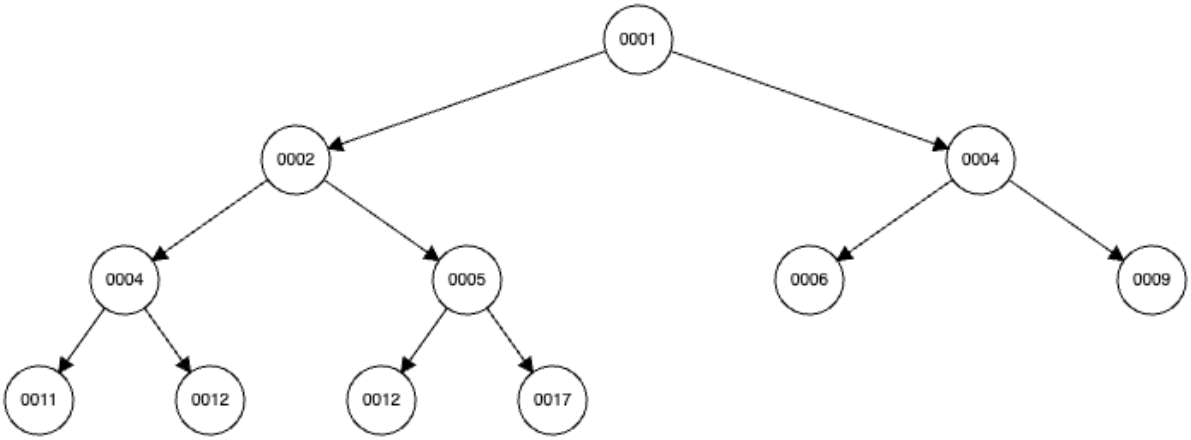
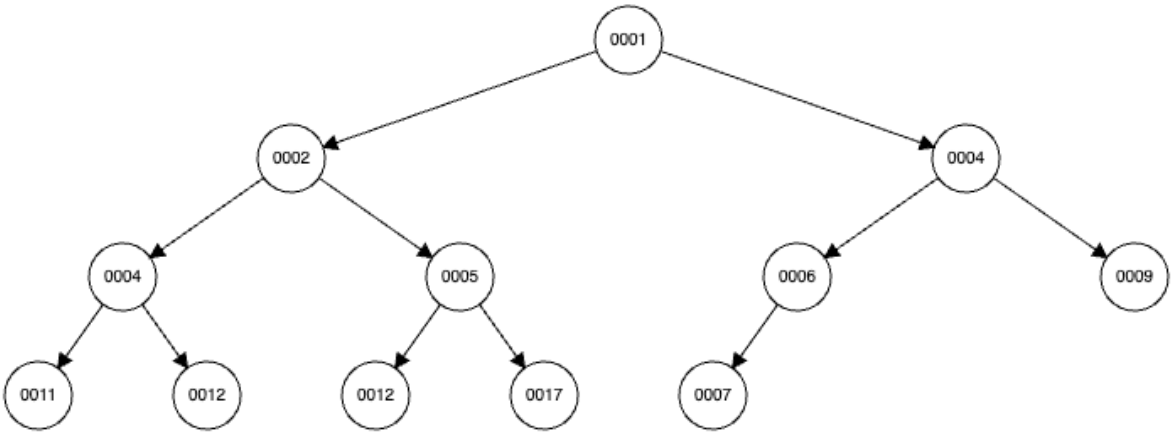
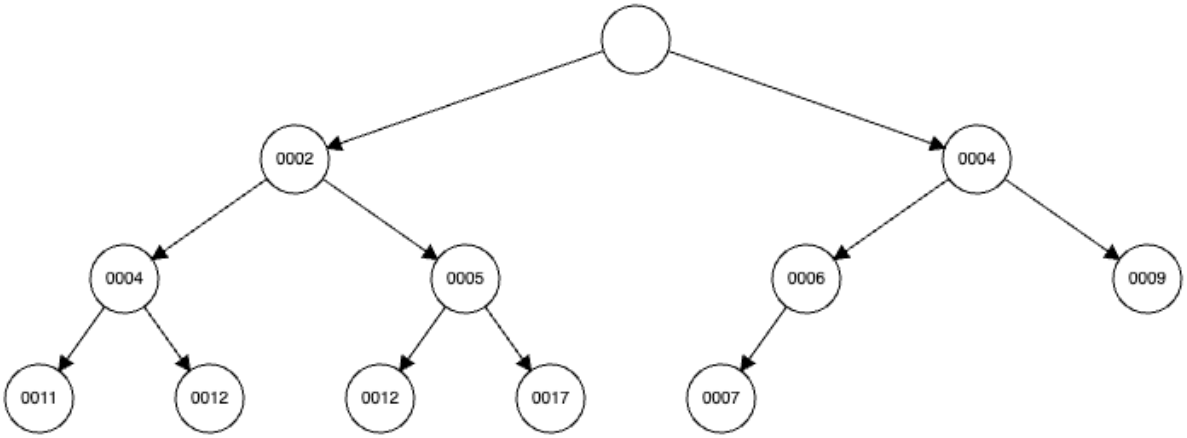
1. Starting with the values 1, 2, 4, 4, 5, 6, 9, 11, 12, 12, 17, do the following:
   a. Create a heap H in which these values are the keys.

