Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127, India
An Autonomous Institute under MHRD, Govt of India
An Institute of National Importance
COM 501 Advanced Data Structures and Algorithms - Lecture Notes

Instructor
N.Sadagopan
Scribe:
P.Renjith

Advanced Data Structures : Min-Max Heap and Deaps

`Objective:` The focus of this lecture is to discuss efficient algorithms for finding simultaneous minimum and maximum.
We below summarize the complexity of finding min() and max() in well-known data structures.

ADT	Insert	Find Min	Find Max	Search
Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Sorted Array	$O(n)$	$O(1)$	$O(1)$	$O(\log n)$
Linked List	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Sorted Linked List	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Min heap	$O(\log n)$	$O(1)$	$O(n)$	$O(n)$
Max heap	$O(\log n)$	$O(n)$	$O(1)$	$O(n)$

A closely related operation is extract-min() and extract-max() which when performed on an n-element min-heap takes $O(\log n)$ and $O(n)$, respectively. It is natural to ask: can we perform find-min() and find-max() in constant time and extract-max() and extract-min() in $O(\log n)$ time. If we use classical min-heap (max-heap), one of the find() operations is a constant time operation and one of the extract() operations is a logarithmic time operation, whereas the other one runs in linear time.
Another approach is to maintain both min-heap and max-heap on the same data set and use min-heap to perform find-min() and extract-min() operations. Similarly, use max-heap to perform find-max() and extract-max() operations. However, to maintain consistency among data, when extract-min() is performed on the min-heap, the same operation must be performed on max-heap as well, which is costly and incurs $O(n)$ effort. We shall next see a variant of min and max heaps, namely min-max-heap which achieves our goals.

Min-max Heap

Min-max heap is a heap with the additional property that the levels are alternately labeled with MIN and MAX. In particular, the root is at min level and its children are at max level, further its children are at min level and so on. Also, an element v in a max level has the property that v is the largest of all the elements in the subtree rooted at v. Similarly, if v is in min level, then v is the smallest of all the elements in the subtree rooted at v. These two properties we refer to as max, and min property, respectively. An example is illustrated in Figure 1.

`Creation of min-max heap: a top-down approach`

This is similar to creation of min-heap in top-down fashion. In top-down approach each new element is inserted at the last level respecting nearly complete binary property and the min-max heapify procedure bubbles up the element to its correct position by recursively checking the min or max property starting from its parent or grand parent and this check is done recursively till the parent or grand parent is the root node. For creation of an n-element min-max heap we need

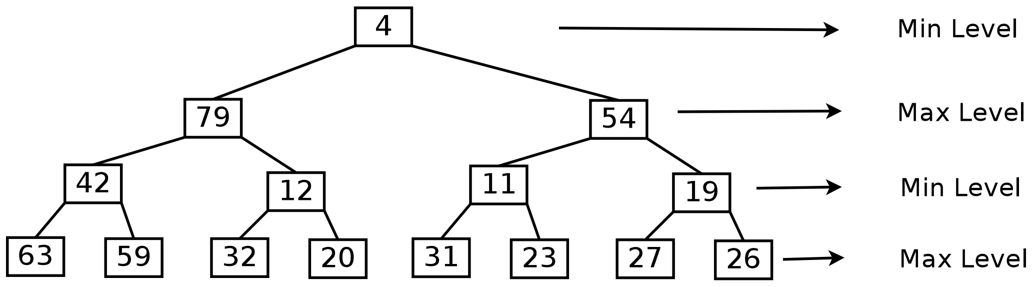


Fig. 1. Min-Max heap - an example

$\sum_{i=1}^{n} \log i = O(n \log n)$. On the other hand, we can also construct the min-max heap in bottom-up fashion in linear time. This construction is also similar to min-heap construction in bottom-up fashion. In this method, the construction is a two step process. We first construct a heap using the given elements and in Step 2, we convert the constructed heap into a min-max heap. While performing such a min-max heap, we check for min and max properties at each element from the last non-leaf element till the root. Here we show an example which creates the min-max heap using the top-down approach.

