Double-ended Priority Queue - Diamond Min-Max-heap Algorithm

Won-Sook Lee
School of Electrical Engineering and Computer Science
University of Ottawa, ON, Canada
wslee@uottawa.ca

Abstract

A heap is a traditional data structure with advantages being an almost complete tree allowing efficient removal of maximum or minimum as a typical example of a priority queue. We propose a double-ended priority queue where min-heap (up-right) and max-heap (flipped upside down) are stacked together in a diamond shape. The numbers on paths from the root of the min-heap to the root of the max-heap are in non-decreasing order and all paths share the same min number and the same max number. The numbers in the middle of the paths show distribution of median numbers. Differently from previously available min-max-heaps, this new min-max-heap inherits all properties of traditional min-heap and max-heap with generalized operations of put(x), removeMin(), removeMax() and bottomUpConstruction() with same Big-Oh complexity. Traditional upheap and downheap methods which swap elements to correct total-order relation between parent-child nodes are generalized to operate through external nodes allowing travelling between min-heap and max-heap.

Keywords: Min-max heap, double ended priority queue, removeMin, removeMax, bottom up construction

1 Introduction

The heap is one of the most efficient abstract data type called a priority queue. In a heap, the minimum (or maximum) number, i.e. highest (or lowest) priority element, is always stored at the root of a min-heap binary trees. Even though a heap is not a sorted structure, paths on the tree shows elements in a sorted manner. Three main operations are insertItem, removeMin and removeMax.

When a heap is a complete binary tree, A heap allows easy conversion to an array structure from the tree while having parent and children nodes to be accessed easily by array index calculation, we often use in-place implementation avoiding extra memory and structures in practice. It also makes the tree to have a smallest possible height. A heap with N nodes has height of log N.

A min-max heap is an example of double-ended priority queue

a min-max heap is a complete binary tree data structure which combines the usefulness of both a min-heap and a max-heap, that is, it provides constant time retrieval and logarithmic time

removal of both the minimum and maximum elements This makes the min-max heap a very useful data structure to implement a double-ended priority queue. Like binary min-heaps and max-heaps, min-max heaps support logarithmic insertion and deletion and can be built in linear time. Min-max heaps are often represented implicitly in an array; hence it's referred to as an implicit data structure.

2 Diamond min-max-heap structure

This is a double-ended priority queue for which you can obtain both the max and the min element using Heap data structure.

This double-ended queue is composed of two heaps and a buffer: a min-heap and a max-heap. Half of the elements are stored in one of the heaps and the other half is stored in the second heap. Each external element **a** of the min-heap is associated with an external element of the max-heap **b** with **a <= b** as shown in red lines in Figure 1. i.e. two associated nodes have the same index when max-heap and min-heap are stored in arrays. If the total number of elements is odd, then the extra element is contained in a 1-element buffer.

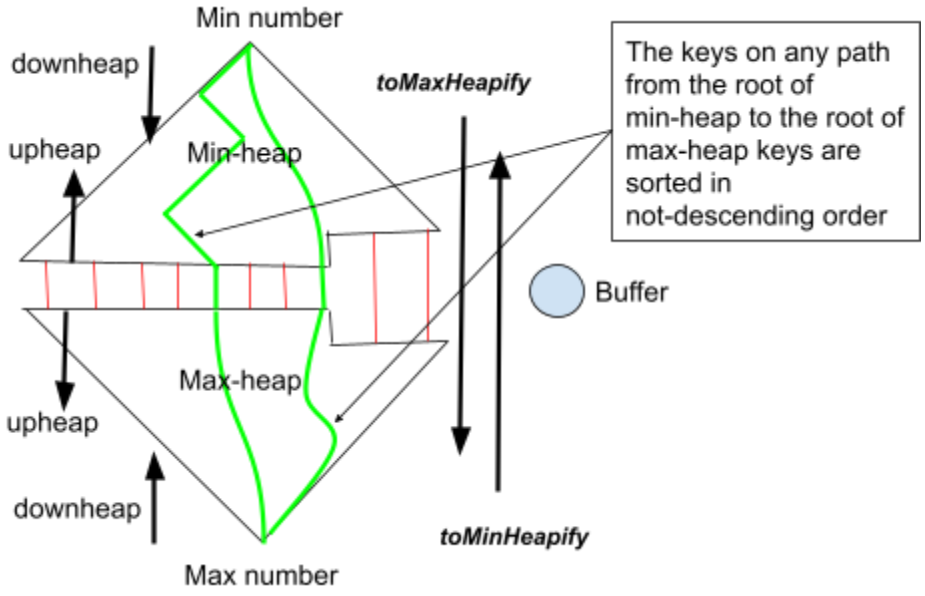
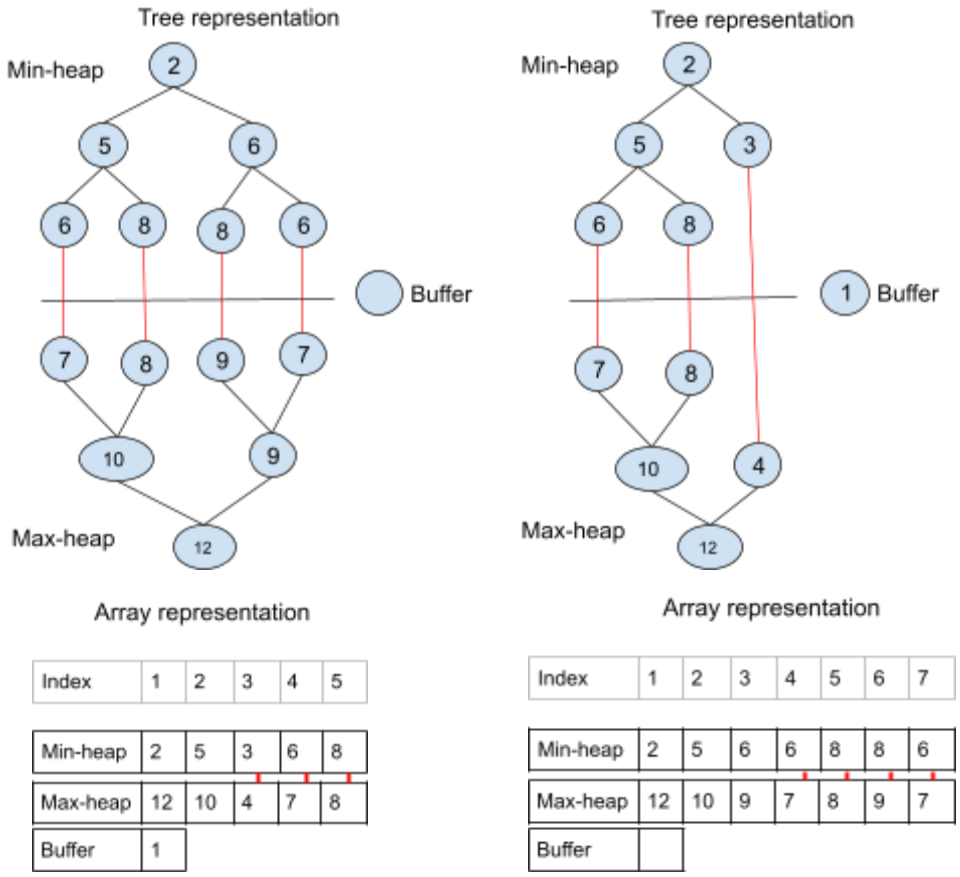


