The homework assignment is available at:
`http://www.jennylam.cc/courses/146-s17/homework07.html`

1. (a) At level i,
   - the input is of size $n/3^i$,
   - the amount of work per node is constant or 1,
   - the number of nodes is 2^i,
   - the total work at this level is $2^i \cdot 1 = 2^i$.

   The levels range between 0 and to $\log_3 n$, so the total amount of work or time is

$$T(n) = \sum_{i=0}^{\log_3 n} 2^i = \frac{2^{\log_3 n+1} - 1}{2-1} = 2n^{\log_3 2} - 1 \text{ or } \Theta(n^{\log_3 2}).$$

   (b) At level i,
   - the input is of size $n/4^i$,
   - the amount of work per node is linear in the input size or $n/4^i$.
   - the number of nodes is 5^i,
   - the total work at this level is $5^i \cdot n/4^i = n(5/4)^i$.

   The levels range between 0 and to $\log_4 n$, so the total amount of work or time is

$$T(n) = \sum_{i=0}^{\log_4 n} n(5/4)^i = n\frac{(5/4)^{\log_4 n+1} - 1}{5/4 - 1} = 4n\left(\frac{5}{4} \cdot \frac{5^{\log_4 n}}{4^{\log_4 n}} - 1\right) = 4n\left(\frac{5n^{\log_4 5}}{4n} - 1\right)$$

   or $\Theta(n^{\log_4 5})$, which is superlinear but subquadratic since $1 < \log_4 5 < 2$.

   (c) At level i,
   - the input is of size $n/7^i$,
   - the amount of work per node is linear in the input size or $n/7^i$.
   - the number of nodes is 7^i,
   - the total work at this level is $7^i \cdot n/7^i = n$.

   The levels range between 0 and to $\log_7 n$, so the total amount of work or time is

$$T(n) = \sum_{i=0}^{\log_7 n} n = n \log_7 n \text{ or } \Theta(n \log n).$$

   (d) At level i,
   - the input is of size $n/3^i$,
   - the amount of work per node is quadratic in the input size or $(n/3^i)^2 = n^2/9^i$.
   - the number of nodes is 9^i,
   - the total work at this level is $9^i \cdot n^2/9^i = n^2$.

   The levels range between 0 and to $\log_3 n$, so the total amount of work or time is

$$T(n) = \sum_{i=0}^{\log_3 n} n^2 = n^2 \log_3 n \text{ or } \Theta(n^2 \log n).$$

(e) At level i,

- the input is of size $n/2^i$,
- the amount of work per node is cubic in the input size or $(n/2^i)^3 = n^3/8^i$.
- the number of nodes is 8^i,
- the total work at this level is $8^i \cdot n^3/8^i = n^3$.

The levels range between 0 and to $\log_2 n$, so the total amount of work or time is

$$T(n) = \sum_{i=0}^{\log_2 n} n^3 = n^3 \log_2 n \text{ or } \Theta(n^3 \log n).$$

(f) Repeatedly expanding $T(\cdot)$, we have

$$T(n) = 2 + T(n-1) = 2 + 2 + T(n-2) = \underbrace{2 + 2 + \cdots + 2}_{n \text{ times}} = 2n = \Theta(n).$$

(g) $T(n) = T(n-1) + n^c$, where $c \geq 1$.
Repeatedly expanding $T(\cdot)$, we have

$$\begin{aligned} T(n) &= n^c + T(n) \\ &= n^c + (n-1)^c + T(n-2) \\ &= n^c + (n-1)^c + \cdots + 1 \\ &\leq \underbrace{n^c + n^c + \cdots + n^c}_{n \text{ times}} \\ &= n^{c+1} \end{aligned}$$

so $T(n)$ is $O(n^{c+1})$. Moreover,

$$\begin{aligned} T(n) &= n^c + (n-1)^c + (n-2)^c + \cdots + 1 \\ &\geq n^c + (n-1)^c + \cdots + (n/2)^c \\ &\geq \underbrace{(n/2)^c + \cdots + (n/2)^c}_{n/2 \text{ times}} \\ &= (n/2)^{c+1}. \end{aligned}$$

The first inequality comes from dropping the lower half of the terms, and the second inequality comes from the fact that $x^c \geq y^c$ if $x \geq y \geq 1$ and $c \geq 1$. Therefore, $T(n)$ is $\Omega(n^{c+1})$. Therefore, this shows that $T(n)$ is $\Theta(n^{c+1})$.