Example: Create the min-max heap with the elements $(1, 4, -5, 6, 7, 9, 3, 40, 20, 2, 70, 100)$

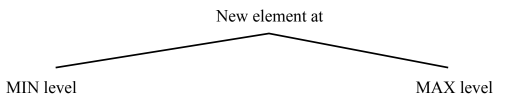
Figures $1 - 18$ illustrate the creation of a min-max heap. Min property violation of Figure 2 is set right in Figure 3. Similarly, max property violation occurs in Figures $4, 6, 8, 11, 16, 17$ and min property is violated in Figure 14.

Insertion in a min-max heap

Insertion of an element in a min-max heap is similar to a heap where we insert an element as a left child of the first leaf node (indexing nodes from top to bottom and left to right) or right child of the last non-leaf. After insertion we set right the min-max heap by recursively checking the min-max heap property starting from the inserted element till the root. Depending on the level in which the new element is inserted, it can violate either min or max property.

For example, if we are inserting an element 80 in Figure 2, then it will be first inserted as the left of the element 63. Note that 80 is now in min level. To set right the heap, we first observe that the max property at the parent element 63 is violated, and we exchange 63 and 80 (See Figure 2 (a)). Note that there is no need to check the min property violation with respect to any min elements above 63 (starting from 42).

We further check max property of grand parent of 80, which is 79, see Figure 2 (b). Max property at 79 is violated, and therefore, exchange 79 and 80. This completes the insertion process and we finally obtain a min-max heap with 80 inserted as shown in Figure 2 (c). Note that in the insertion, either the min property is violated or the max property but not both and therefore we may need

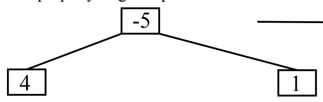


if value(parent)<newnode, then the max property of parent is violated

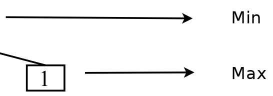
if value(grand parent)>newnode, then the min property of grand parent is violated

if value(parent)>newnode, then the min property of parent is violated

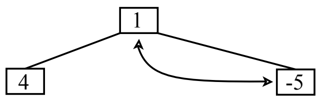
if value(grand parent)<newnode, then the max property of grand parent is violated



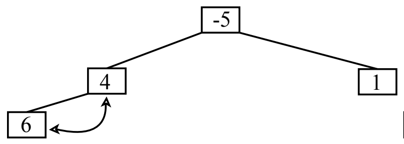
(1)



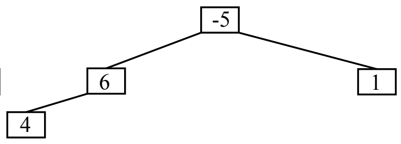
(2)



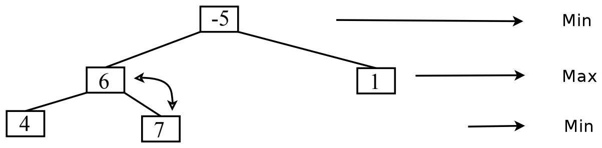
(3)



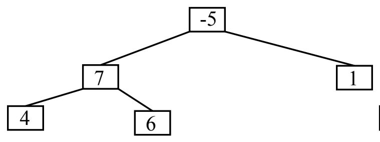
(4)



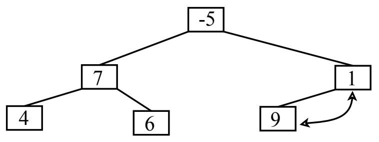
(5)



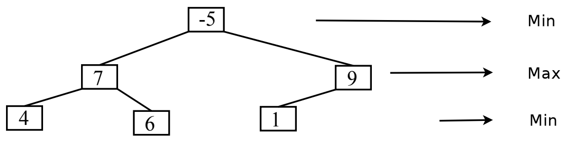
(6)



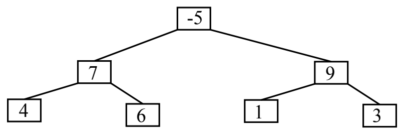
(7)