Figure 1: Min-max-heap structure. The external nodes in a min-heap and a max-heap are connected as shown in red lines. If we follow any path from the root of the min-heap to the root of the max-heap, the key values are non-descending order.



(a) (b)

Figure 2: ***Two examples with tree representation and array representation. (a) example with 11 elements and (b) example with 14 elements***

2.1 Review of Min-heap and Max-heap

- InsertElement(x) has complexity of O(log n) and the steps as follows:
  - Insert x as the last element and then upheap the element to the root until swapping is not required, i.e. the comparison between a parent and a child satisfies the heap definition.
- RemoveMin() and RemoveMax() has complexity of O(log n) and the steps as follows:
  - Remove element in the root position and move the last element to root position then downheap the element following a path toward an external node until swapping is not required.
- bottomUpConstruction() has complexity of O(n) and the steps as follows:
  - Backward starting from the last internal node to the root node, perform downheap toward an external node until swapping is not required.

As a summary, upheap means for last external element to move toward a root and downheap is for any element in any level toward an external node.

2.2 Diamond min-max-heap

Heapify is an operation to create a heap out of given array of elements. For this diamond heap, the min-heap and max-heap are connected through their external nodes and downheap process in one heap continues to another heap.
In addition, upheap can start from any external node and swap elements, if needed, until it reaches a given level.

We define one concept and three operations.

- **Associated**

If an external element in a min-heap and an external element in a max-heap share the same index in the array representation, they are associated.

- **SwapExternal**

If the external element in the min-heap is bigger than its associate element in the max-heap, swap them.

- **toMaxHeapify** (an element x, level i)

We apply downheap operation in min-heap for x and it is connected to upheap operation in the max-heap up to a level i through possible external element swap using swapExternal operation. The final level for upheap in max-heap is level i . The default level i is the level of the root node in the max-heap.

- **toMinHeapify** (an element x, level i)

We apply downheap operation in max-heap for x and it is connected to upheap operation in the min-heap up to a level i through possible external element swap using swapExternal operation. The final level for upheap in min-heap is level i . The default level i is the level of the root node in the min-heap.

The toMaxHeapify operation is shown in Figure 3. Call the element in min-heap as (A). Apply downheap for (A) in the min-heap. Check if new index of the element (A) is external after downheap. If it is external, compare with the associated element (B) with the same index in max-heap. If (A) bigger than (B), swap them and apply upheap for (A) in the max-heap to level i.

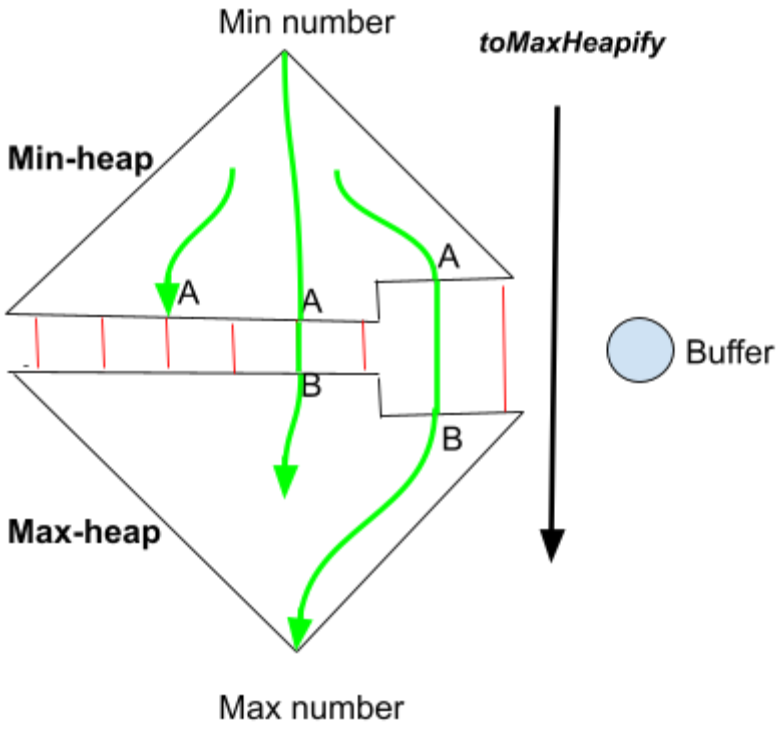


Figure 3: toMaxHeapify - When an element in min-heap got a new value, we perform downheap in the min-heap and possibly continue in the max-heap using upheap through swapExternal.

