

Solutions to 2015-10-22 Exam

Algorithms and Data Structures 1 (1DL210)

October 14, 2016

Problem 1

Answer

In the order growing asymptotic growth:

$(\frac{1}{4})^n$, $(\frac{1}{2})^{2n}$, $\log(n)$, $\log(n^3)$, $4 \log(n)$, $\log(2^n)$, $n \log(n)$. Note that:

- The two functions $(\frac{1}{4})^n$ and $(\frac{1}{2})^{2n}$ have the same asymptotic growth.
- The functions $\log(n)$, $\log(n^3)$ and $4 \log(n)$, also have the same asymptotic growth.

Problem 2

Answer

- (i) True.
- (ii) True.
- (iii) True.
- (iv) True.
- (v) True.
- (vi) True.

Problem 3

Answer

- a) SAMEVALUEEVERYWHERE gets as input an array and checks whether all elements of the array are identical. If it is the case it returns True, if not, it returns False.
- b) The worse case running time happen for arrays where all elements are identical. In that case the inner loop (line 4 to 6) has $n - j$ iterations for j going from 1 to $n - 1$. The time complexity in the worse case can therefore be expressed as follows:

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

Where $c > 0$ is some constant. Clearly we have that $T(n) = \Theta(n^2)$, i.e. SAMEVALUEEVERYWHERE is quadratic in the worse case.

- c) Another way to check whether all elements of the array are identical is simply to do a linear traversal of the input array and check whether each pair of consecutive elements are identical.

SAMEVALUEEVERYWHERE(A)

```
1   $n \leftarrow A.length$ 
2  for  $j \leftarrow 1$  to  $n - 1$ 
3      do if  $A[j] \neq A[j + 1]$ 
4          then return FALSE
5  return TRUE
```

This algorithm is linear in the worse case.

Problem 4

Answer

One way to find a pair of elements x and y from the array S such that $x = y + 1$ is, first, to sort the array S and second, since S only contains integers, to do a linear traversal of the array and check whether one of the pairs (x, y) of consecutive elements in the sorted array are such that $x = y + 1$.

FINDXY(S)

```

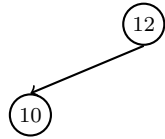
1  MERGESORT( $S$ )
2   $n \leftarrow S.length$ 
3  for  $j \leftarrow 1$  to  $n - 1$ 
4      do if  $S[j + 1] = S[j] + 1$ 
5          then return ( $S[j + 1], S[j]$ )
6  return NIL

```

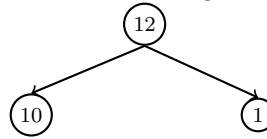
We use MERGESORT to sort the array. Since the time complexity of the traversal of the array is linear in the worse case, and since the time complexity of the merge sort is in $n \log(n)$, we conclude that FINDXY is in $n \log(n)$.

Problem 5

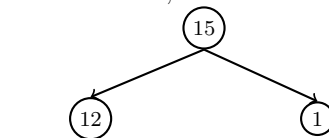
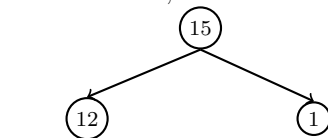
We start with an empty heap \perp , after we insert 10 we get , after we insert 12 we get



