HW7

~~HW6~~

Name: Key

1. Do 6.1.8 (pg. 207). Please give clear pseudocode. A good answer should be easy to convert directly into working code.

```
MaxOpenInterval (List<Pair>L) // Pair could just 2-value
                              // object we define
  max = 0
  sort (L) on first value // Use comparator or Lambda
                          // or ordered by second item
  open = 0
  MinH = <empty min. heap with capacity for all pairs>
  while (!L.isEmpty())
      if ( MinH.size()>0
        && MinH.find_max() <= L[0].first) {
          MinH.delete_max()
          --open;
      } else {
          MinHInsert(L[0])
          L.delete(0) // delete 1st item
          ++open
          if(open>max) max=open
      }
  }
  return max;
```

Q(n log n) w/ merge sort

O(log n) ≈ n times in worst case

insert (O(log n)) in n times → O(n log n)

<Alternate, avoids other data structure>

HW7

Name: Key

1. Do 6.1.8 (pg. 207). Please give clear pseudocode. A good answer should be easy to convert directly into working code.

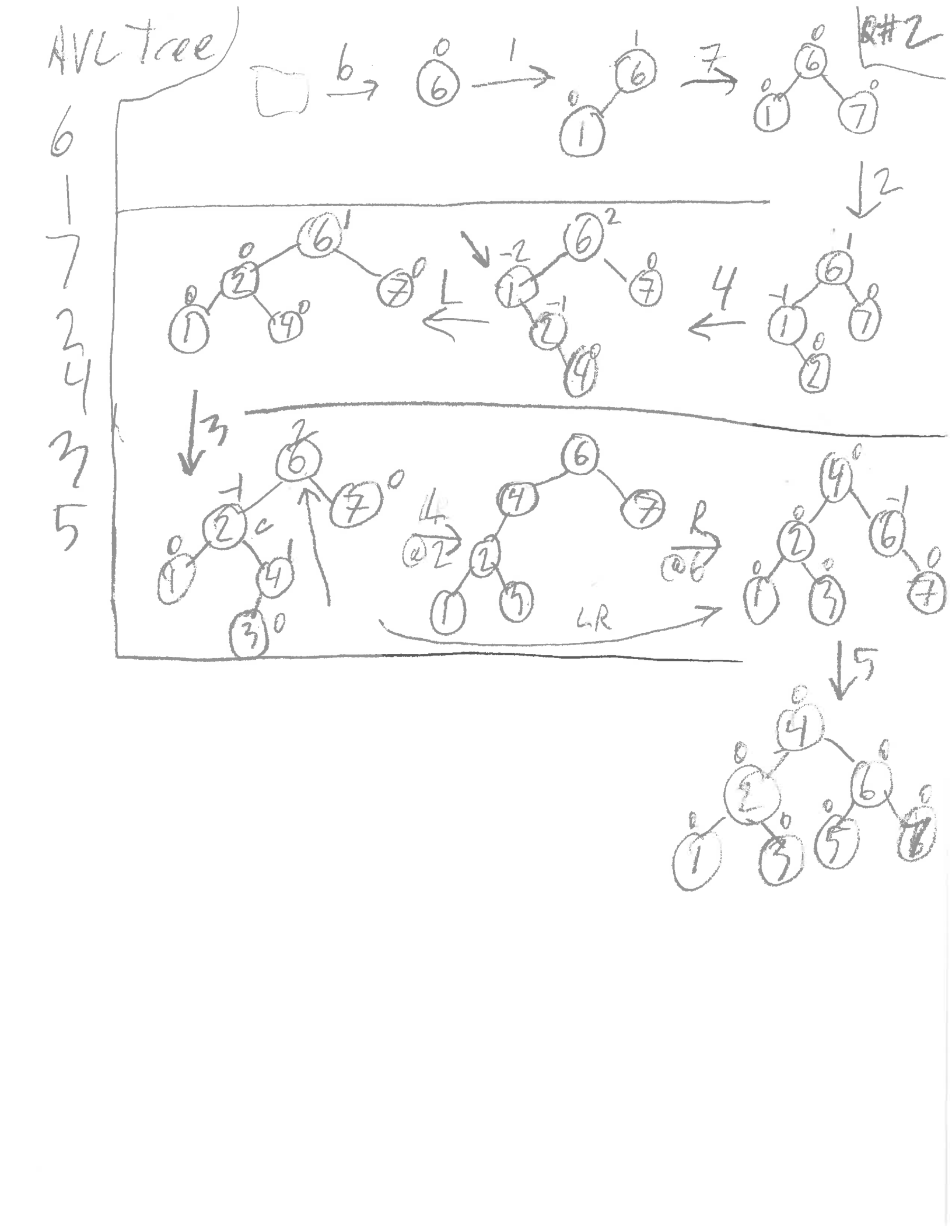
```
maxOpenInterval (List<Interval> intervals) {
    List<Pair> ends;                                          O(n)
    for each interval in Intervals {
        add new Pair("Open", interval.first) to ends
        add new Pair("Close", interval.second) to ends }
    sort(ends) // use custom comparison function              O(n log n)
               // sort on second value, so the close < open

    int max = 0
    int open = 0
    for (int i = 0; i < ends.size; ++i) {                     O(n)
        if p[i].first == "Close" {
            --open
        } else { ++open.
            if (open > max) max = open;
        }
    }
    return max;
}
```