(h) Repeatedly expanding $T(\cdot)$, we have

$$\begin{aligned} T(n) &= c^n + T(n-1) \\ &= c^n + c^{n-1} + T(n-2) \\ &= c^n + c^{n-1} + c^{n-2} + \cdots + c^0 \\ &= \frac{c^{n+1} - 1}{c - 1} \end{aligned}$$

which is dominated by c^{n+1} because $c > 1$. Therefore, $T(n)$ is $\Theta(c^{n+1})$ or $\Theta(c^n)$.

2. (a) Write a recurrence equation for the time complexity of the following program, and use the recursion-tree method to solve the equation.

Solution. The recurrence equation for the time complexity of this program is $T(n) = 3T(n/2) + n + n^2$. At level i of the recursion tree,

- the input is of size $n/2^i$,
- the amount of work per node is quadratic in the input size or $(n/2^i) + (n/2^i)^2 = n/2^i + n^2/4^i$.
- the number of nodes is 3^i,
- the total work at this level is $3^i \cdot (n/2^i + n^2/4^i) = n(3/2)^i + n^2(3/4)^i$.

The levels range between 0 and to $\log_2 n$, so the total amount of work or time is

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\log_2 n} n(3/2)^i + n^2(3/4)^i \\
&= n\sum_{i=0}^{\log_2 n}(3/2)^i + n^2\sum_{i=0}^{\log_2 n}(3/4)^i \\
&= n\frac{(3/2)^{\log_2 n+1} - 1}{3/2 - 1} + n^2\frac{1-(3/4)^{\log_2 n+1}}{1-3/4} \\
&= 2n\left(\frac{3\cdot 3^{\log_2 n}}{2\cdot 2^{\log_2 n}} - 1\right) + 4n^2\left(1-(3/4)^{\log_2 n+1}\right) \\
&= 2n\left(\frac{3n^{\log_2 3}}{2n} - 1\right) + 4n^2\left(1-(3/4)^{\log_2 n+1}\right)
\end{aligned}$$

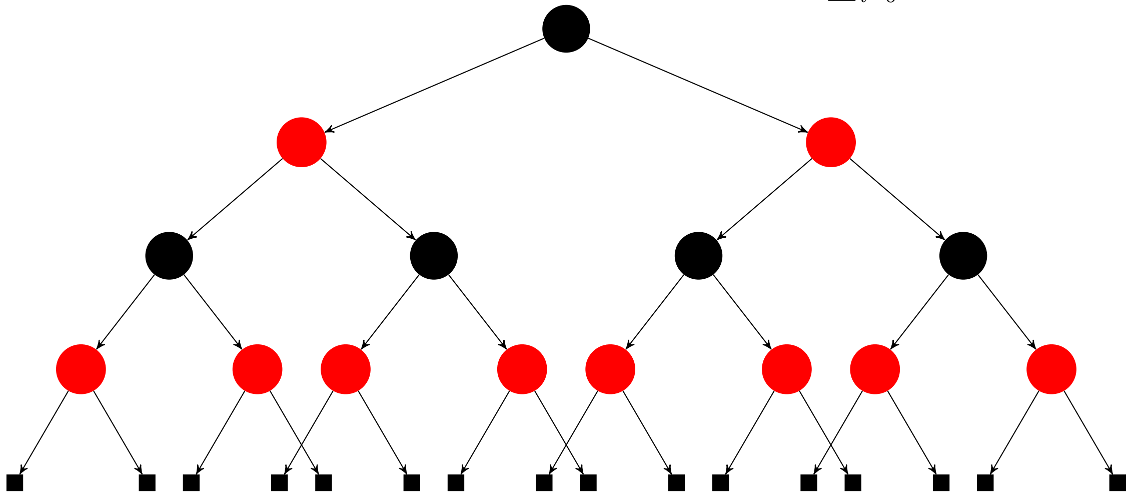
which is $\Theta(n^{\log_2 3} + n^2)$ or $\Theta(n^2)$. •

(b) You are given an array of n elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. Show how to remove all duplicates from the array in time $O(n \log n)$.

