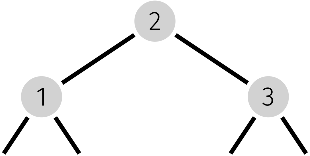
`insert(2): add`



`insert(2): rotate_left`
(since right child red)

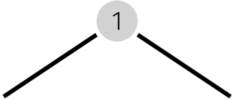


`insert(3): add`

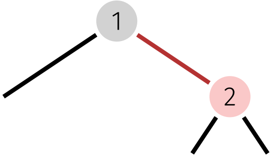


`insert(3): push_up`
(two red children)

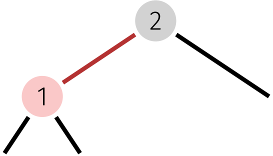
Red-Black Trees: Insertion



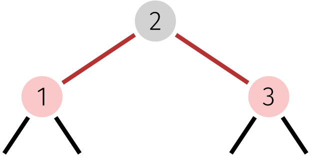
`insert(1)`



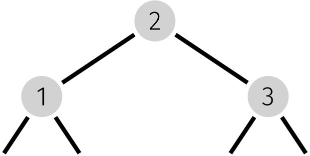
`insert(2): add`



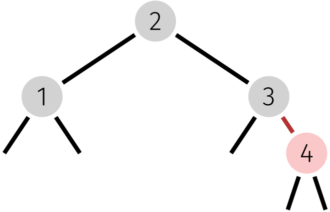
`insert(2): rotate_left`
(since right child red)



`insert(3): add`

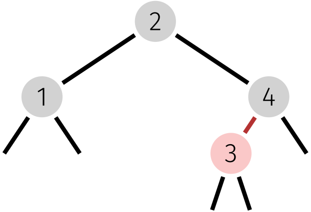


`insert(3): push_up`
(two red children)



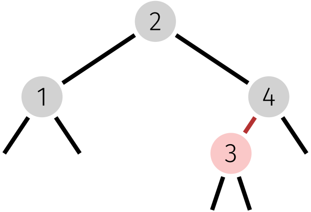
`insert(4): add`

Red-Black Trees: Insertion

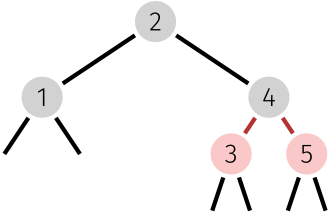


`insert(4): rotate_left`
(since right child red)

Red-Black Trees: Insertion

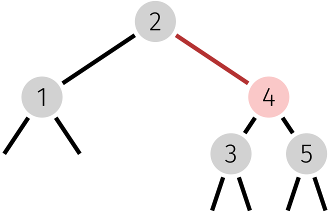


`insert(4): rotate_left`
(since right child red)



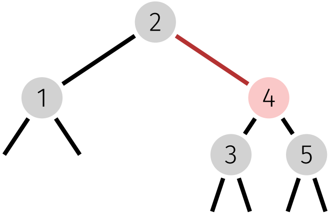
`insert(5): add`

Red-Black Trees: Insertion

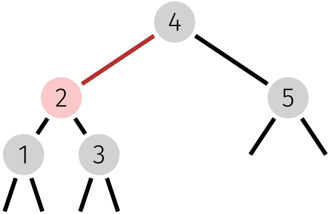


`insert(5): push_up`
(two children red)

Red-Black Trees: Insertion

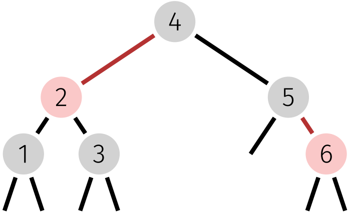


`insert(5): push_up`
(two children red)



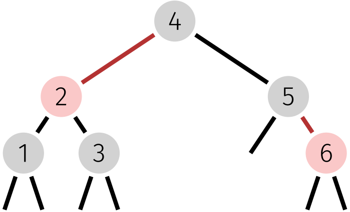
`insert(5): rotate_left`
(since right child red)

Red-Black Trees: Insertion

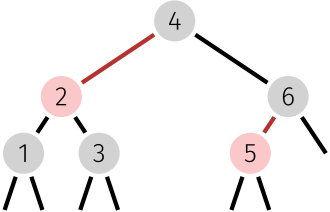


`insert(6): add`

Red-Black Trees: Insertion

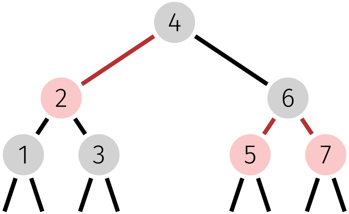


`insert(6): add`



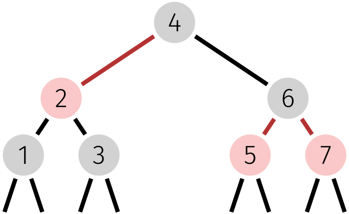
`insert(6): rotate_left`
(since right child red)

Red-Black Trees: Insertion

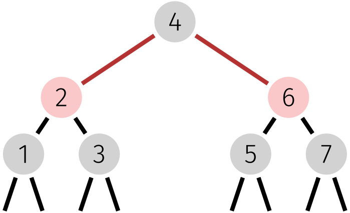


`insert(7): add`

Red-Black Trees: Insertion

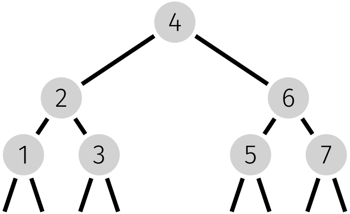


`insert(7): add`



`insert(7): push_up`
(since two children red)

Red-Black Trees: Insertion



`insert(7): push_up`
(since both children red)

"*But how do I do . . . when . . . ?*"

"*But how do I do . . . when . . . ?*"

Study the lecture slides

- they have answers to and algorithms for every case!

6. Code-Example

Code-Example

Exercise class 06: Binary Trees on Code-Expert

- Binary Tree: Simple Tasks
- Augmenting a Binary Search Tree

7. Old Exam Question

Heap

In der folgenden Tabelle ist ein Min-Heap in seiner üblichen Form gespeichert. Wie sieht die Tabelle aus, nachdem `ExtractMin` ausgeführt wurde?

The following table comprises a Min-Heap in its canonical form. What does the table look like after `ExtractMin` *has been executed?*

2	5	3	8	7	4	10	13	9	33	11
1	2	3	4	5	6	7	8	9	10	11

1	2	3	4	5	6	7	8	9	10

Heap – Explanation

In der folgenden Tabelle ist ein Min-Heap in seiner üblichen Form gespeichert. Wie sieht die Tabelle aus, nachdem `ExtractMin` ausgeführt wurde?

The following table comprises a Min-Heap in its canonical form. What does the table look like after `ExtractMin` *has been executed?*

2	5	3	8	7	4	10	13	9	33	11
1	2	3	4	5	6	7	8	9	10	11

1	2	3	4	5	6	7	8	9	10

Heap – Solution

In der folgenden Tabelle ist ein Min-Heap in seiner üblichen Form gespeichert. Wie sieht die Tabelle aus, nachdem `ExtractMin` ausgeführt wurde?

The following table comprises a Min-Heap in its canonical form. What does the table look like after `ExtractMin` *has been executed?*

2	5	3	8	7	4	10	13	9	33	11
1	2	3	4	5	6	7	8	9	10	11

3	5	4	8	7	11	10	13	9	33
1	2	3	4	5	6	7	8	9	10

8. Outro

General Questions?

See you next time

Have a nice easter break!