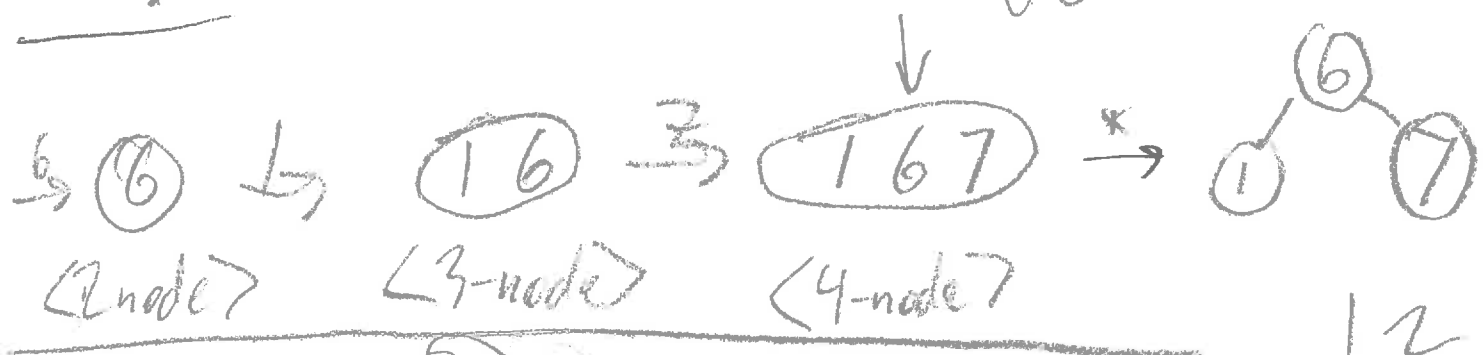
AVL tree

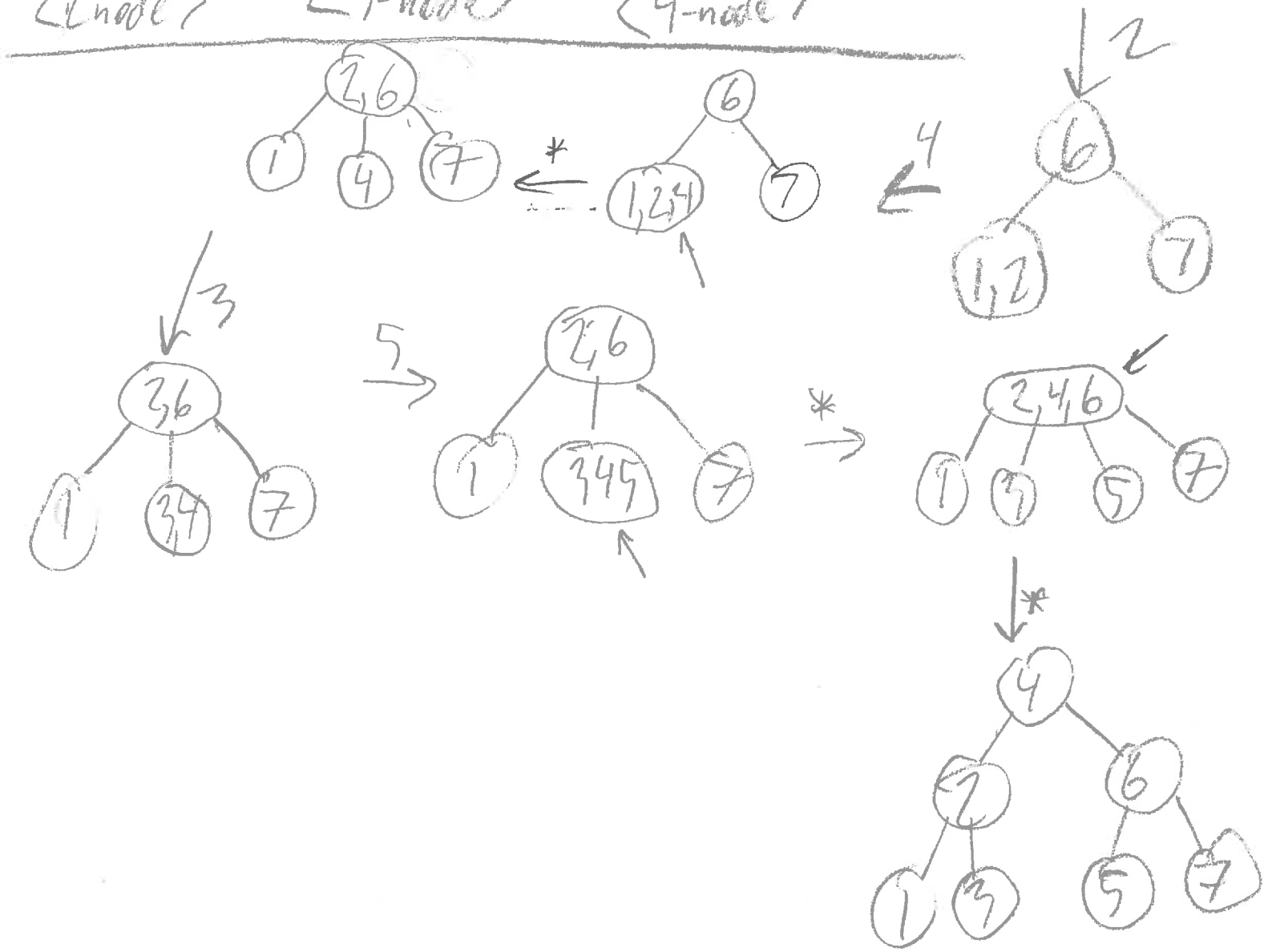
Q#2

6 1 7 2 4 3 5



Q#3 | 2-3 tree from 6 1 7 2 4 3 5