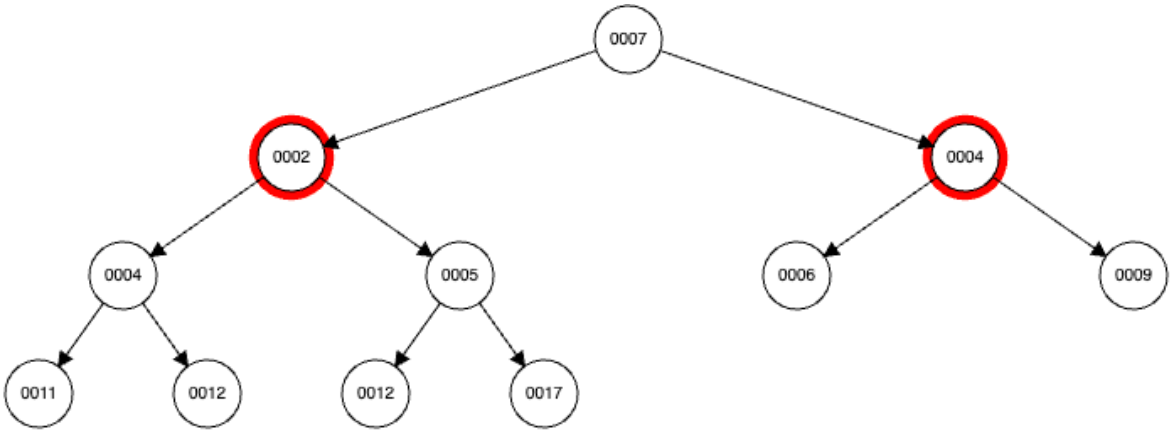


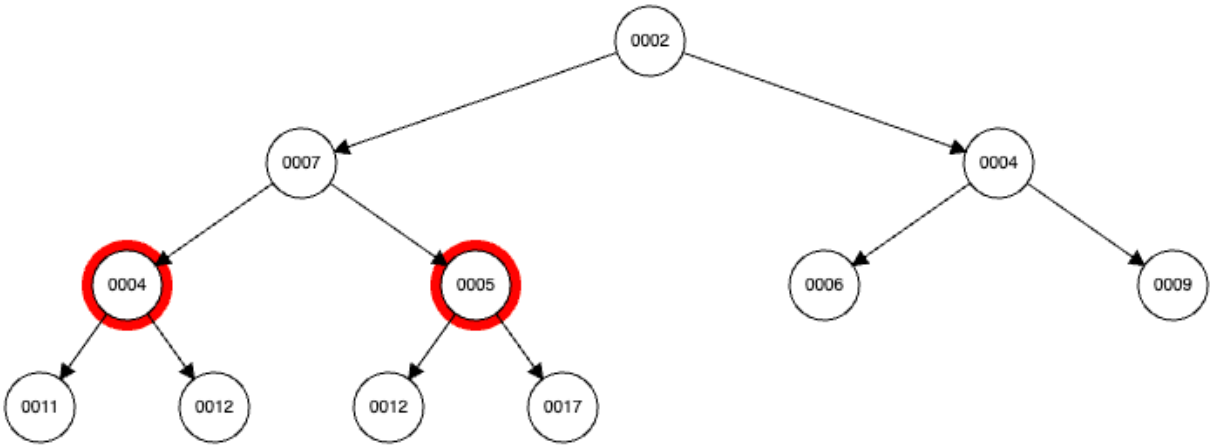
b. Perform the insertItem algorithm to insert the value 7 into H. Show all steps.