3 Insert an Element

Adding a new element is done as follows:

- If the buffer is empty, then insert the new element in it.
- Else if the buffer contains already an element, we have two elements to insert to the min-max-heap.
  1. Compare the new element and the element in the buffer. Insert a smaller element into the min-heap as a last element and insert a bigger element into the max-heap as a last element. Empty buffer.
  2. Heapify
     2.1. For the last element in min-heap, run toMinHeapify, i.e. upheap in the min-heap.
     2.2. For the last element in max-heap, run toMinHeapify (see Figure 5(b))
     2.3. For a last element in max-heap, run toMaxHeapify, i.e. upheap in the max-heap
     2.4. For a last element in min-heap, run toMaxHeapify (see Figure 5(c)

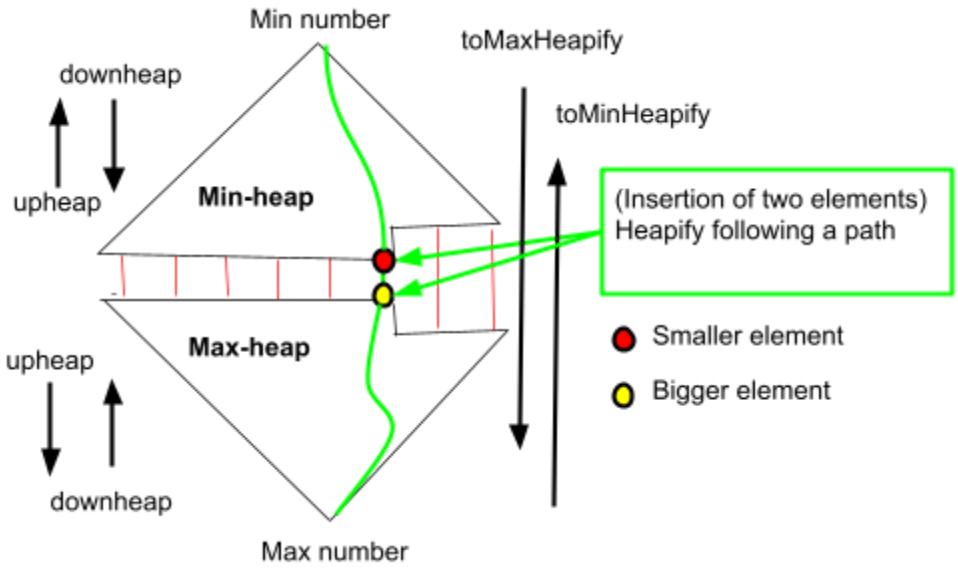
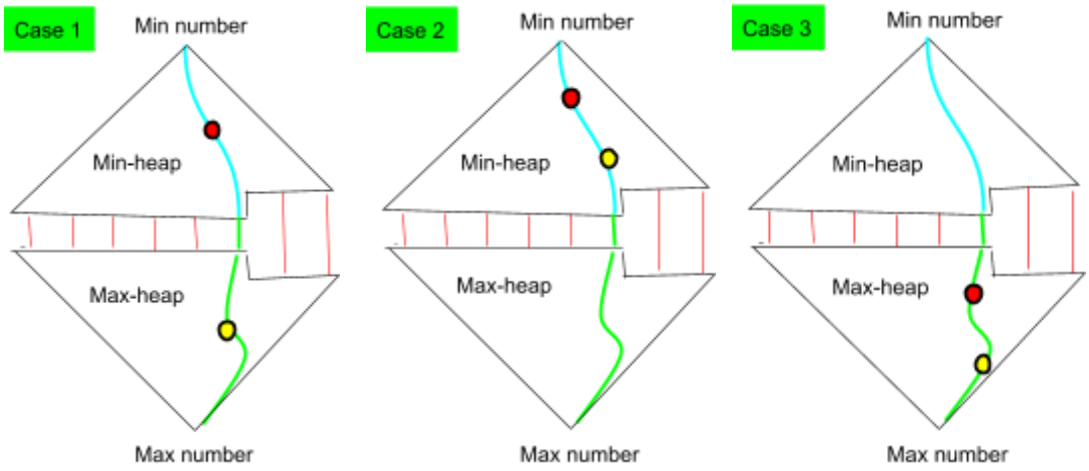


Figure 4: When there are two elements to insert to min-max-heap tree (one from the buffer and one new element given for insertion), we compare them and insert a smaller to the min-heap as the last element of it and insert a bigger to the max-heap as the last element of it. The process to heapify (such as downheap and upheap) is to decide which positions of these two elements in the path of sorted elements.



(a) (b) (c) .

Figure 5: When an element is inserted either to the min-heap or to the max-heap, the relation of parent and child is destroyed so we need to restore it to be a heap by doing heapify. Three cases after heapify; (Case 1) one belongs to min-heap and the other belongs to max-heap (Case 2) both belong to min-heap. (Case 3) both belong to max-heap).