Solution. Sort the array using mergesort, duplicates now form contiguous blocks. So create a new array; scan left to right, and copy each new value seen into the new array. Sorting is $O(n \log n)$, and scanning and copying is $O(n)$, for a total time of $O(n \log n)$. •

3. (a) What is the largest possible number of internal nodes in a red-black tree with black height k?

Solution. Assume for simplicity that we start with a red-black tree with black height k. We cannot increase the number of internal nodes (and keep the black height fixed) by adding more black nodes as this will increase the black height. Since every black node except for the root can have a red parent, we can try to add as many red nodes as possible without increasing the black height simply by alternating red and black along any path down the tree. For a tree of black height 2, the height is 4 and the maximal number of internal nodes is $\sum_{i=0}^{3} 2^i = 2^4 - 1 = 15$:

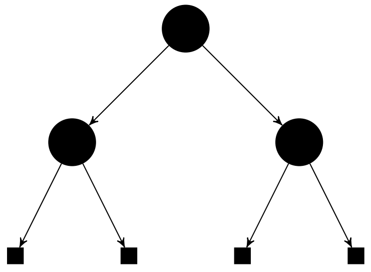


In general, the largest possible number of internal nodes for a tree of black height k is $\sum_{i=0}^{2k-1} 2^i = 2^{2k} - 1$. Note: we subtract 1 from the total height because we only want to include the internal nodes. •

(b) What is the smallest possible number?

Solution. Using a similar idea, we try to remove as many red nodes while fixing the black height in order to decrease the number of internal nodes. We cannot further remove any of the black nodes without violating the black-height property. Therefore, a red-black tree of black-height k with the fewest number of internal nodes is a full binary tree of height k consisting of only black nodes. This has $\sum_{i=0}^{k-1} 2^i = 2^k - 1$ internal nodes.

For example, for $k = 2$, the minimum number of internal nodes is 3.