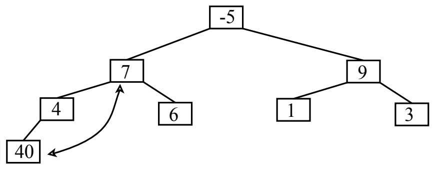
(8)



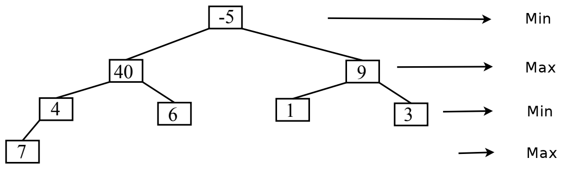
(9)



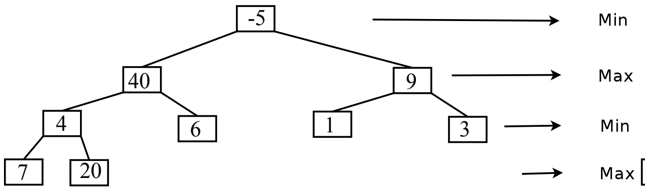
(10)



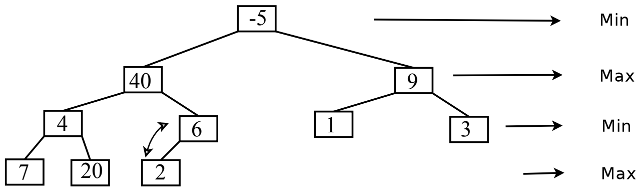
(11)



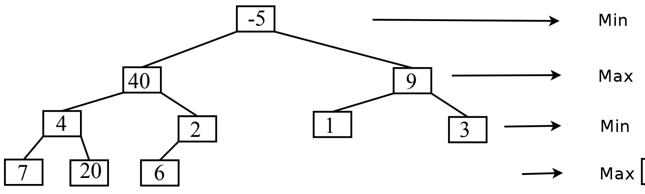
(12)



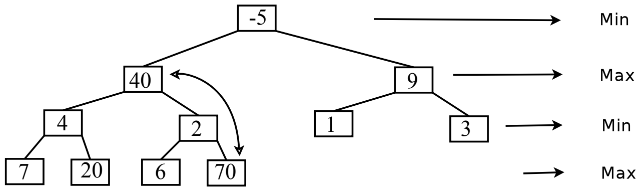
(13)



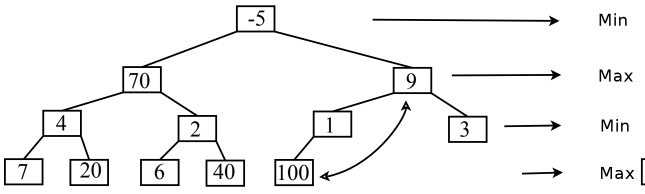
(14)



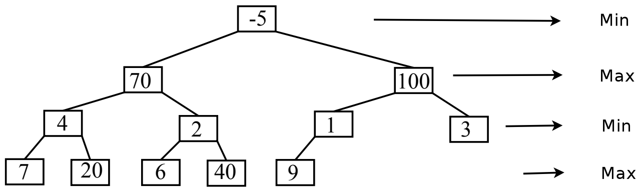
(15)



(16)



(17)



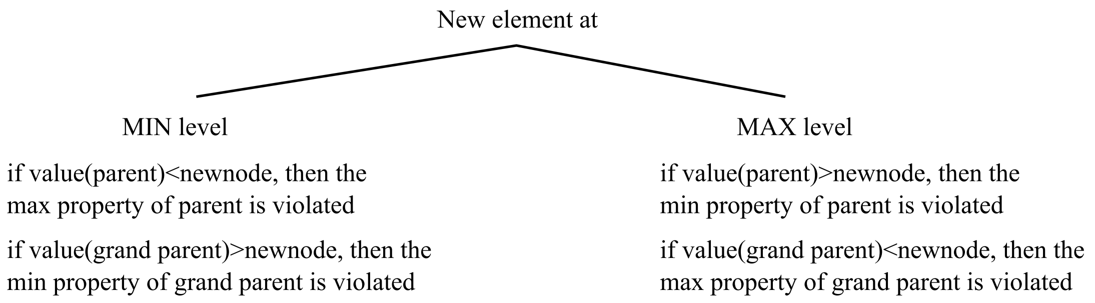
(18)

to check one among them recursively till the root. Moreover, we need $O(\log n)$ effort for inserting an element in a min-max heap.