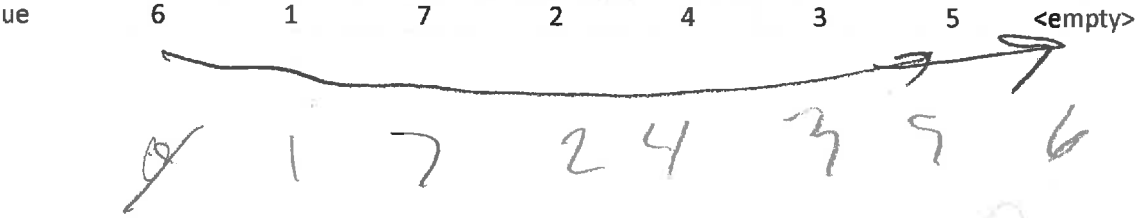


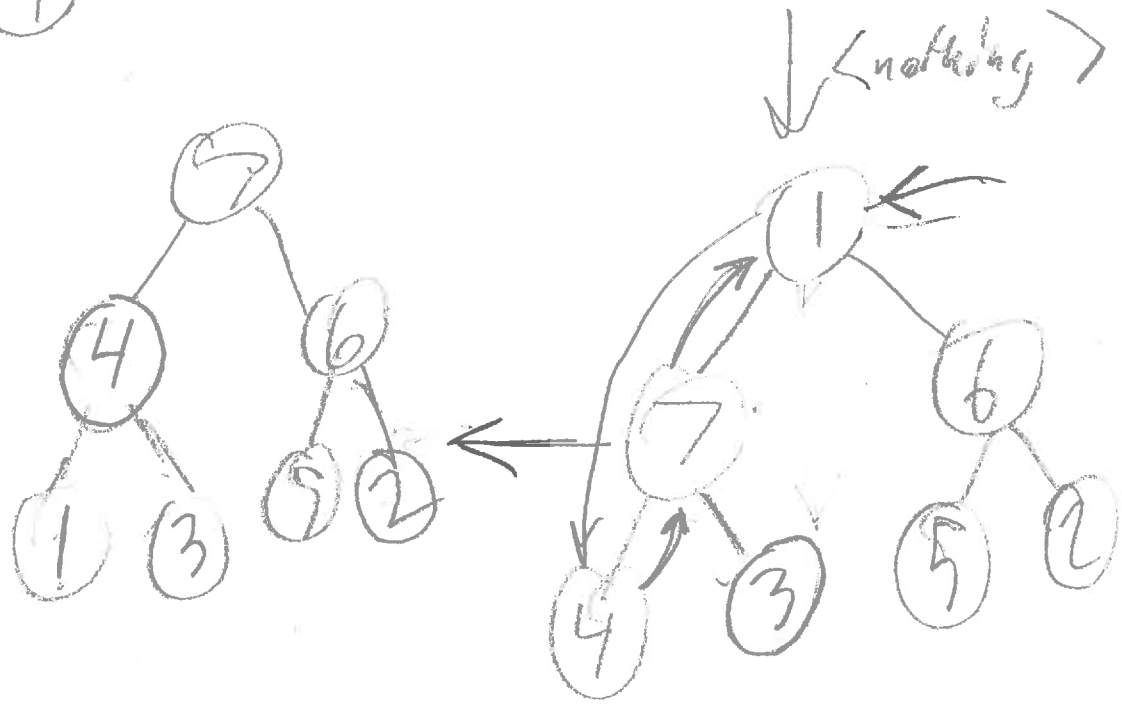


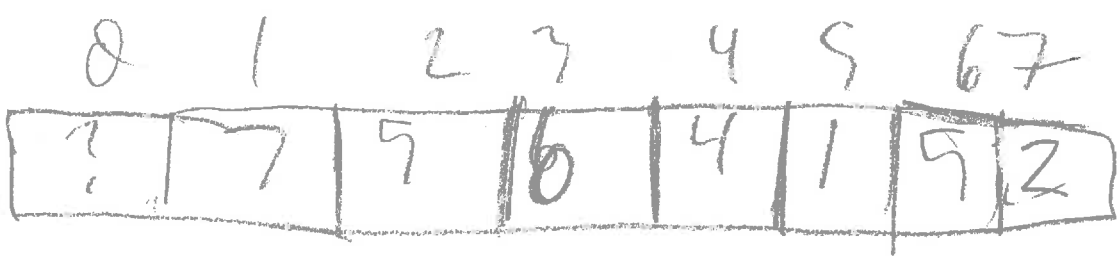
<if you copy first to end>

4. Max-Heapify the following array. You must use the bottom-up approach. You may construct the tree to show work. Give the resulting *array* at the end. You may assume the is extra capacity at the end.