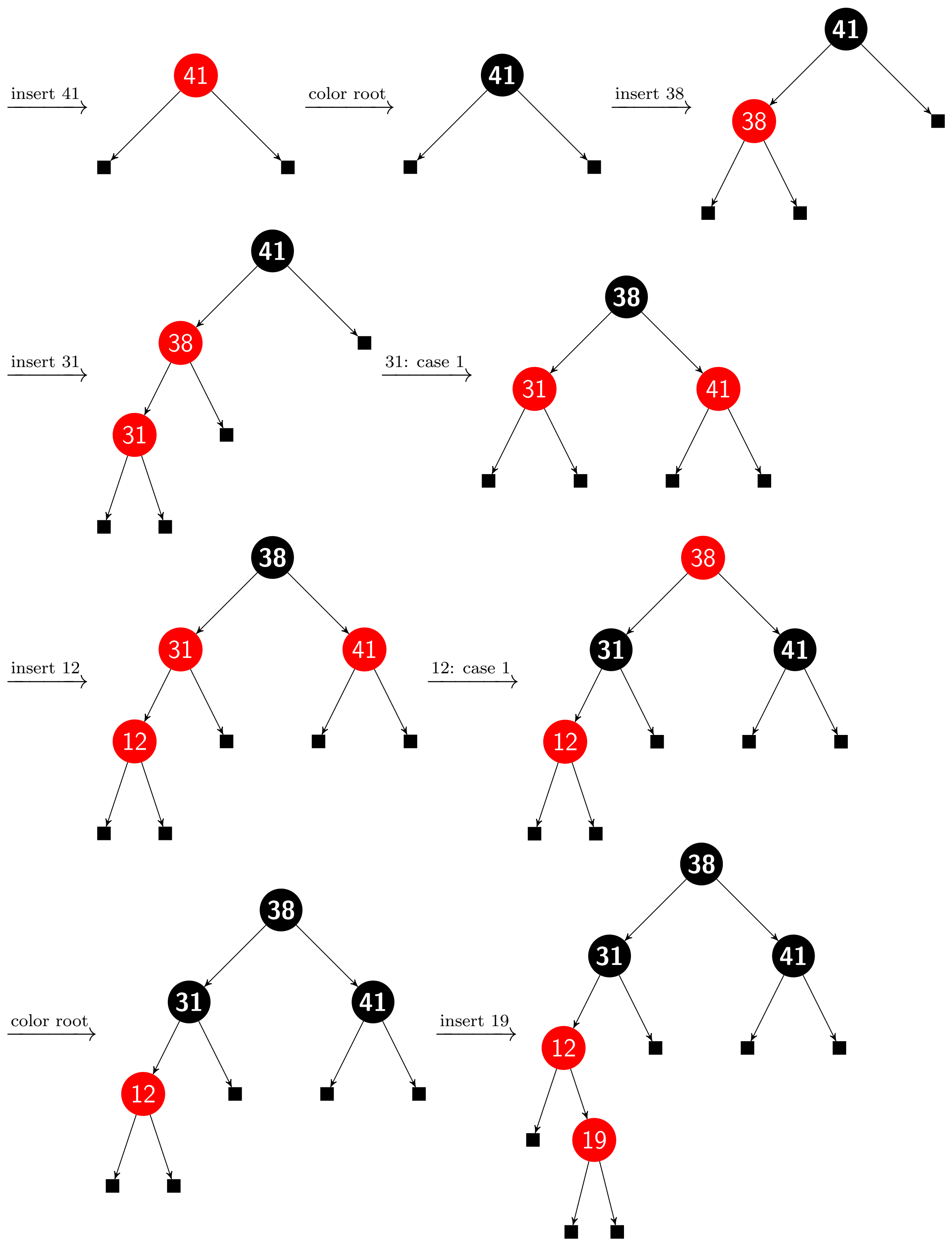


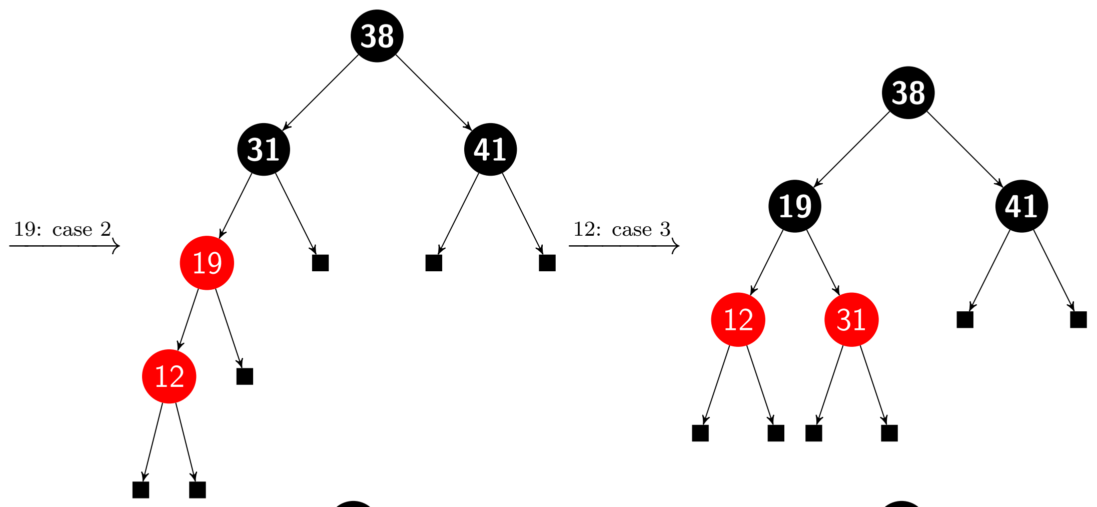
•

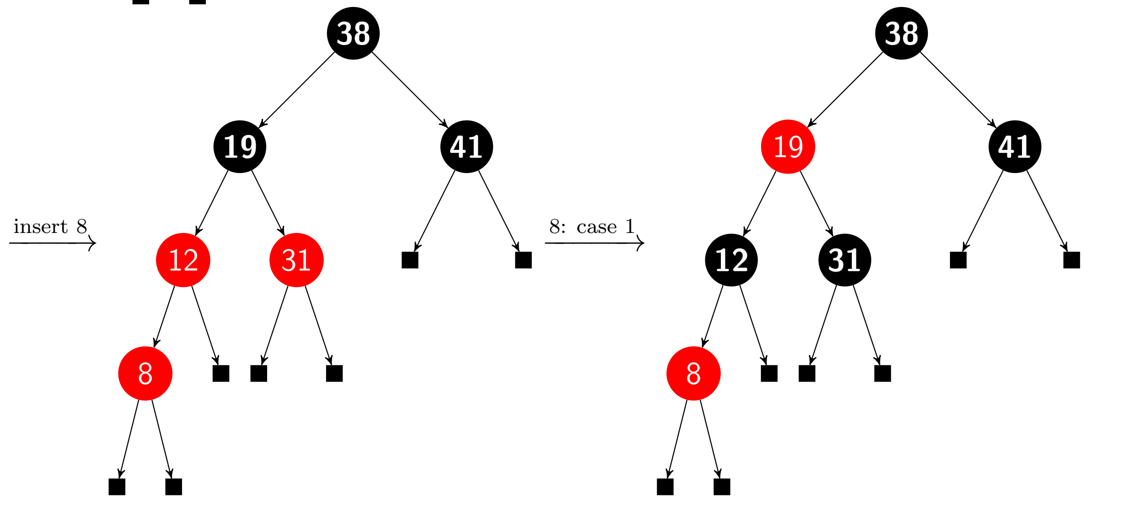
4. The keys 41, 38, 31, 12, 19, 8 are successively inserted into an initially empty red-black tree.

(a) Show the red-black tree after each of these insertions, as well as the intermediate steps that result from fixing up the tree after an insertion. Label