Practice 9

Mayra Cristina Berrones Reyes

March 24, 2020

1 Introduction

The subject of this practice is binomial heaps. Binomial heap is an extension of binary heap, and the main application of binary heap is as implement priority queue. So before we enter the subject of binomial heap, we are going to explain this two concepts first.

1.1 Priority queue

A priority queue is a type of data structure like a queue or a stack, with the added element of having a priority associated with each element. Every priority queue has to have the following properties:

- Every item is associated with a priority.
- An element with high priority is going to be dequeued before an element with low priority.
- If two elements have the same priority, they are served in the order they came in.

This type of queues can be solved with a linked list, but it is only efficient when the elements inside the queue do not need to be moved. Generally, priority queues are solved with the implementation of **binary heaps**.

1.2 Binary heap

A binary heap is like a binary tree but with the following properties:

- It is a complete tree.This property makes it so that this type of tree can be stored in an array.

- This binary heap can be either min or max heap. In a min heap the key root must be the minimum value among all the keys inside the tree. In the max heap is the opposite.

Binary heaps are can be efficient in the implementation of priority queues because they support several operations in such as insert, delete, extract max, decrease key. Binomial heap and fibonacci heap are some variations of binary tree.

2 Binomial heap

Binomial heap is an extension of binary heap, and in this case it provides a faster union operation, but it also provides all the other operations mentioned in binary heap. A binomial tree of order 0 has only 1 node. In Figure 1 we see the order of the different types of trees. Following this example, a binomial tree of order k can be constructed by taking 2 binomial trees of order $k-1$ and making one of them the left-most child of the other. This order k binomial tree has the following properties:

- It has exactly 2^k nodes.
- It has a depth of k
- The root has a degree of k and children of the root are themselves Binomial Trees with order k-1, k-2... 0 form left to right.

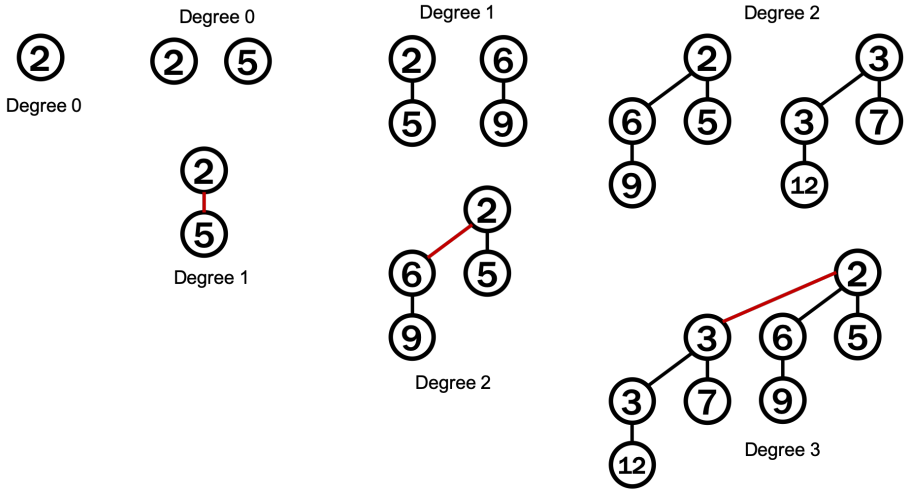

Figure 1: Example of degree level of a binomial heap after we merge trees of the same degree.

Now, as we mentioned before, binomial heap has several operations. First we start with the operation of inserting a new node. The insert operation creates a new heap with the inserted element, which can then be combined using the union operation. In Figure 2 we insert a node with the key 4. Since we already have a node with order 1, we can merge those two together with the union operation. As a result we have a heap with order 1.