Index	0	1	2	3	4	5	6	7
Value	6	1	7	2	4	3	5	<empty>







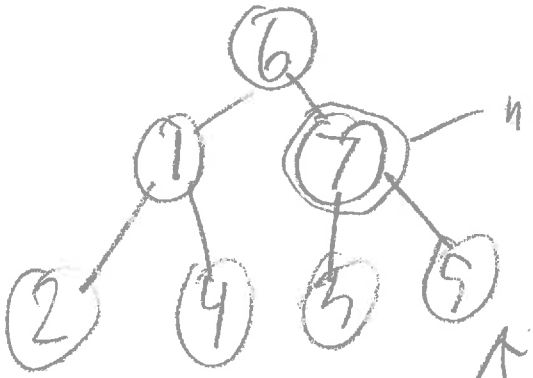
< if you shift entire array right one)

4. Max-Heapify the following array. You must use the bottom-up approach. You may construct the tree to show work. Give the resulting array at the end.

Index	0	1	2	3	4	5	6	7
Value	6	1	7	2	4	3	5	<empty>

n=7

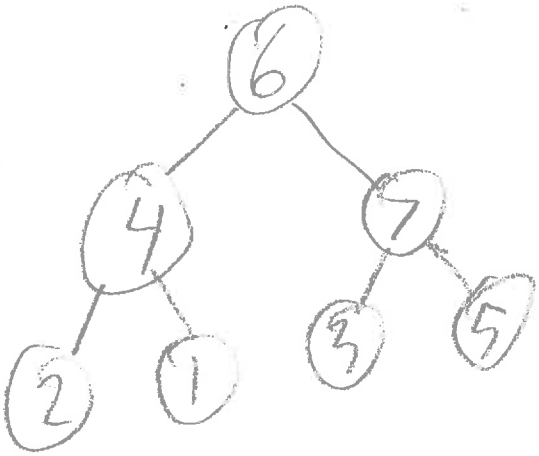
$\lfloor 7/2 \rfloor = 3$ (1st Parent)



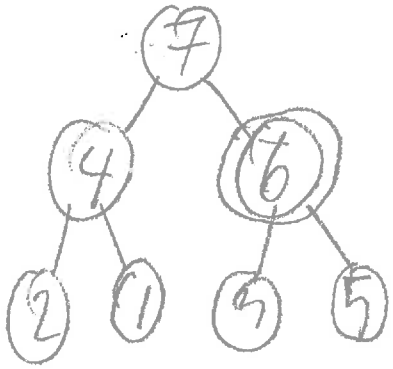
nothing to do @ 7

K=3, 7 valid to start

<sup, 1, 4> if (4 > 1 < max(4,2)

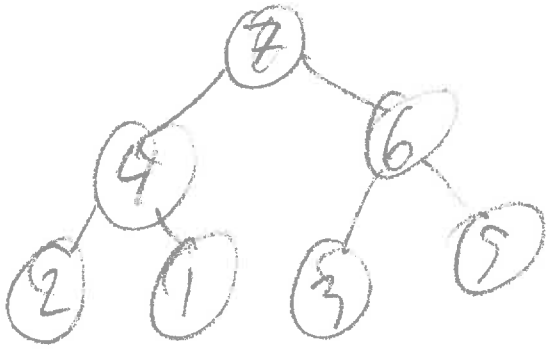


6 < max(4,7)



↓ 6 is valid parent

<done>



0	1	2	3	4	5	6	7
3	7	4	6	2	1	3	5

really probably has 6, but don't care