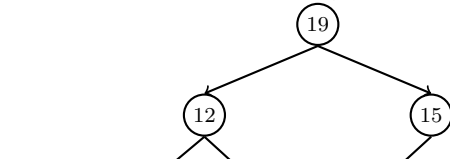
, after we insert 1 we get



, after we insert 15 we get

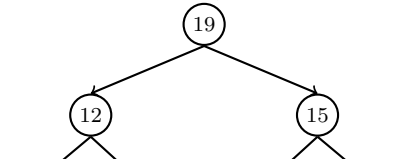


, after we insert 8 we get

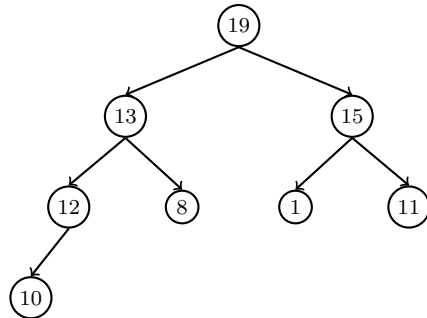


, after we insert

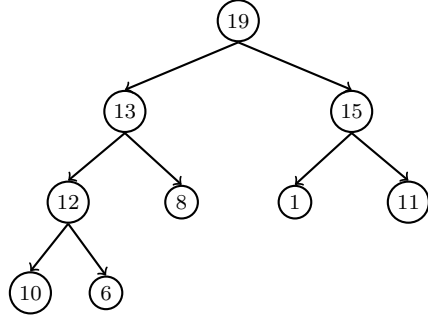
19 we get



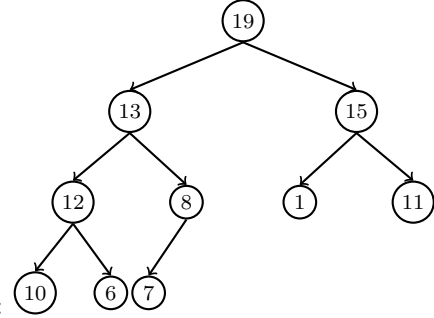
, after we insert 11 we get



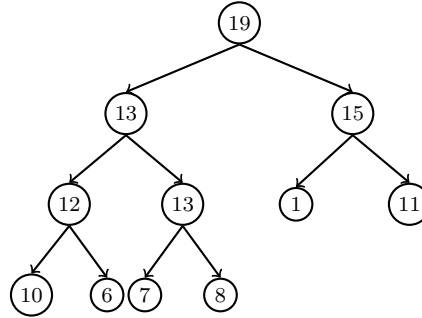
after we insert 13 we get



, after we insert 6 we get:

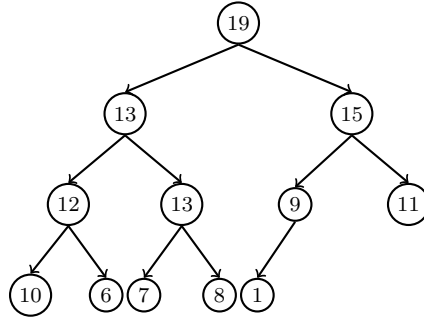


, after we insert 7 we get:



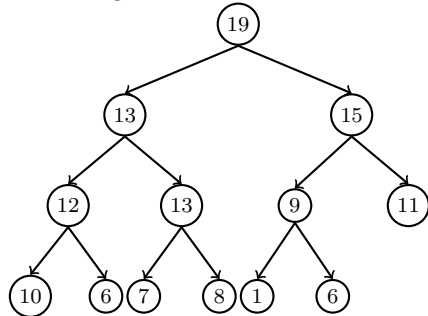
after we insert 13 we get:

, after we insert



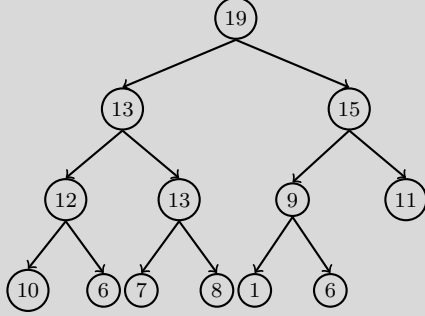
9 we get:

, finally after we insert 6 we get:



Answer

The max-heap we obtain after successively inserting the elements 10 12 1 15 8 19 11 13 6 7 13 9 6 into an empty heap is:

