Institut for Matematik og Datalogi
Syddansk Universitet, Odense

20. March 2025

# Exercise Sheet Week 6

# DSK814

## A(I). To be solved in the practice sessions

1. Draw all possible binary search trees of height 2 and contain four nodes with keys 1, 2, 3, and 4.

2. Cormen et al., 4. Edition: Exercise 12.2-1 (page 319), part a–c [Cormen et al., 3. Edition: Exercise 12.2-1 (page 293), part a–c]

   You are searching for the number 363 in a binary search tree containing numbers between 1 and 1000. Which of the following sequences cannot be the sequence of nodes examined?

   a. $2, 252, 401, 398, 330, 344, 397, 363$
   b. $924, 220, 911, 244, 898, 258, 362, 363.$
   c. $925, 202, 911, 240, 912, 245, 363.$

3. Cormen et al., 4. Edition: Exercise 12.2-3 (page 320) [Cormen et al., 3. Edition: Exercise 12.2-3 (page 293)]

   Write the TreePredecessor procedure.

4. Cormen et al., 4. Edition: Exercise 12.1-5 (page 315) [Cormen et al., 3. Edition: Exercise 12.1-5 (page 289)]

   Argue that since sorting $n$ elements takes $\Omega(n \log n)$ time in the worst case in the comparison model, any comparison-based algorithm for constructing a binary search tree from an arbitrary list of $n$ elements takes $\Omega(n \log n)$ time in the worst case.

5. Consider the task of sorting $n$ integers with values between 0 and $n^4$. Explain the asymptotic worst-case running times for the following sorting algorithms: CountingSort, RadixSort, QuickSort, MergeSort and InsertionSort.

6. Use CountingSort to sort the array $A = [7, 4, 1, 2, 6, 4, 0, 4, 4, 4, 7, 2]$. Provide the state of array $C$ after each important step of the algorithm.

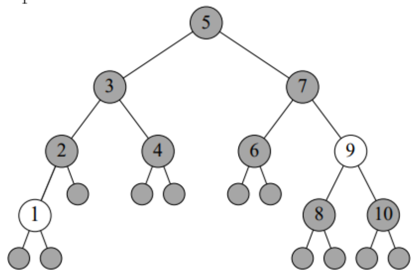
## A(II). Solve in the practice sessions

1. Cormen et al., 4. Edition: Exercise 13.1-2 (page 334) [Cormen et al., 3. Edition: Exercise 13.1-2 (page 311)].

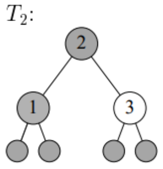
   Draw the red-black tree that results after Tree-Insert is called on the tree in Figure 13.1 with key 36. If the inserted node is colored red, is the resulting tree a red-black tree? What if it is colored black?

2. Which of the following four trees are red-black trees? Justify your answers.

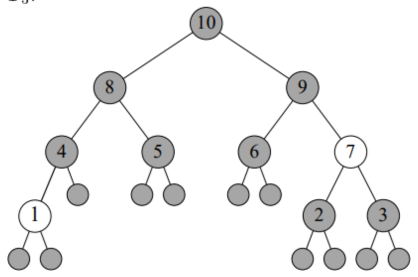
$T_1$:



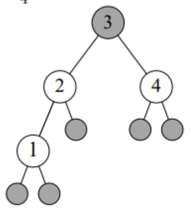
$T_2$:



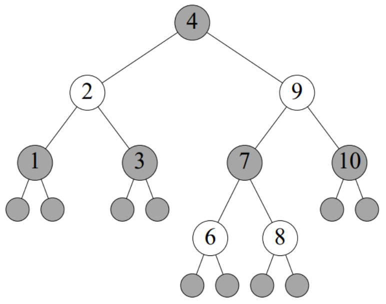
$T_3$:



$T_4$:



3. Consider the following red-black tree. Insert an element with key 5 and show the individual steps in the insertion process. Draw the tree as it looks after the insertion.