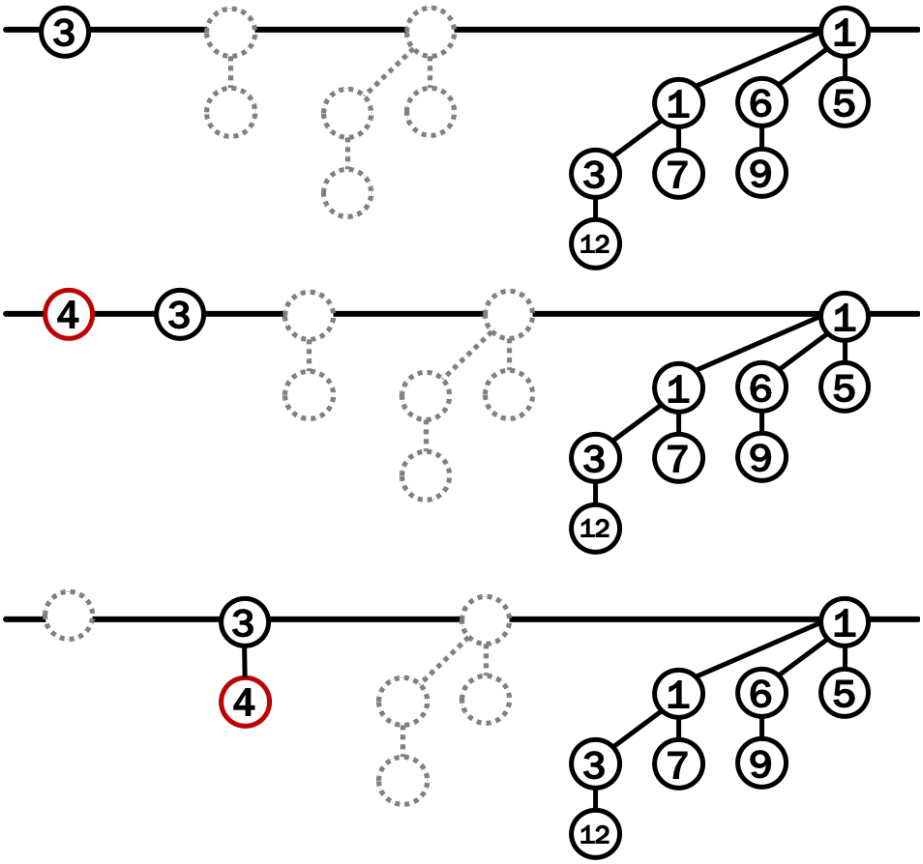

Figure 2: Example of the insert and union operation on heaps. We represent the not used orders with the faded heaps with no keys.

Another operation is the decreasing of a key in a node. As we mentioned in the introduction, this trees are arranged in a way that the smallest key node is at the top (in the case of min tree), so if we decrease the key of a node, depending on the father node of this decreased node, we may need to *"bubble"* up this node until the parent node is smaller than it. To see it in an example, we see in Figure 3, in the first step we take the node with the key 12 and change it to 2. Now, the parent to this node has a key value of 3, so it is necessary to move it up one space, replacing the node 3 with the node 2. Now we see that the new parent is key 1. Since this is smaller than 2, we leave the node in that place.