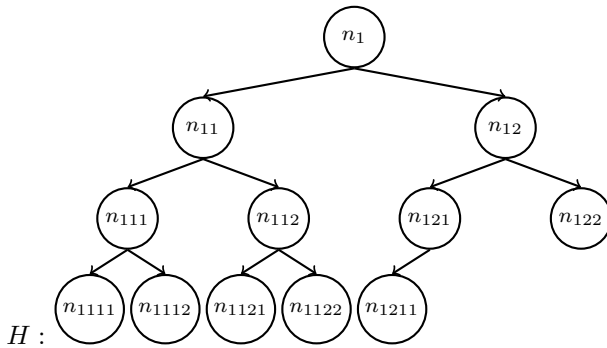


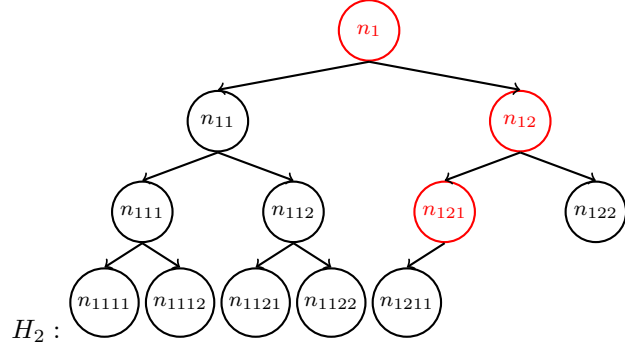
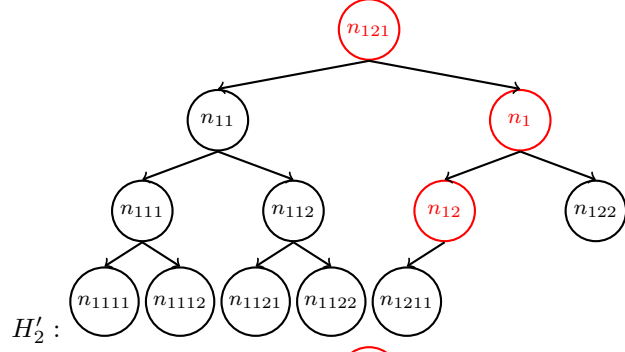
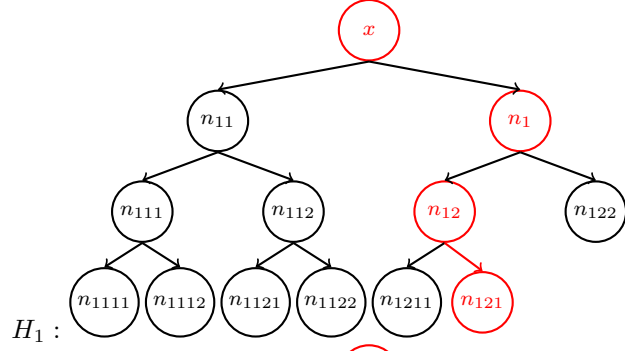
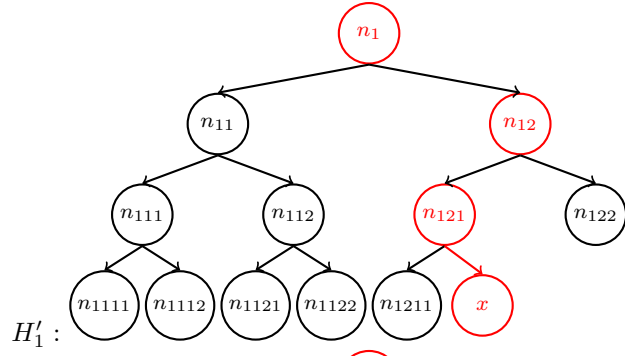
Problem 6

It is the case that $H_2 = H$.

Here is the justification (We use heaps H, H_1, H'_1, H'_2 and H_2 for illustration):

- The new key x will be first inserted in a leaf (see heap H'_1). Since x is larger than any element present in the heap, it will pop-up along a path in the tree (n_{121}, n_{12}, n_1 in heap H'_1) while pushing all the nodes along that path one level down. Again, since the inserted element x is larger than any other element already present in the heap, the pop-up only stops when the inserted key becomes the root. We obtain heap H_1 .
- When extracting the max element, here the root x , from the heap H_1 , all the nodes that have been pushed down one level in the tree will come back to their original place and so we will have the same tree. More precisely, the delete procedure will:
 - First sets the root of the heap to the last leaf in the heap, i.e. the root becomes n_{121} (see heap H'_2).
 - Then, max-heapify is applied to the new root of the heap. We know already from H being a max-heap that $n_1 \geq n_{11}$ and $n_1 \geq n_{12}$. That means that it is n_1 (and not n_{11}) that is swapped with n_{121} in the first step of the max-heapify. The same reasoning goes till n_{121} reaches its initial place and we obtain the original heap: $H_4 = H$.





Problem 7

Possible answer

- a) Hash function: $h(k) = k \bmod 7$ and hash table size equal to 7. h is really bad for the set of elements present in S , because it maps all of them to slot 1 in the hash table. In fact, $h(1) = h(8) = h(15) = h(22) = h(29) = h(36) = 1$. On the other hand, since 7 is a prime number, h is a good hash function for random input.
- b) Hash function: $h(k) = k \bmod 8$ and hash table size equal to 8. h performs well for the elements in S because it maps them to pairwise different slots of the hash table. In fact, we have $h(1) = 1, h(8) = 0, h(15) = 7, h(22) = 6, h(29) = 5$ and $h(36) = 4$. Thus, there won't be any collision after all elements from S have been inserted in the hash table.
- On the other hand, since $8 = 2^3$ is a power of 2, it will perform poorly for random input.