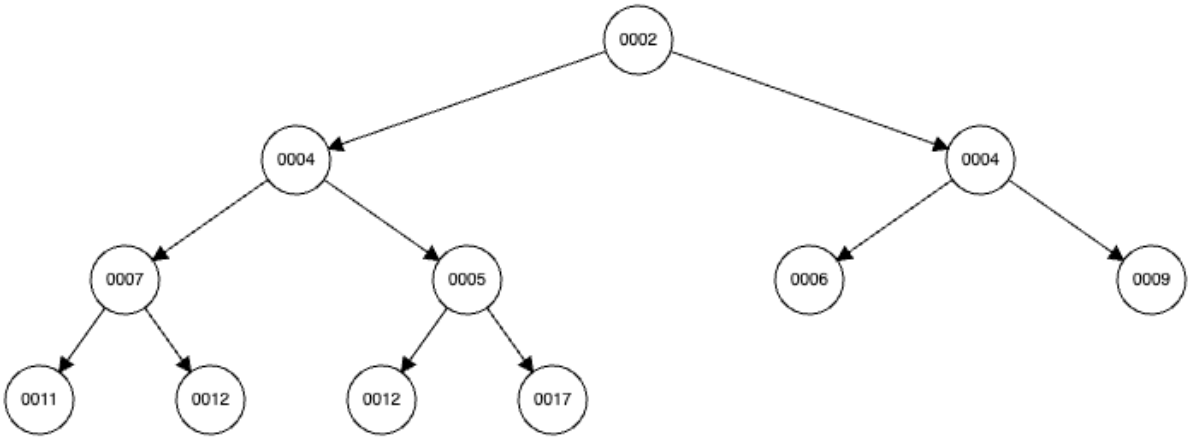


c. Perform the removeMin algorithm on H and show all steps.









d. Represent H in the form of an array A.
[2, 4, 4, 7, 5, 6, 9, 11, 12, 12, 17]

e. Perform the array-based insertItem algorithm to insert 14 into A – show all steps.
newIndex = 2i + 1 -> i = (newIndex - 1) / 2 = (11 - 1) / 2 = 5
arr[5] = 6, 11 > 6 —> no need rotation
[2, 4, 4, 7, 5, 6, 9, 11, 12, 12, 17]

f. Perform the array-based removeMin algorithm on A – show all steps.
[2, 4, 4, 7, 5, 6, 9, 11, 12, 12, 17, 14]
-> [14, 4, 4, 7, 5, 6, 9, 11, 12, 12, 17]
-> [4, 14, 4, 7, 5, 6, 9, 11, 12, 12, 17]
-> [4, 5, 4, 7, 14, 6, 9, 11, 12, 12, 17]
-> [4, 5, 4, 7, 12, 6, 9, 11, 12, 14, 17]

2. Carry out the array-based version of HeapSort on the input array [1, 4, 3, 9, 12, 2, 4] Show steps and outputs along the way. Make sure to distinguish between Phase I and Phase II of the algorithm.

Phase 1:

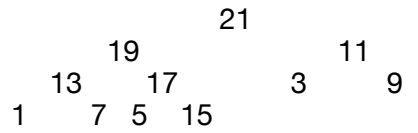
	1 | 4 3 9 12 2 4
	1 4 | 3 9 12 2 4
upheap	4 1 | 3 9 12 2 4
	4 1 3 | 9 12 2 4
	4 1 3 9 | 12 2 4
upheap	4 9 3 1 | 12 2 4
upheap	9 4 3 1 | 12 2 4
	9 4 3 1 12 | 2 4
upheap	9 12 3 1 4 | 2 4
upheap	12 9 3 1 4 | 2 4
	12 9 3 1 4 2 | 4
	12 9 3 1 4 2 4
upheap	12 9 4 1 4 2 3

Phase 2:

3 9 4 1 4 2 | 12
9 3 4 1 4 2 | 12
9 4 4 1 3 2 | 12
2 4 4 1 3 | 9 12
4 2 4 1 3 | 9 12
4 3 4 1 2 | 9 12
2 3 4 1 | 4 9 12

4 3 2 1 | 4 9 12
1 3 2 | 4 4 9 12
3 1 2 | 4 4 9 12
2 1 | 3 4 4 9 12
1 2 3 4 4 9 12

4. Draw an example of a MaxHeap whose keys are all the odd numbers lie in [1, 21] (with no repeats), such that the insertion of an item with key 14 would cause up-heap to proceed all the way up to a child of the root (replacing that child's key with 14).
   [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]



->