Small Group Tutorial 3, 20 - 24 March 2017 Solutions

Please note that all tasks marked as ”ADDITIONAL” will be done during the SGT session only if the rest of the exercises is done. Please try to do the remaining tasks at home. Answers will be published for all questions.

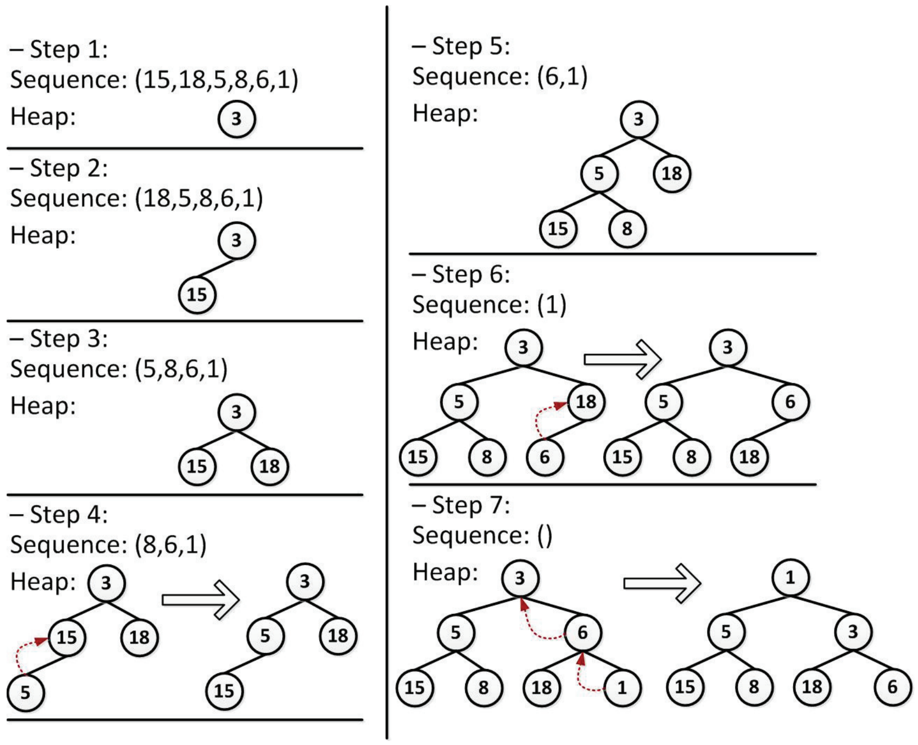
1. Illustrate the execution of the *heap–sort* algorithm on the following input sequence:

$$(3, 15, 18, 5, 8, 6, 1).$$

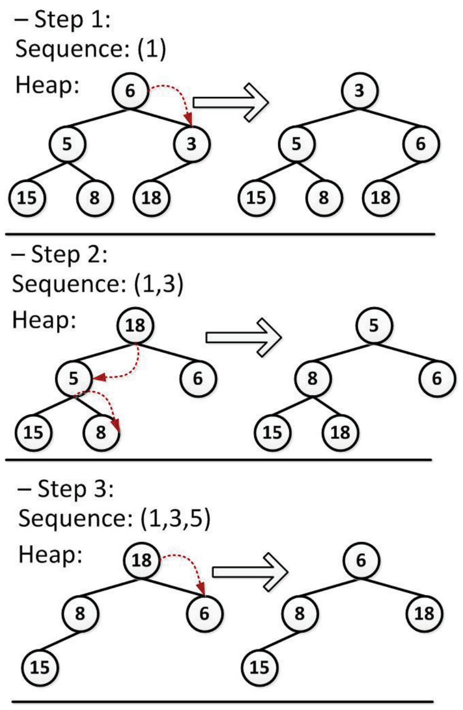
Consider both phases of the heap–sort: (i) insertion to the heap, and (ii) removing from the heap. Show how the sequence and heap look like after each step of each of the phase.

Answer

- Phase 1 – insertion the elements of the sequence into a heap.



- Phase 2 – removal of the elements from a heap and insertion them into a sequence.



2. (a) Insert into an empty binary search tree, entries with keys $(30, 40, 24, 58, 48, 26, 11, 13)$ (in this order). Show the final 8–element binary search tree (do not show the intermediate trees).

   (b) Present step by step the execution of method: $TreeSearch(58, root)$ on tree T.

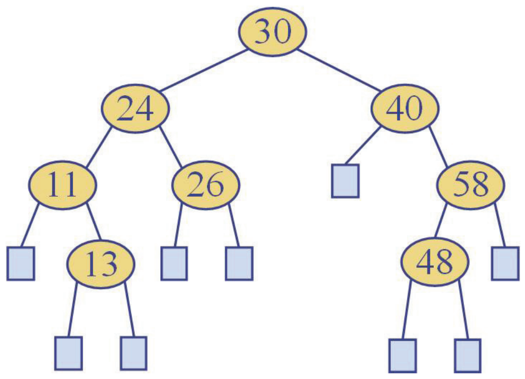
   (c) Present step by step the execution of operations:
   — removal of node with key 58 from tree *T*,
   — removal of node with key 40 from tree *T*.
   — ADDITIONAL – removal of node with key 30 from tree *T*.