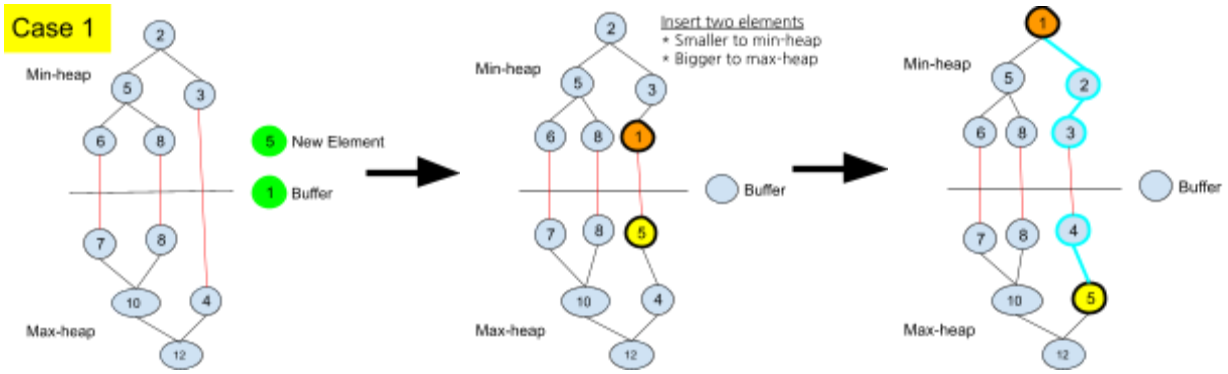


Figure 6: An example of case 1 where the element inserted in the min-heap is settled in the min-heap and the element inserted in the max-heap is settled in the max-heap after heapify

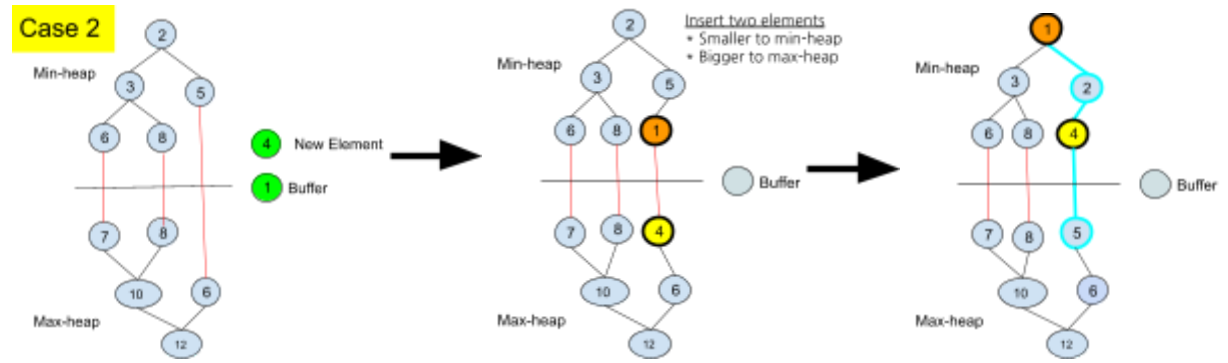


Figure 7: An example of case 2 where both the element inserted in the min-heap and the element inserted in the max-heap settled in the min-heap after heapify

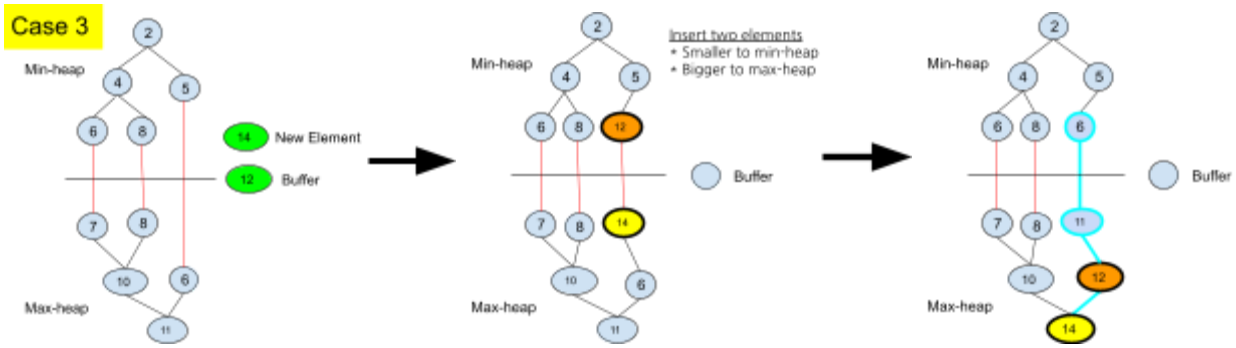


Figure 8: An example of case 2 where both the element inserted in the min-heap and the element inserted in the max-heap settled in the max-heap after heapify

4 Remove a Minimum Element

Removing the min element is done as follows:

1. (Case 1) If buffer is not empty and the element in the buffer is smaller than the element at the root of min-heap, then simply return the element in the buffer and empty the buffer.

2. Else you remove the element at the root of the min-heap and then
   2.1. (Case 2) If the buffer is not empty, then
      2.1.1. move the element in the buffer to the root of min-heap.
      2.1.2. Empty buffer.
   2.2. (Case 3) Else if the buffer is empty, then
      2.2.1. Move a last element of of min-heap to its root.
      2.2.2. Move a last element of of max-heap to the buffer.
   2.3. (Case 2 and 3) Apply toMaxHeapify to the element in the root of the min-heap.