Extract Min and Max in a min-max heap

For extract min in a min-max heap we replace the root element with the last leaf (the last element in the last level), henceforth referred to as *key*. Note that this replace operation ensures heap property is maintained, however, there may be violation of min-max property. Further we may need to set right the min-max heap property. Depending on whether the key is in min-level or max-level, or left subtree or right subtree of the root, to bring back the min-max heap property, we recursively do the following checks.

1. Compare the key with second minimum which is one of the four grand children. If key is larger than the second min, swap key and second min.
2. While swapping key and second min, violation may happen with respect to max property. i.e., key is greater than its parent which is at the max level. If so, swap key and its parent after performing the above swap.
3. Recursively do these checks until no longer possible.



New element at

MIN level

if value(parent)<newnode, then the max property of parent is violated

if value(grand parent)>newnode, then the min property of grand parent is violated

MAX level

if value(parent)>newnode, then the min property of parent is violated

if value(grand parent)<newnode, then the max property of grand parent is violated

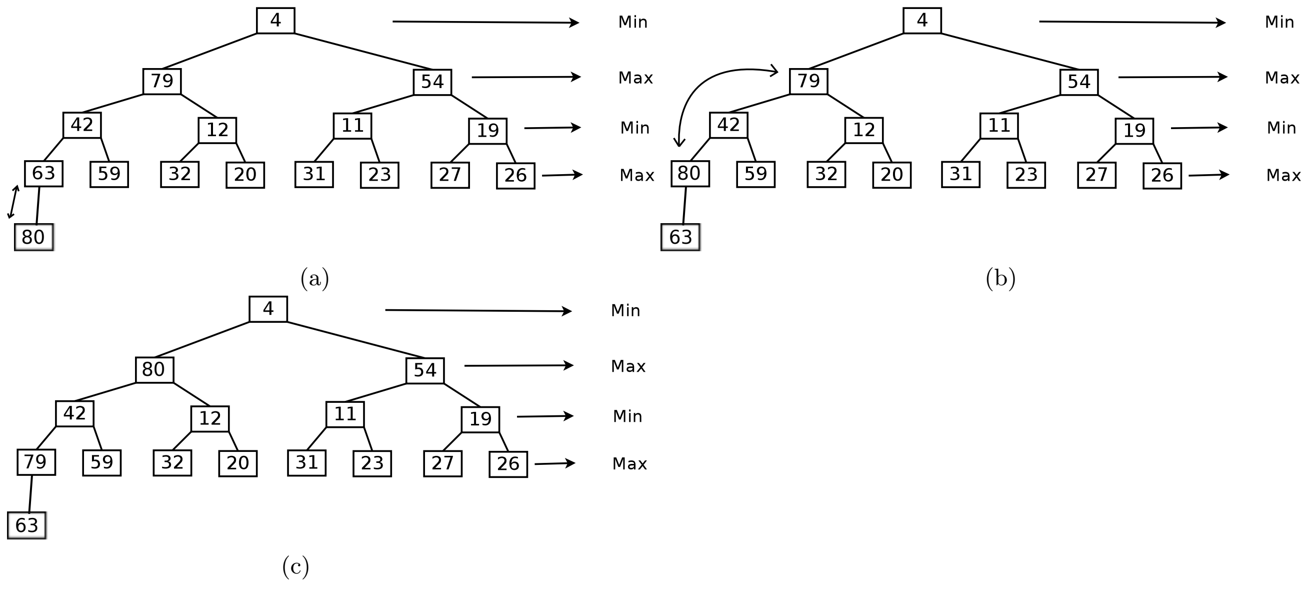


Fig. 2. Insertion of element 80

In Figure 3, for extract min operation, the element 10 in the root is replaced with 33. Then the root is exchanged with 20, the second minimum which is shown in Figure 3 (c). Now since there is no max violation at the element 40. As part of recursive check, 33 is swapped with its second minimum grand child 25. Now, we notice a max violation and hence swap 33 and 30 and this completes the extract min procedure. Similar approach works for extract max.