Problem 8

- a) A little help for this question:

Multiples of 13 are: 0,13,26,39,52,65,78,91,104,117,130,143,156,...

Answer

$$h(152) = h(11 * 13 + 9) = 9$$

$$h(44) = h(3 * 13 + 5) = 5$$

$$h(39) = h(3 * 13 + 0) = 0$$

$$h(22) = h(1 * 13 + 9) = 9$$

$$h(134) = h(10 * 13 + 4) = 4$$

$$h(53) = h(4 * 13 + 1) = 1$$

$$h(144) = h(11 * 13 + 1) = 1$$

$$h(131) = h(10 * 13 + 1) = 1$$

$$h(0) = h(0 * 13 + 0) = 0$$

$$h(135) = h(10 * 13 + 5) = 5$$

0	→ 0 → 39
1	→ 131 → 144 → 53
2	
3	
4	→ 134
5	→ 135 → 44
6	
7	
8	
9	→ 22 → 152
10	
11	
12	





The resulting hash table is:

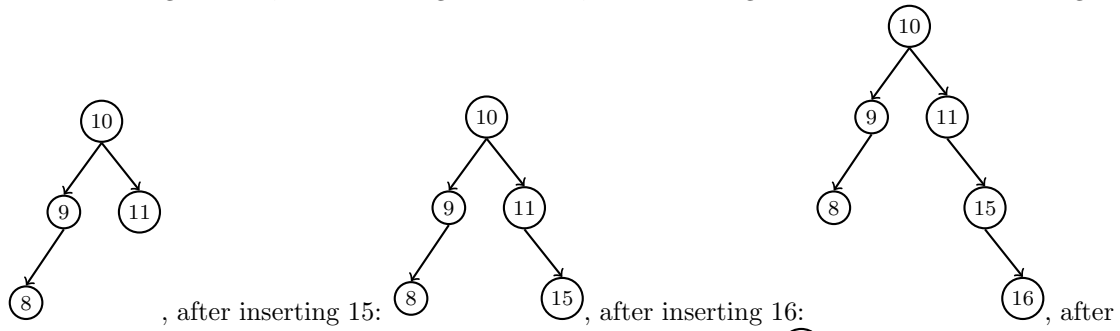
Answer

Increasing the size of the hash table without changing the hash function will not decrease the number of collisions. In fact, the number of collisions will stay exactly the same for the same set of keys.

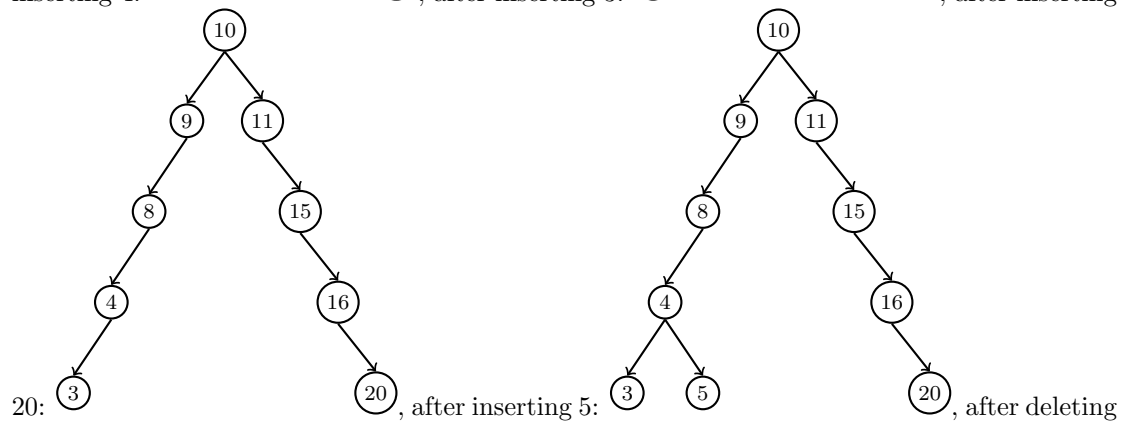
b)