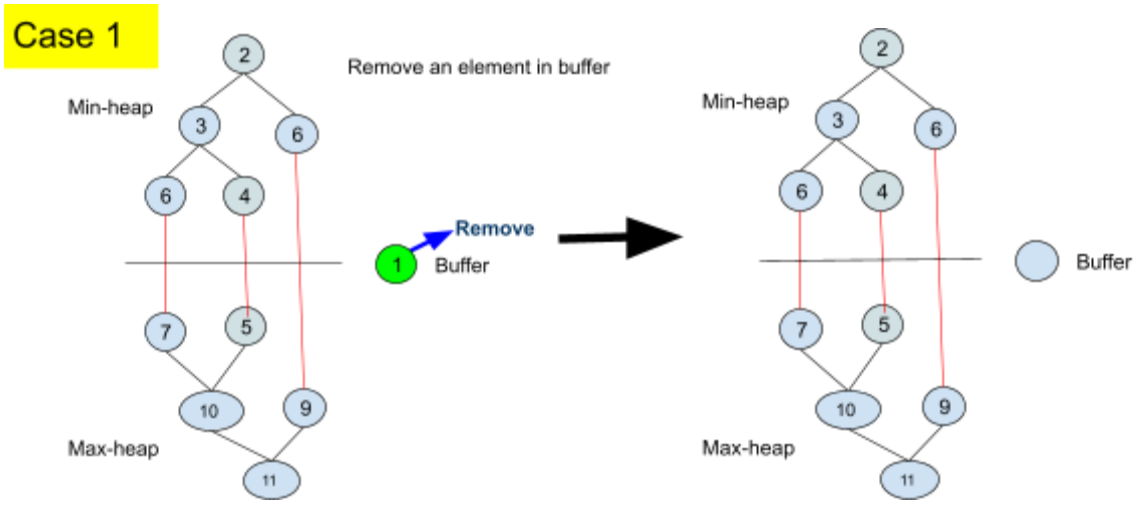


Figure 9: RemoveMin Case 1 where the element in the buffer is smaller than the root element in the min-heap, so we just return the element in the buffer and empty the buffer

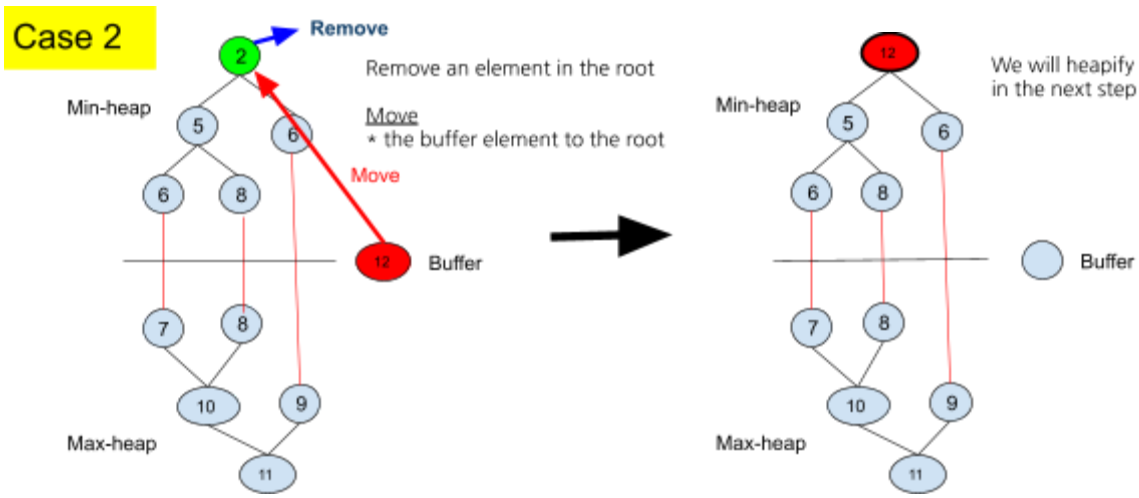


Figure 11: RemoveMin Case 2 where we remove the root element in the min-heap, move the element in the buffer to the root of min-heap and process toMaxHeapify.

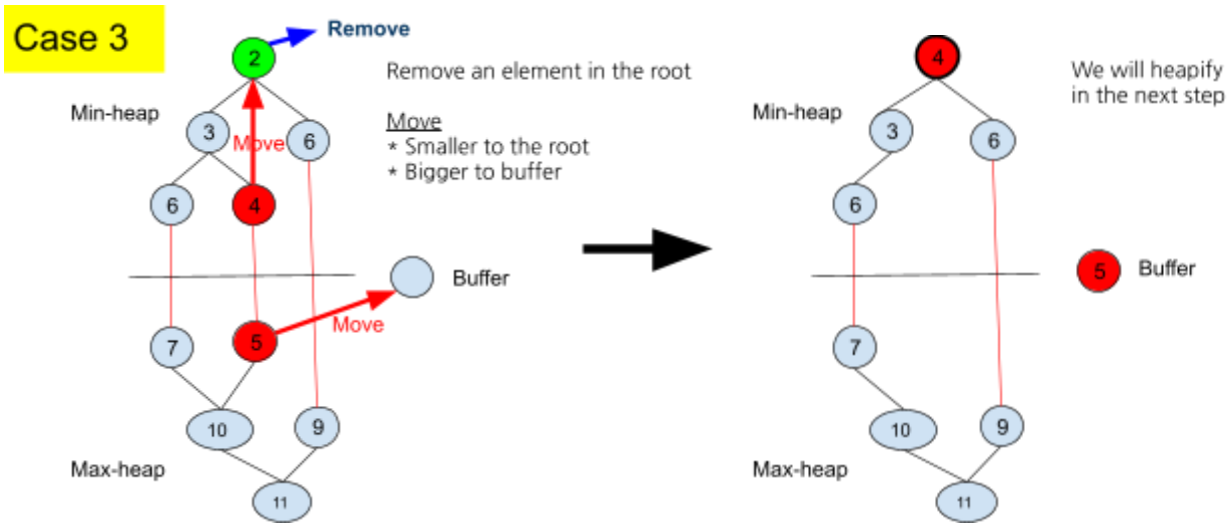


Figure 10: RemoveMin Case 3 where we remove the root element in the min-heap and move two last elements, one to min-heap root and the other to buffer and then toMaxHeapify to correct parent-child relation in the min-heap and the max-heap