Extract min (or max) need $O(1)$ time for replacing the min or max element with the last leaf element and at most $\frac{l}{2}+1$ exchanges, where l is the number of levels in the n element min-max heap. Since $l = \theta(\log n)$, extract min (or max) can be performed in $\theta(\log n)$ time.

We shall present another data structure Deaps which also performs extract min and extract max in $\theta(\log n)$ time.

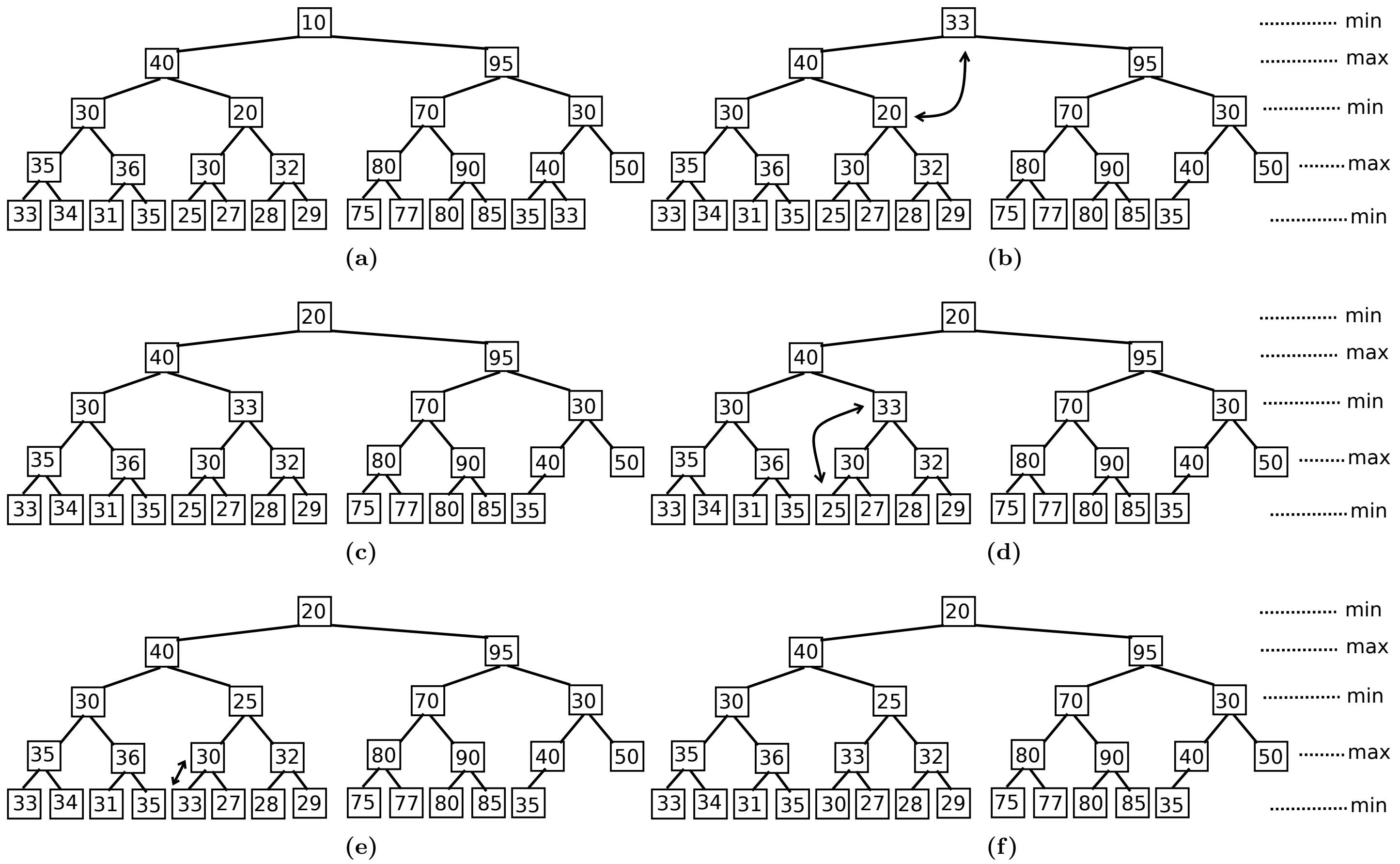


Fig. 3. Extract Min - An example

Deaps

Deaps is an almost complete binary tree with the root being a null node and the left subtree is a min heap and the right subtree is a max heap. We associate index with each element, with root being 0, its left child (min element) gets 1 and its right child gets 2, and so on. For every element we also define the *corresponding element*. For an element with label x at level l in the min heap of a deap, the corresponding element in the max heap, denoted as $corr(x)$,

$$corr(x) = \begin{cases} x + 2^{l-1} & \text{if an element exists at this index} \\ \lceil \frac{x}{2} \rceil - 1 + 2^{l-2} & \text{otherwise} \end{cases}$$

We arrange elements in an array such that for an element at position x, children of x is seen at $2x$ and $2x+1$. For an element at position x and level l, if the corresponding element at $x + 2^{l-1}$ is not present (does not exist), then $corr(x) = corr(parent(x)) = corr(\lceil \frac{x}{2} \rceil - 1) = \lceil \frac{x}{2} \rceil - 1 + 2^{l-2}$. All the elements in the deap respects the *correspondence property* which ensures that value of element at index x is less than or equal to the value of its corresponding element. That is, $val(x) \leq val(corr(x))$, where $val(x)$ is the value of element at position x. For an element with label y at level l in the max-heap of a deap, its corresponding element can be found at
$corr(y) = y - 2^{l-1}$