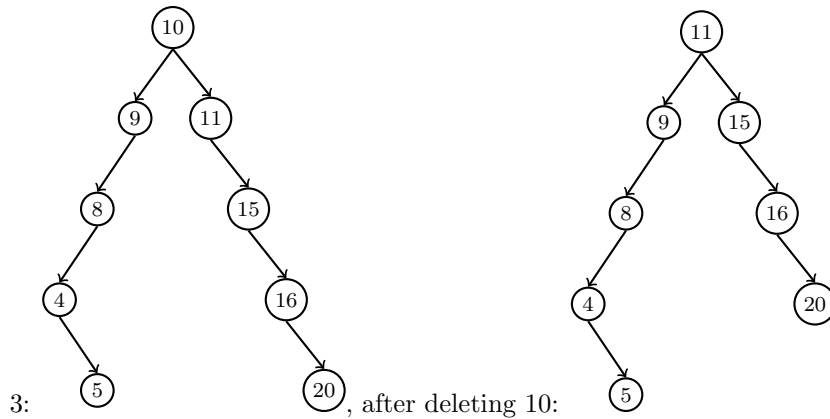
Problem 9

a) After inserting 10: , after inserting 11: , after inserting 9:  , after inserting 8: 

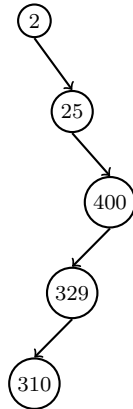


inserting 4: $\textcircled{4}$ $\textcircled{16}$, after inserting 3: $\textcircled{3}$ $\textcircled{16}$, after inserting



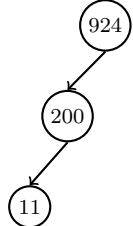


- b) • 2, 25, 400, 329, 310, 134, 397, 363 → shows the following path in the binary search tree:



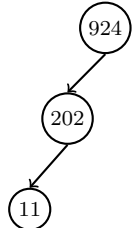
. At node 329, since we're looking for $363 > 329$ we should have went right. This sequence can therefore not be a search sequence of the key 363.

- 924, 200, 11, 24, 89, 258, 362, 363 → shows the following path in the binary search tree:



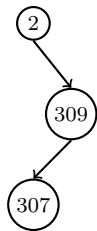
. At node 200, since we're looking for $363 > 200$ we should have went right. This sequence can therefore not be a search sequence of the key 363.

- 925, 202, 11, 124, 12, 245, 363 → shows the following path in the binary search tree:



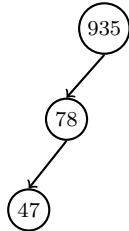
. At node 202, since we're looking for $363 > 202$ we should have went right. This sequence can therefore not be a search sequence of the key 363.

- 2, 309, 307, 19, 266, 382, 31, 278, 363 → shows the following path in the binary search tree:



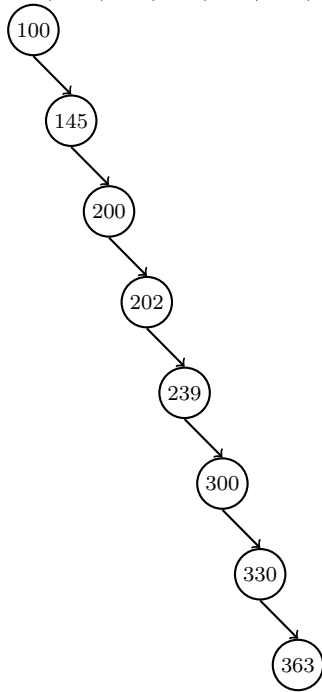
. At node 309, since we're looking for $363 > 309$ we should have went right. This sequence can therefore not be a search sequence of the key 363.

- 935, 78, 47, 21, 299, 392, 358, 363 \rightarrow shows the following path in the binary search tree:



. At node 78, since we're looking for $363 > 78$ we should have went right. This sequence can therefore not be a search sequence of the key 363.

- 100, 145, 200, 202, 239, 300, 330, 363 \rightarrow shows the following path in the binary search tree:



. This sequence can be a search sequence of the key 363.

Problem 10

Answer

1. The tree will be identical to the original tree. The reason is that:

- The insertion of a new key x into a BST corresponds to the creation of a new leaf,
- The deletion of an element placed in a leaf of a BST simply corresponds to the deletion of that leaf.

2. Insertion into a BST is not a commutative operation, i.e. $\text{INSERT}(y, \text{INSERT}(x, T)) \neq \text{INSERT}(x, \text{INSERT}(y, T))$. As a counter-example, consider the tree T : $\textcircled{5}$ and the elements x and y such that $x.\text{key} = 10$ and $y.\text{key} = 15$. We have that