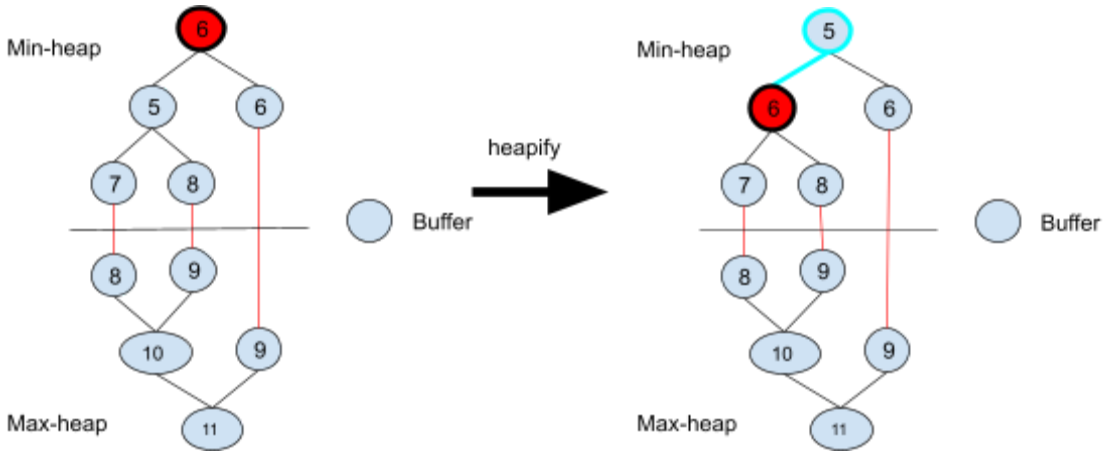


Figure 12: An example of toMaxHeapify where the element which was in the root of min-heap finishes in min heap while a new element in the root is moved to correct parent-child relation in the min-heap and then in the max-heap

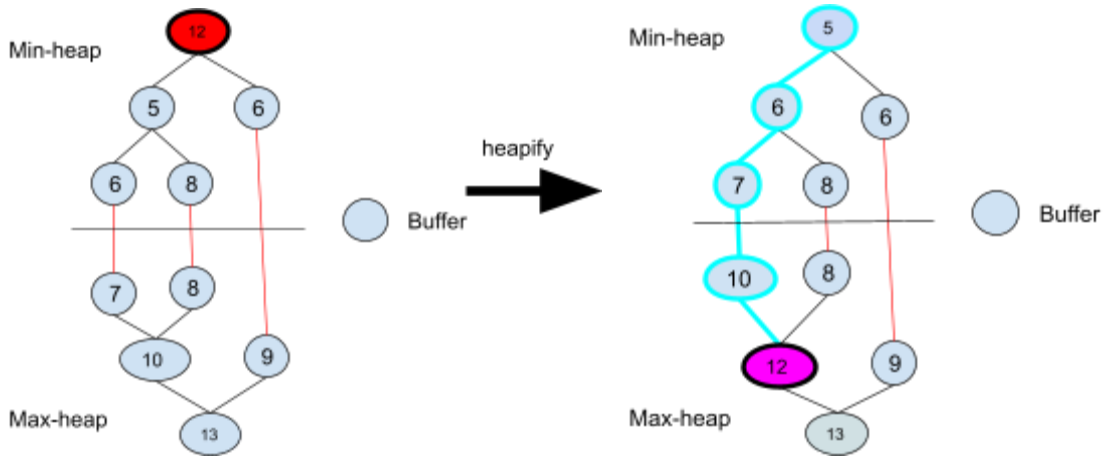


Figure 13: An example of toMaxHeapify where the element which was in the root of min-heap finishes in max heap

5 Remove a Maximum Element

Removing the max element is done by following the same procedure as for the removal of the min element as explained above, except that it now operates from the max-heap.

1. If buffer is not empty and the element in the buffer is bigger than the element at the root of max-heap, then simply return the element in the buffer and empty the buffer.

2. Else you remove the element at the root of the max-heap and then
   2.1. If the buffer is empty, then
      2.1.1. Move a last element of of max-heap to its root.
      2.1.2. Move a last element of of min-heap to the buffer.
   2.2. Else if the buffer is not empty, then
      2.2.1. move the element in the buffer to the root of max-heap.
      2.2.2. Empty buffer.
   2.3. Apply toMinHeapify to the element in the root of the max-heap.

6 Bottom Up Min-Max-Heap Construction

Bottom up heap construction for min-max-heap is done as follows:

1. Swap external elements on last level in the min-heap with associate elements in the max-heap.
2. For (current_level = last level - 1; current_level >= 1; current_level -= 1)
   {
      toMaxHeapify elements on current_level in min-heap to level current_level+1 in max-heap

      toMinHeapify elements on current_level in max-heap to level current_level in min-heap
   }