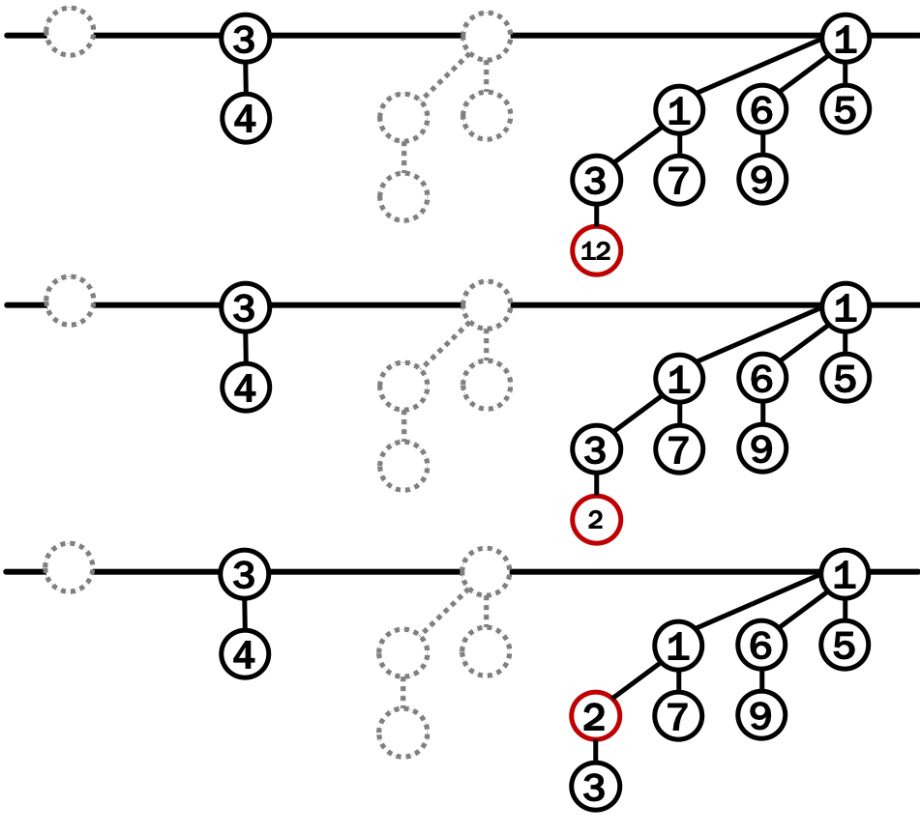

Figure 3: Example of the insert and union operation on heaps. We represent the not used orders with the faded heaps with no keys.

Then we have the find minimum and extract minimum. In this case, since we are working with a min heap, we only need to compare the values of the top node of each heap. In Figure 4 this top nodes are represented with the ones that are on the black line. In this case, the node with the minimum value is 1, so we take it out.

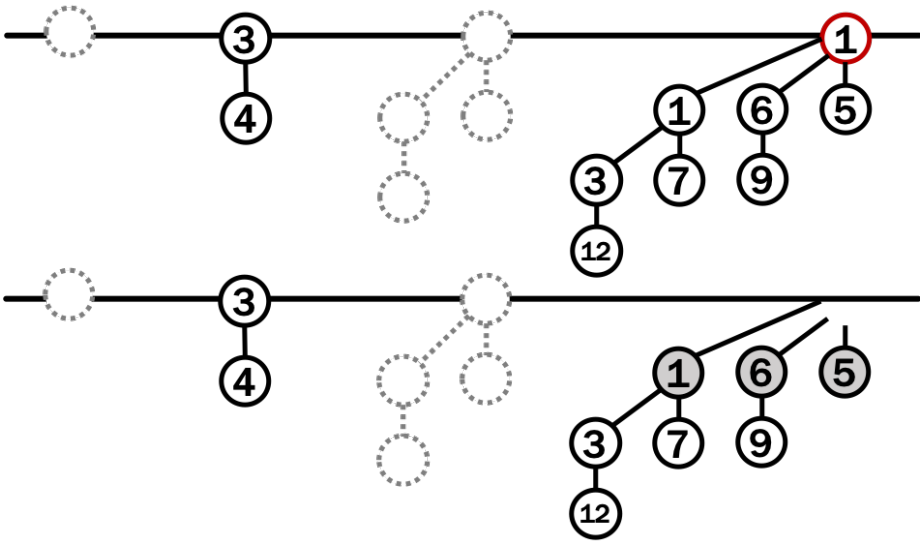

Figure 4: Example of finding the minimum node, and deleting it.

Then, on Figure 5 we put all of the remaining heaps on the line. For convenience we arrange them by their order level, because then it is easier to merge them together with the union operation. In the end we are left with one heap of order 1 and another with the order 3.