these transformations by whether they represent case 1, 2 or 3. See figure 13.4 in the book for what your solution should look like.

Solution.


insert 41
color root
insert 38
insert 31
31: case 1
insert 12
12: case 1
color root
insert 19

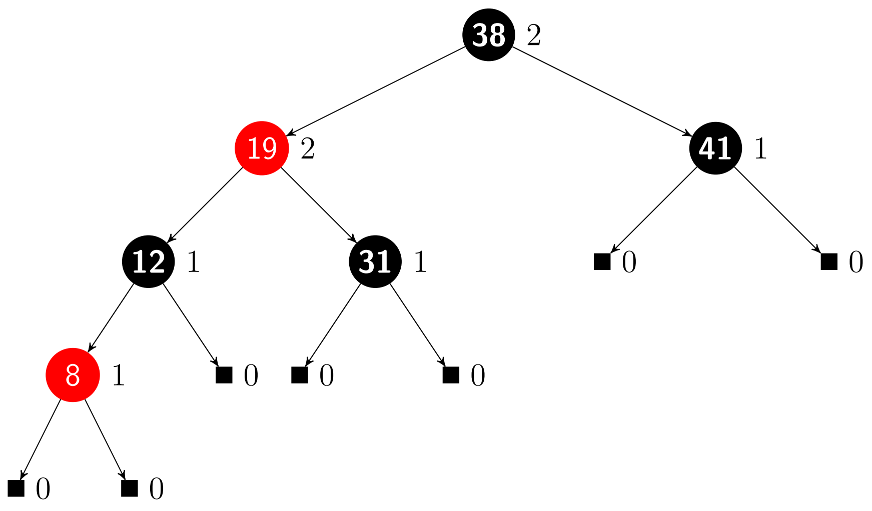




•

(b) Show the black-heights of all the nodes of the last red-black tree in part (a).

Solution.



•