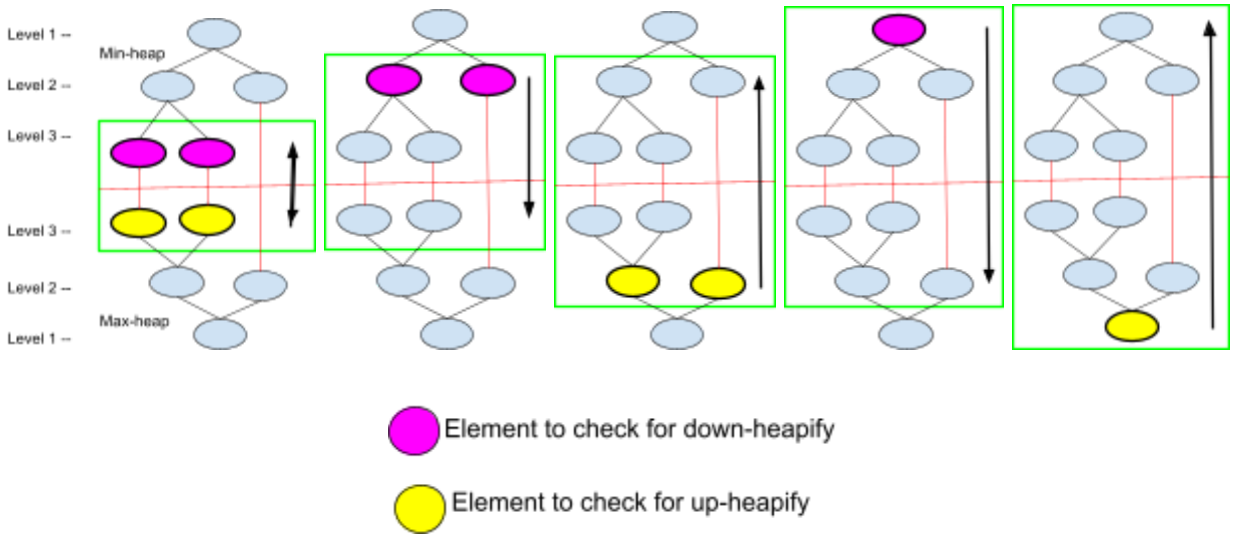


Figure 14: The scope for toMaxHeapify and toMinHeapify is changing for bottom up construction for each step. An element in blue is to be checked for swapping with its child