For the delete operation, it is similar to this procedure. The only difference is that we first search for the node that we want to delete, then we change that node key to $-\infty$, forcing the node to go up until it reaches the top of the heap, and then we can remove it with the same steps seen in Figure 4 and 5

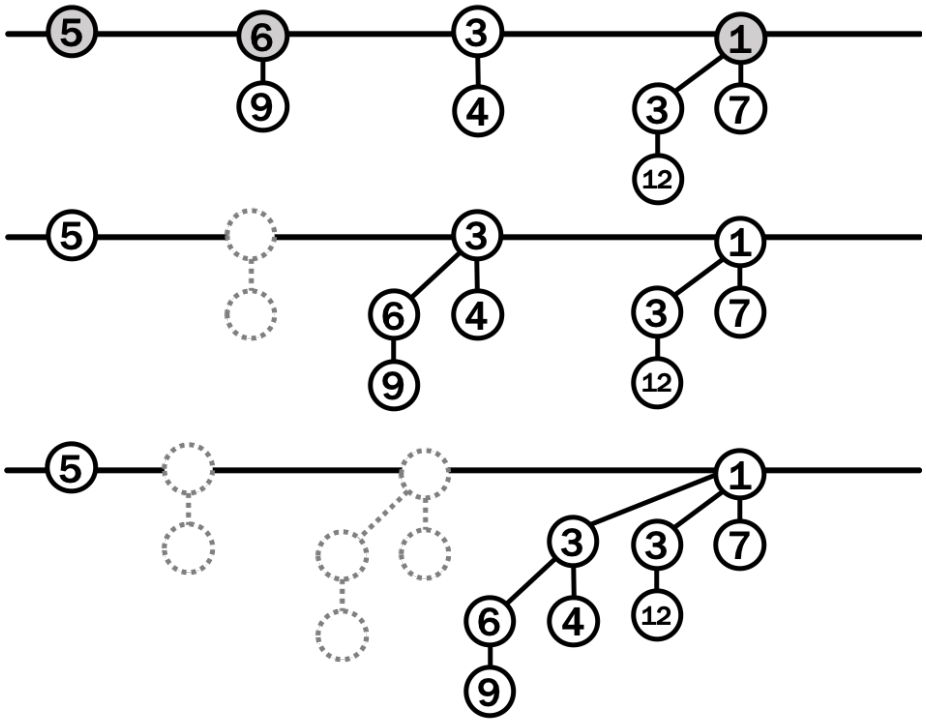

Figure 5: Example of the relocation of the heaps, and the merge of the heaps of the same order.

3 Example

With the same sequence of numbers used in the previous practice, we go inserting one node at the time in the order of *3, 1, 5, 7, 6, 8, 9, 10*. We see the results in Figure 6 and 7.

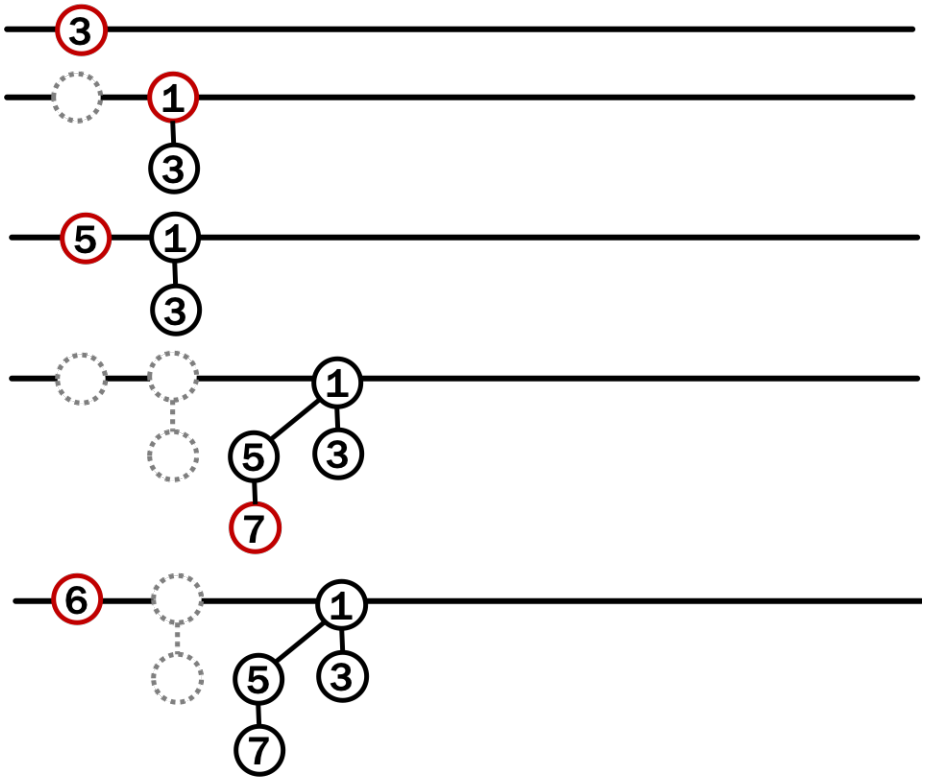

Figure 6: First part of example.