Method *TreeSearch*(k,v) was presented during the Lecture 9. It is presented below:

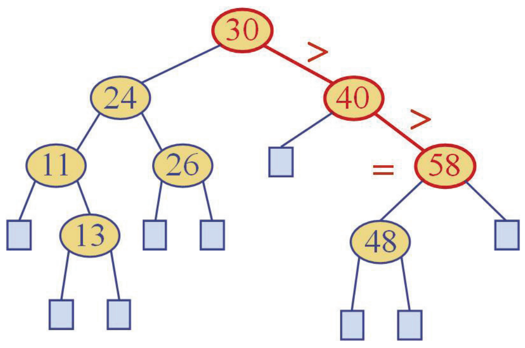
```
Algorithm TreeSearch(k,v)
    if isExternal (v)
        return v
    if k < key(v)
        return TreeSearch(k,left(v))
    else if k = key(v)
        return v
    else { k > key(v) }
        return TreeSearch(k,right(v))
```

Answer

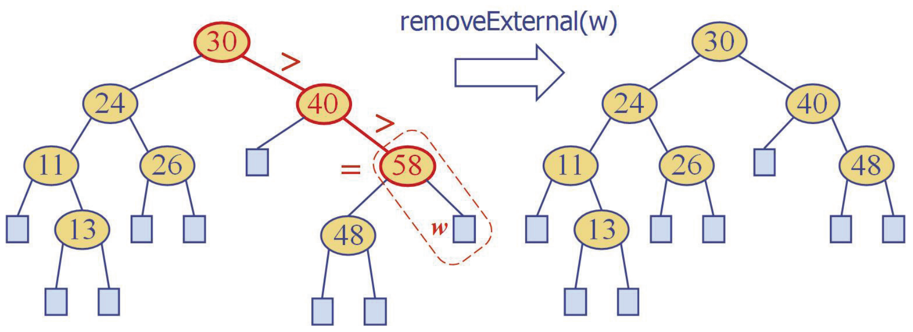
(a) Insert into an empty binary search tree, entries with keys $(30, 40, 24, 58, 48, 26, 11, 13)$ (in this order).



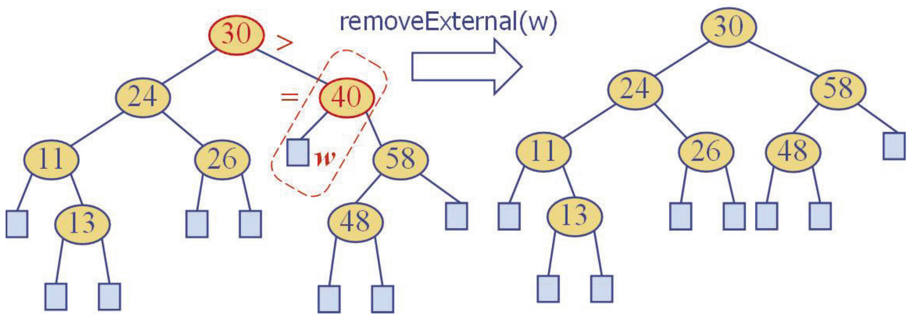
(b) Search for a node with key 58.



(c) — Removal of node with key 58 from tree T – To perform removal of key 58 from tree T, first we search for key 58. Note that the node that stores key 58 has a leaf child. Thus, we perform operation *removeExternal*(w), which removes leaf child w and its parent.



— Removal of node with key 40 from tree T – To perform removal of key 40 from tree T, first we search for key 40. Note that the node that stores key 40 has a leaf child. Thus, we perform operation *removeExternal*(w), which removes leaf child w and its parent. The process is presented in the figure below.



— Removal of node with key 30 from tree T.

i) Let's assume that key $k = 30$ is stored in node v.

ii) Find the internal node w that follows v in an inorder traversal.