4. Cormen et al., 4. Edition: Exercise 13.3-2 (page 346) [Cormen et al., 3. Edition: Exercise 13.3-2 (page 322)].

   Show the red-black trees that result after successively inserting the keys $41, 38, 31, 12, 19, 8$ into an initially empty red-black tree.

5. Cormen et al., 4. Edition: Exercise 13.1-6 (page 335) [Cormen et al., 3. Edition: Exercise 13.1-6 (page 312)].

   What is the largest possible number of internal nodes in a red-black tree with black-height $k$? What is the smallest possible number?

6. ($*$) Cormen et al., 4. Edition: Exercise 12.3-3 (page 326) [Cormen et al., 3. Edition: Exercise 12.3-3 (page 299)].

   You can sort a given set of $n$ numbers by first building a binary search tree containing these numbers (using TREE-INSERT repeatedly to insert the numbers one by one) and then printing the numbers by an in-order tree walk. What are the worst-case and best-case running times for this sorting algorithm?

   Note that the problem assumes unbalanced search trees (chapter 12). Afterwards, answer the problem for red-black trees instead of unbalanced binary search trees.

## B(I). Solve at home before tutorial in week 7

1. Cormen et al., 4. Edition: Exercise 12.1-2 (page 315) [Cormen et al., 3. Edition: Exercise 12.1-2 (page 289)]

Remember the min-heap property: for every node $i$ other than the root holds that $A[\text{PARENT}(i)] \leq A[i]$.

What is the difference between the binary-search-tree property and the min-heap property? Can the min-heap property be used to print out the keys of an $n$-node tree in sorted order in $O(n)$ time? Show how, or explain why not.

2. (∗) Cormen et al., 4. Edition: Exercise 13.4-9 (page 355) [Cormen et al., 3. Edition: Exercise 14.2-4 (page 348)]

   Consider the operation RB-ENUMERATE$(T, r, a, b)$ which outputs all the keys $k$ such that $a \leq k \leq b$ in a subtree rooted at node $r$ in an $n$-node red-black tree $T$. Describe how to implement RB-ENUMERATE in $\Theta(m + \log n)$ time, where $m$ is the number of keys that are output. Assume that the keys in $T$ are unique and that the values $a$ and $b$ appear as keys in $T$. How does your solution change if $a$ and $b$ might not appear in $T$?

   *Hint: Far more often the operation is called* RANGESEARCH *rather than* ENUMERATE. *Either make a variation of* INORDER-TREE-WALK *(Cormen et al., 4. Edition: page 314 [Cormen et al., 3. Edition: page 288]) or repeatedly use* TREE-SUCCESSOR *(Cormen et al., 4. Edition: page 319 [Cormen et al., 3. Edition: page 292]). This makes stating the algorithm easy while the challenging part is to find an argument for the running time.*

## B(II). Solve at home before tutorial in week 7

1. Implement COUNTINGSORT in Java or Python based on the book's pseudocode (Cormen et al., 4. Edition: page 209 [Cormen et al., 3. Edition: page 195]). Remember to set an upper limit $k$ on the `int`'s that you generate as input. Test that your code works by generating arrays with different contents and sorting them. Add timing to your code (only the sorting itself, not the part of the program that generates the array contents).

   Then run your code with input that is random `int`'s for the interval $[0; k]$ for $k = n/50$, $n$, an $50n$ (i.e., three values of $k$ for each value of $n$). You can use `java.util.Random.nextInt(k+1)` in Java or `random.randint(0,k)` in to generate the numbers. Do this for at least three different values of $n$ (number of items to sort), selecting values that will cause the program to spend from about 100 to about 5000 milliseconds for the $k = n$ run. Repeat each individual run three times and find the average number of milliseconds spent for the three runs. Divide the resulting numbers by $n + k$ and check how well the analysis fits with practice – the resulting numbers should be constant according to the analysis.

Compare with your running times for the same $n$ with QuickSort or MergeSort from previous exercises. Is QuickSort or CountingSort faster? Does it depend on $k$ (for retained $n$)?