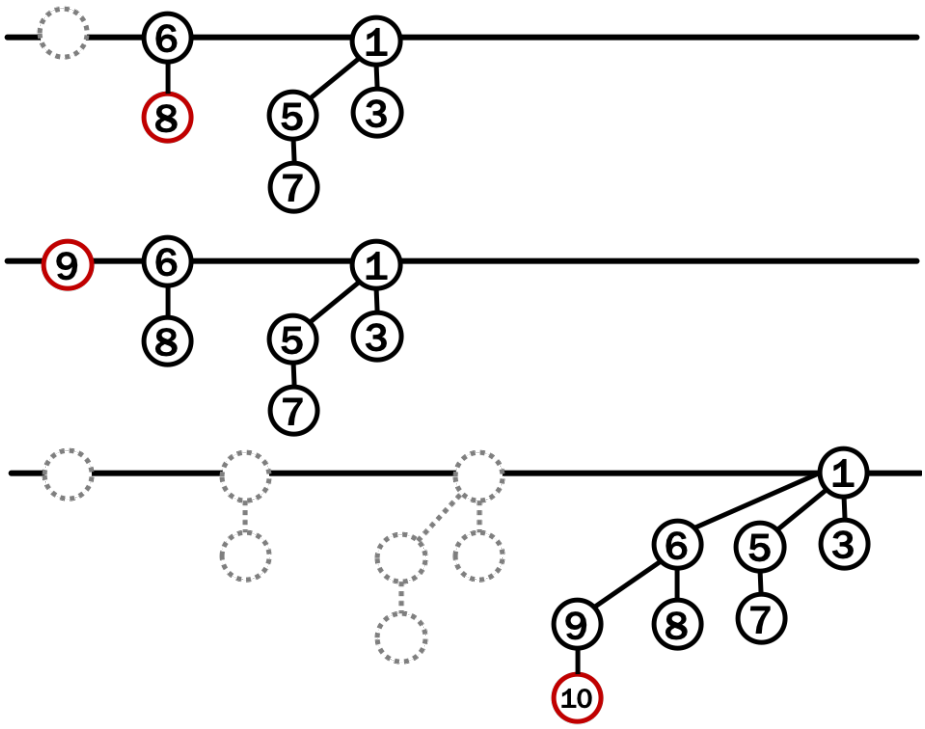

Figure 7: Last part of the example.

Table 1: Performance in running time

Operation	**Binary**	**Binomial**	**Fibonacci**
Find-min	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Delete-min	$\mathcal{O}\log n$	$\mathcal{O}\log n$	$\mathcal{O}\log n$
Insert	$\mathcal{O}\log n$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Dec-key	$\mathcal{O}\log n$	$\mathcal{O}\log n$	$\mathcal{O}(1)$
Merge	$\mathcal{O}\ m\log(n+m)$	$\mathcal{O}\log n$	$\mathcal{O}(1)$

4 Conclusions

In the end, it was a bit difficult for me to understand in which way I could use the binomial heap. The only way I could see it was in a sorting algorithm. In this case, the time it takes to search for the minimum is really fast, because of the way the heaps form and arrange when you put all the elements together and when you take them. I did not understand the fibonacci all that well, but I hope after seeing the explanation of my classmates I can understand better why the fibonacci has a better performance in running time.

References

[1] Binomial heap. https://www.youtube.com/watch?v=7UQd9SYUoNk

[2] Binomial heap https://www.geeksforgeeks.org/binomial-heap-2/

[3] Implement A Binary Heap https://www.youtube.com/watch?v=g9YK6sftDi0