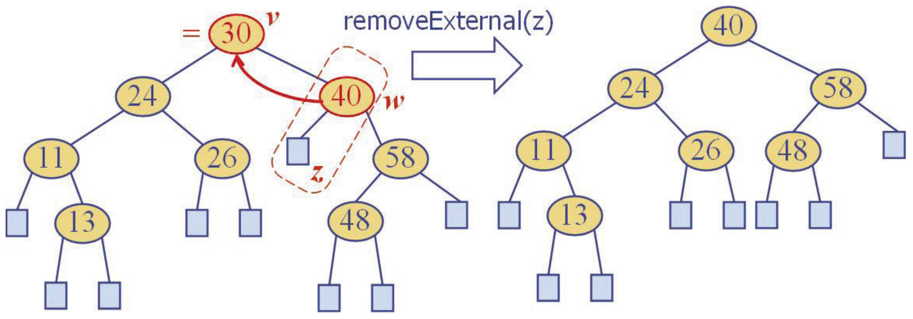
iii) Inorder traversal: 11 13 24 26 30 40 48 58.

iv) Node w that follows node v has key $k = 40$.

v) Copy $k = 40$ into node v.

vi) Remove node w and its left child z (which must be a leaf) by means of operation *removeExternal*(z).

The process is presented in the figure below.



3. Draw the 13–entry hash table that results from using a hash function:

$$h(i) = (i + 5) \mod 13,$$

to hash the keys:

$$5, 6, 11, 4, 1, 15, 14, 2, 17, 28, 24.$$

Assume that collisions are handled by:

(a) Double hashing using the following secondary hashing function: $h'(i) = 5 - (i \bmod 5)$.
(b) ADDITIONAL TASK — Linear probing.
(c) ADDITIONAL TASK — Separate chaining.

Answer

(a) **Double hashing**

— Double hashing uses a secondary hash function $h'(i)$.
— If h maps some key i to a cell $A[x]$, with $x = h(i)$, that is already occupied, then we iteratively try the buckets:

$$A[(x + f(j)) \bmod N] \quad \text{for } j = 1, 2, \cdots, N1$$

where $f(j) = j \cdot h'(i)$
— The secondary hash function $h'(i)$ cannot have zero values
— The table size N must be a prime to allow probing of all the cells
— Common choice of compression function for the secondary hash function:

$$h'(i) = q - (i \bmod q)$$

where: $q < N$ and q is a prime.

13–entry hash table:

0	1	2	3	4	5	6	7	8	9	10	11	12
2		28	11	24		1	15	14	4	5	6	17

The values of $h(i)$ and $h'(i)$ functions as well as the cells that have been probed:

i	$h(i)$	$h'(i)$	Probes	No. of probed buckets (cells in the array)
5	10	5	10	1
6	11	4	11	1
11	3	4	3	1
4	9	1	9	1
1	6	4	6	1
15	7	5	7	1
14	6	1	6|7|8	3
2	7	3	7|10|0	3
17	9	3	9|12	2
28	7	2	7|9|11|0|2	5
24	3	1	3|4	2

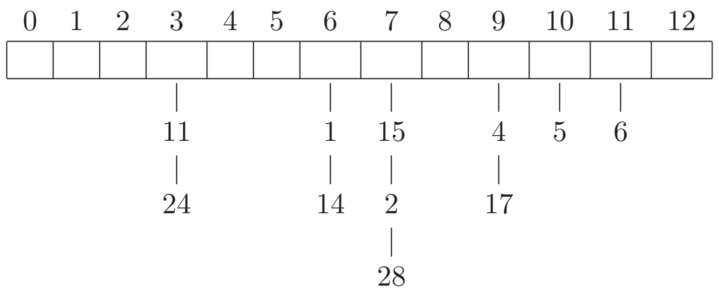
(b) **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell:
— If we try to insert an entry (i, v) into a cell $A[x]$ that is already occupied, where $x = h(i)$, then we try next at $A[(x + 1) mod N]$.
— If $A[(x + 1) mod N]$ is taken then we try $A[(x + 2) mod N]$, and so on, until we find an empty cell.

13–entry hash table:

0	1	2	3	4	5	6	7	8	9	10	11	12
17	28		11	24		1	15	14	4	5	6	2

(c) **Separate chaining:** let each cell in the table point to a linked list of entries that map there.

13–entry hash table:

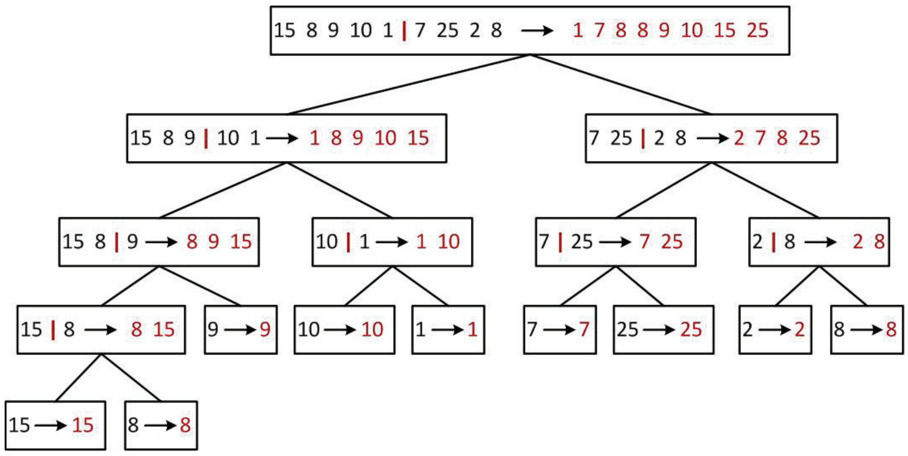


4. Illustrate the execution of the *merge–sort* algorithm on the following input sequence:

$$(15, 8, 9, 10, 1, 7, 25, 2, 8).$$

Present an execution of merge–sort as a binary tree.