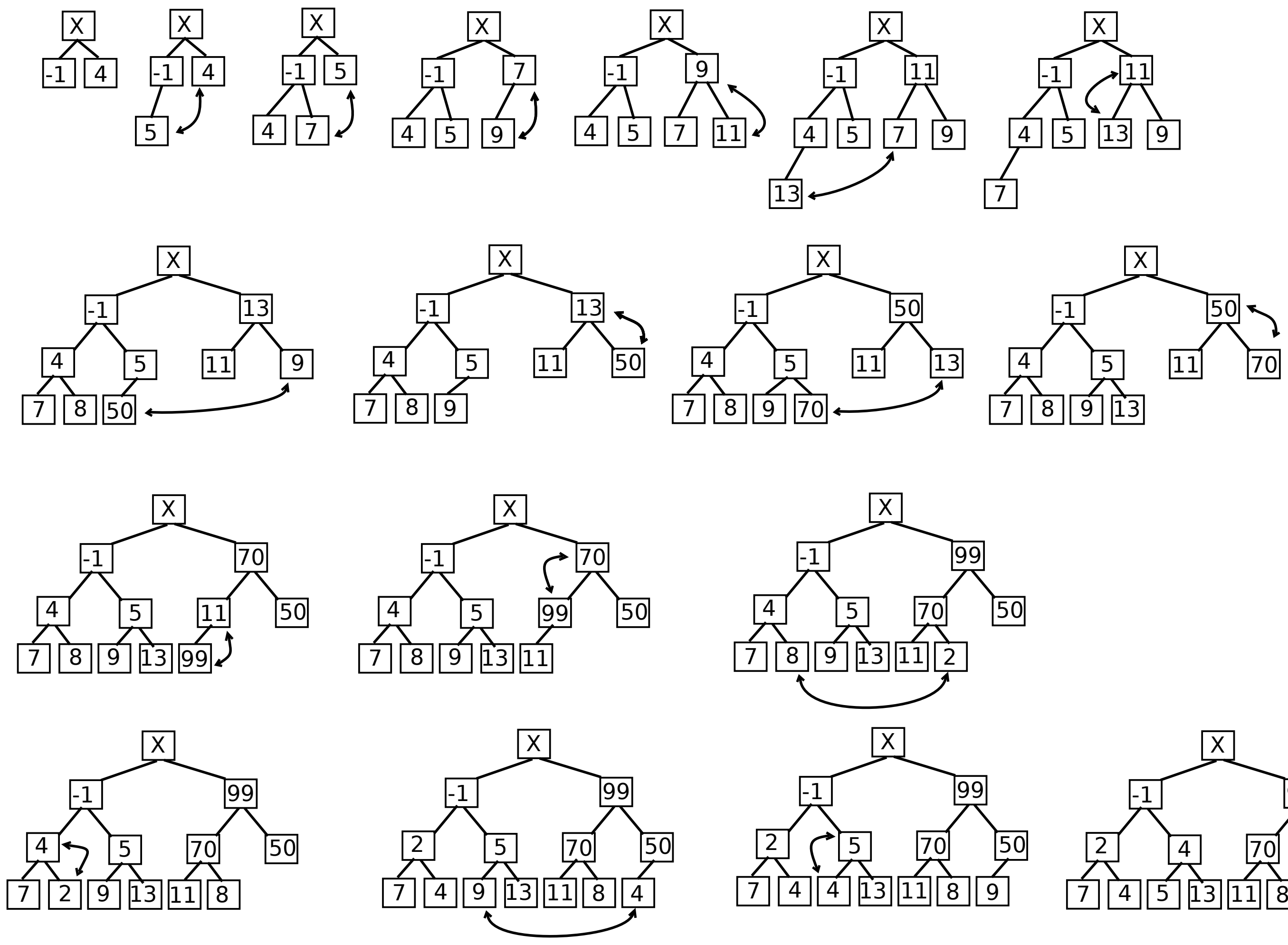


Fig. 4. A Deap on elements $(-1, 4, 5, 7, 9, 11, 13, 8, 50, 70, 100, 2, 4)$

Insertion in Deaps

Similar to min-max heap, Deap can be constructed in top-down fashion starting with a single node and iteratively adding further elements by checking necessary properties. An element is always inserted in the last position. That is, element is inserted either as right child of the last non leaf or as the left child of the first leaf.

- Every time we insert an element, we first check the correspondence property; suppose the new element is inserted on the min-heap side with no correspondence property violation, then there may be a violation of min-heap property with respect to the new element.
- In such a case the newly inserted element is checked with its parent recursively to set right the min heap property.
- On the other hand, if there is a correspondence property violation, then we exchange the element with its corresponding element, and set right the max heap property. Similar check is done if the new element is inserted on the max heap end.

– That is, when a new element is inserted, the following can take place;
  1. Insertion creates a violation of correspondence property only
  2. Correspondence property violation followed by a call to max-heapify routine
  3. Correspondence property violation followed by a call to min-heapify routine
  4. No violation of correspondence property but a violation of max-heapify property
  5. No violation of correspondence property but a violation of min-heapify property

An example is illustrated in Figure 4. Similar to the min-max heap, creation of an n-element deap require $\sum_{i=1}^{n} \log i = O(n \log n)$ time.

Extract min and max in Deaps
As part of extract min operation, we replace the min node (left child of the root) with the last leaf, henceforth referred to as *key*. Then we set right the Deap by doing the following three steps.

1. Recursively check whether the min-heap property is violated. If so, then exchange the key with its child. When recursion stops, proceed to the next step.
2. Check the correspondence property and do exchange if necessary. If there is a exchange, there may be max-heap violation, proceed to the next step;
3. If correspondence property is violated and the key gets exchanged to the corresponding element on the max heap side, then recursively check the key with its parent and exchange if necessary. This is to set right the max-heap property.

Clearly, the above three steps incur $O(\log n)$ time. Extract max operation is symmetric and incurs $O(\log n)$. An example is illustrated in the Figure 5.