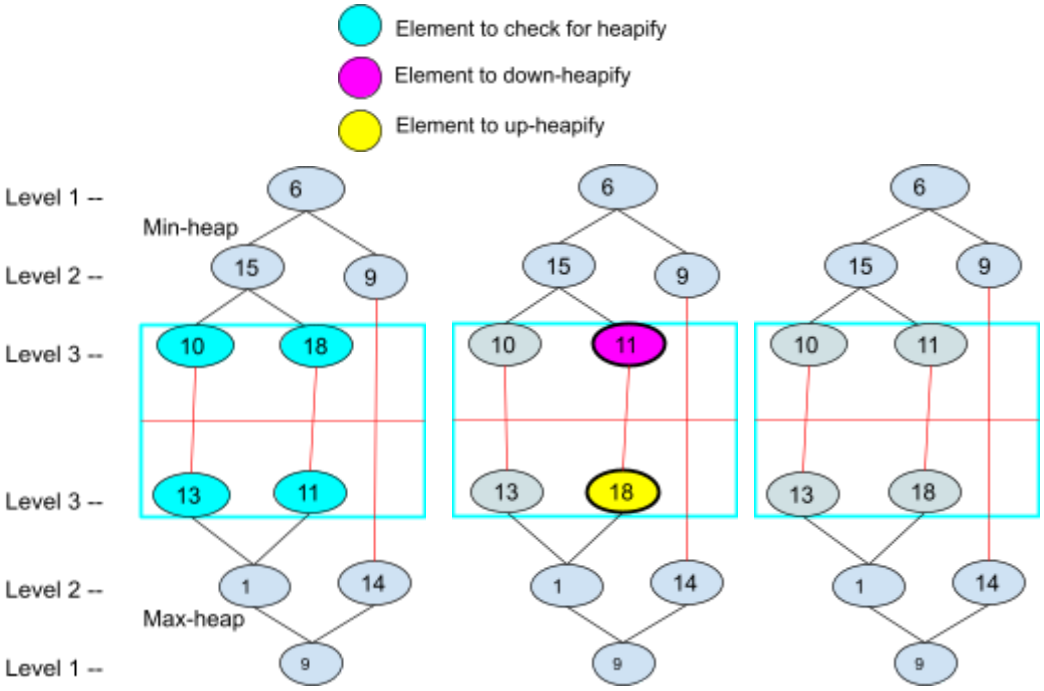


Figure 15: Heapify on last level in min-heap and max-heap

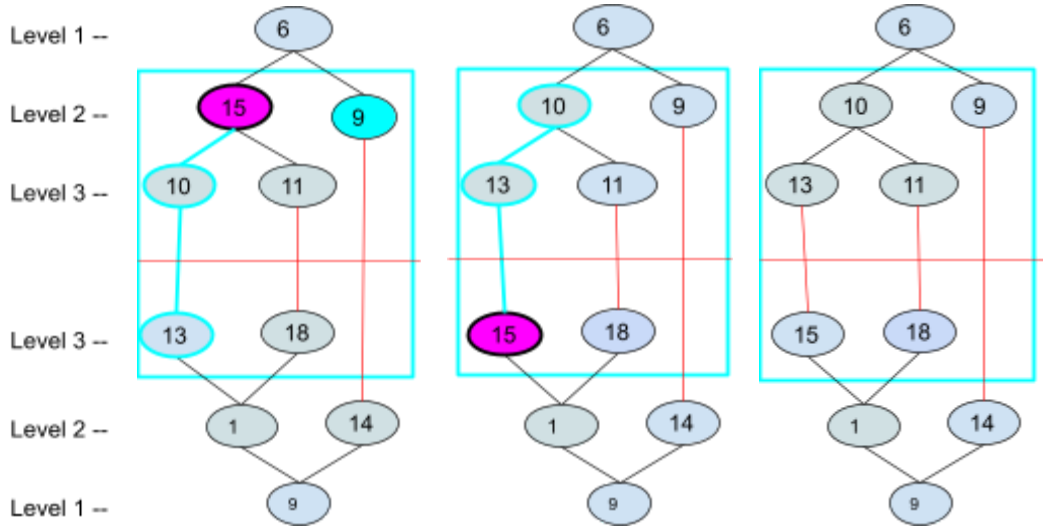


Figure 16: toMaxHeapify on Level 2 in min-heap to level 3 in max-heap

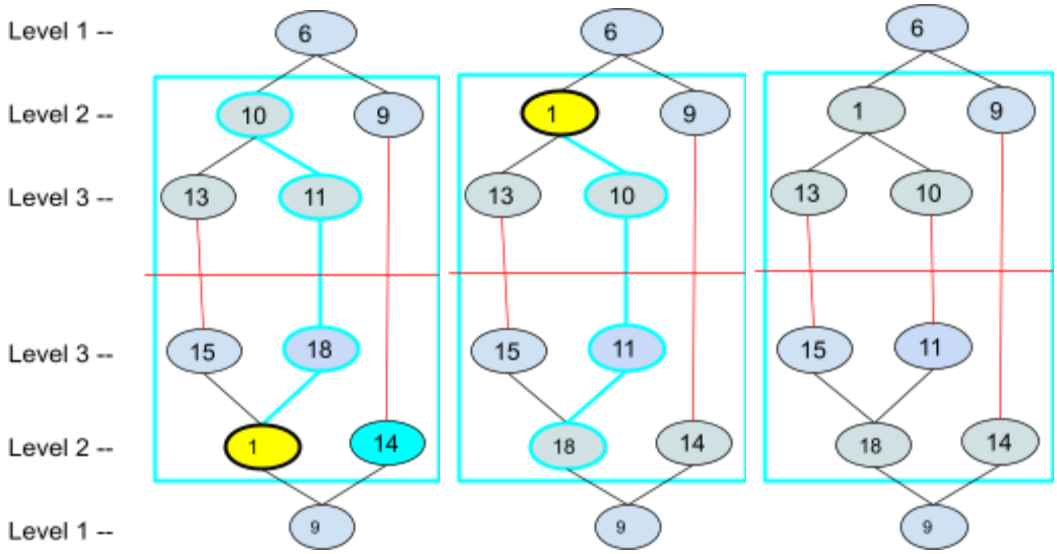


Figure 17: toMinHeapify on Level 2 in max-heap to level 2 in min-heap

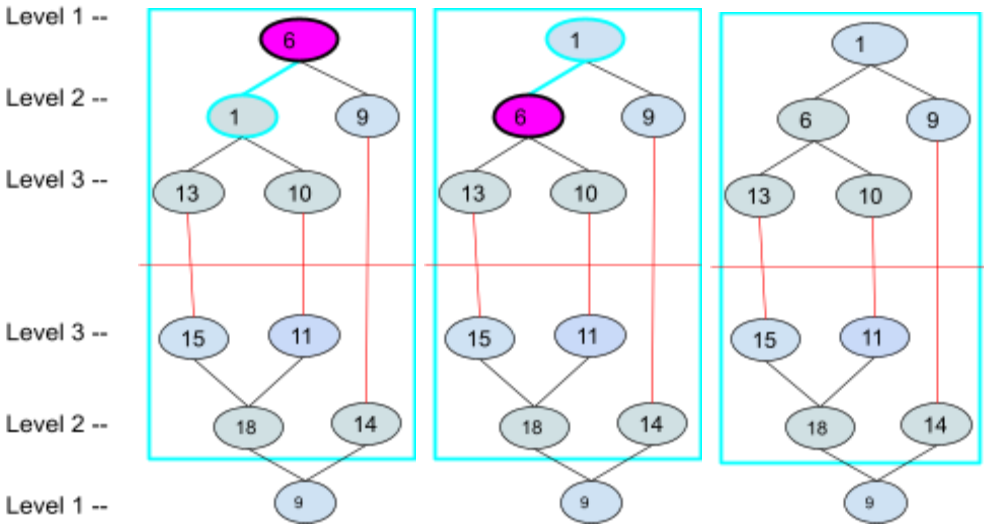


Figure 18: toMaxHeapify on Level 1 in min-heap to level 2 in max-heap

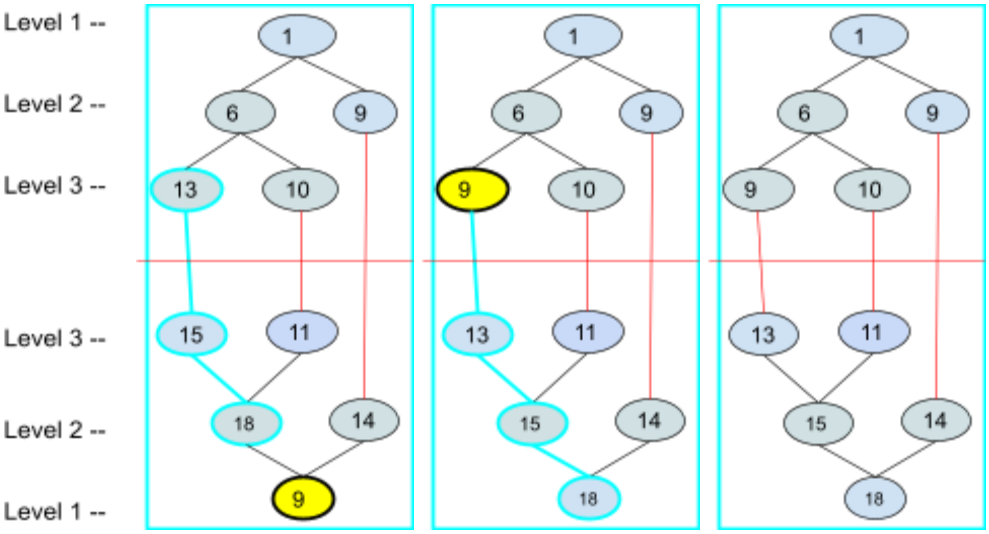


Figure 19: toMinHeapify on Level 1 in max-heap to level 1 in min-heap