Answer

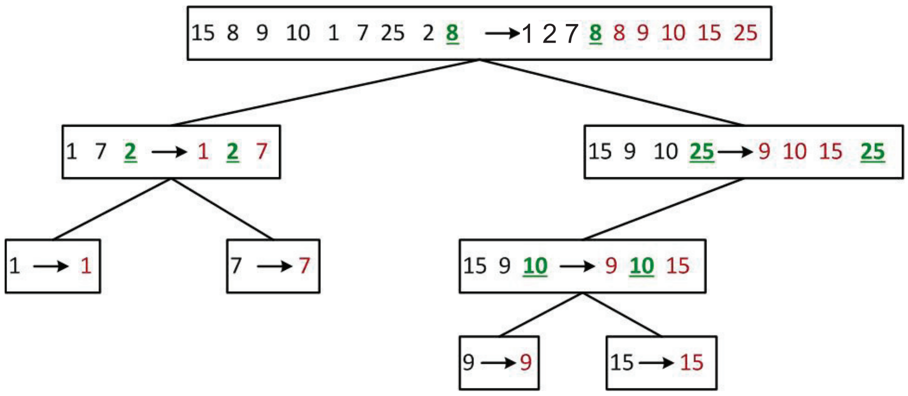


5. Illustrate the execution of the *quick–sort* algorithm on the following input sequence:

$$(15, 8, 9, 10, 1, 7, 25, 2, 8).$$

Present an execution of quick–sort as a binary tree. Choose the pivot to be the last element in the sequence.

<u>Answer</u>



6. Sort the following sequence using *bucket–sort* method:

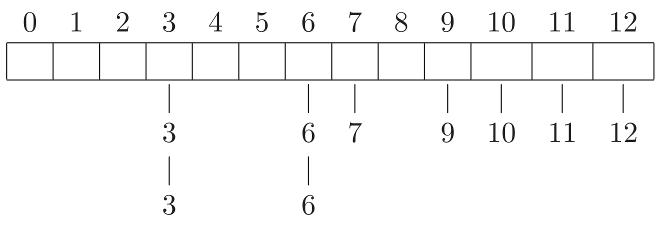
$$(3, 7, 11, 10, 6, 3, 6, 9, 12).$$

Present both phases of bucket–sort: (i) phase 1 – move entries from a sequence into appropriate bucket in an array, (ii) phase 2 – move entries from buckets to the final sequence.

Answer

Phase 1

Sequence — $(3, 7, 11, 10, 6, 3, 6, 9, 12)$.
Array with buckets:



Phase 2 Sequence — $(3, 3, 6, 6, 7, 9, 10, 11, 12)$.

ADDITIONAL EXERCISES:

7. Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row of A, all the 1's come before any 0's in that row. Assuming A is already in memory, describe a method running $O(n\text{log}n)$ time (not $o(n^2)$ time!) for counting the number of 1's in A.

   Answer

   To count the number of 1s in A, we can do a binary search on each row of A to determine the position of the last 1 in that row. Then we can simply sum up these values to obtain the total number of 1s in A. This takes $O(\log n)$ time to find the last 1 in each row. Done for each of the n rows, then this takes $O(n \log n)$ time.

8. An airport is developing a computer simulation of air–traffic control that handles events such as landing and takeoffs. Each event has a time–stamp that denotes the time when the event occurs. The simulation program needs to efficiently perform the following two fundamental operations:

   - Insert an event with a given time–stamp (that is, add a future event).
   - Extract the event with the smallest time–stamp (that is, determine the next event to process).

   Which data structure should be used for the above operations? Why?

   Answer

   The best data structure for this air–traffic control simulation is *a priority queue.* The priority queue will enable the handling of the time–stamps and keep the events in order so that the event with the smallest time–stamp is extracted easily.

9. Explain why during the lecture 7, the case where node r has a right child but not a left child was not considered in the description of the down–heap bubbling.

   Answer

   Since a heap is a complete binary tree, the levels of the heap are filled left to right. Thus, a node with the left child nay not have the right child. However, if a node has the right child, it must also have the left child.

10. How many *different binary search trees* can store the keys $1, 2, 3, 4$? Justify your answer.

    Answer

    14 (five with '1' as root, two with '2' as root, two with '3' as root, and five with '4' as root).