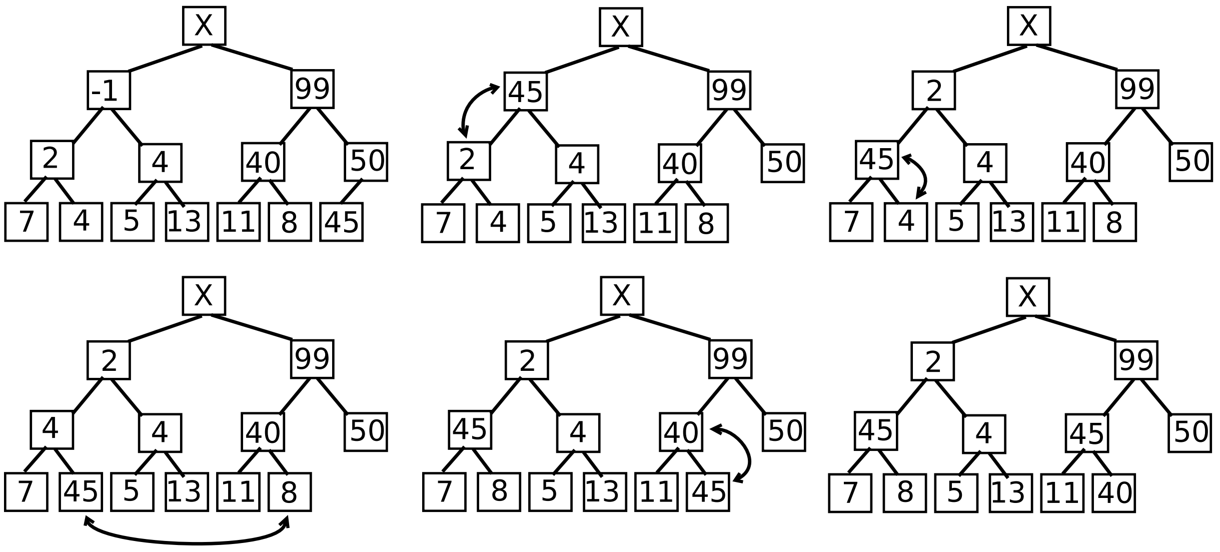


Fig. 5. Extract min in a Deap - an illustration

Exercises

1. Give an example of a min-heap such that there exist an element x in level i, y in level j, $i < j$ and $x > y$ in the min-heap.
2. How to create a **max-min** heap in linear time?
3. Is it possible to construct an n-element Deap in $O(n)$ time?
4. Justify true or false. For a Deap, there exist an element x in min-heap at level i, y in max heap at level j, $i > j$ and $x > y$.
5. Construct a Deap having l levels (root is at level 0) such that when extract max operation is initiated, after replacing the max element with the last leaf, $2\lfloor \frac{l}{2} \rfloor - 1$ exchanges are required to set right the deap. Does there exist a deap with more than the $2\lfloor \frac{l}{2} \rfloor - 1$ exchanges? Give a proper justification.

Acknowledgements: Lecture contents presented in this module and subsequent modules are based on the text books mentioned at the reference and most importantly, author has greatly learnt from lectures by algorithm exponents affiliated to IIT Madras/IMSc; Prof C. Pandu Rangan, Prof N.S.Narayanaswamy, Prof Venkatesh Raman, and Prof Anurag Mittal. Author sincerely acknowledges all of them. Special thanks to Teaching Assistants Mr.Renjith.P and Ms.Dhanalakshmi.S for their sincere and dedicated effort and making this scribe possible. Author has benefited a lot by teaching this course to senior undergraduate students and junior undergraduate students who have also contributed to this scribe in many ways. Author sincerely thank all of them.

References:
1. E.Horowitz, S.Sahni, S.Rajasekaran, Fundamentals of Computer Algorithms, Galgotia Publications.
2. T.H. Cormen, C.E. Leiserson, R.L.Rivest, C.Stein, Introduction to Algorithms, PHI.
3. Sara Baase, A.V.Gelder, Computer Algorithms, Pearson.
4. S.Sahni, Handbook of Data Structures.