SORTING

CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

17: Heapsort

Alexandra Papoutsaki
she/her/hers

Lecture 17: Heapsort

- Heapsort

Basic plan for heap sort

- Use a priority queue to develop a sorting method that works in two steps:
- 1) Heap construction: build a binary heap with all n keys that need to be sorted.
- 2) Sortdown: repeatedly remove and return the maximum key.

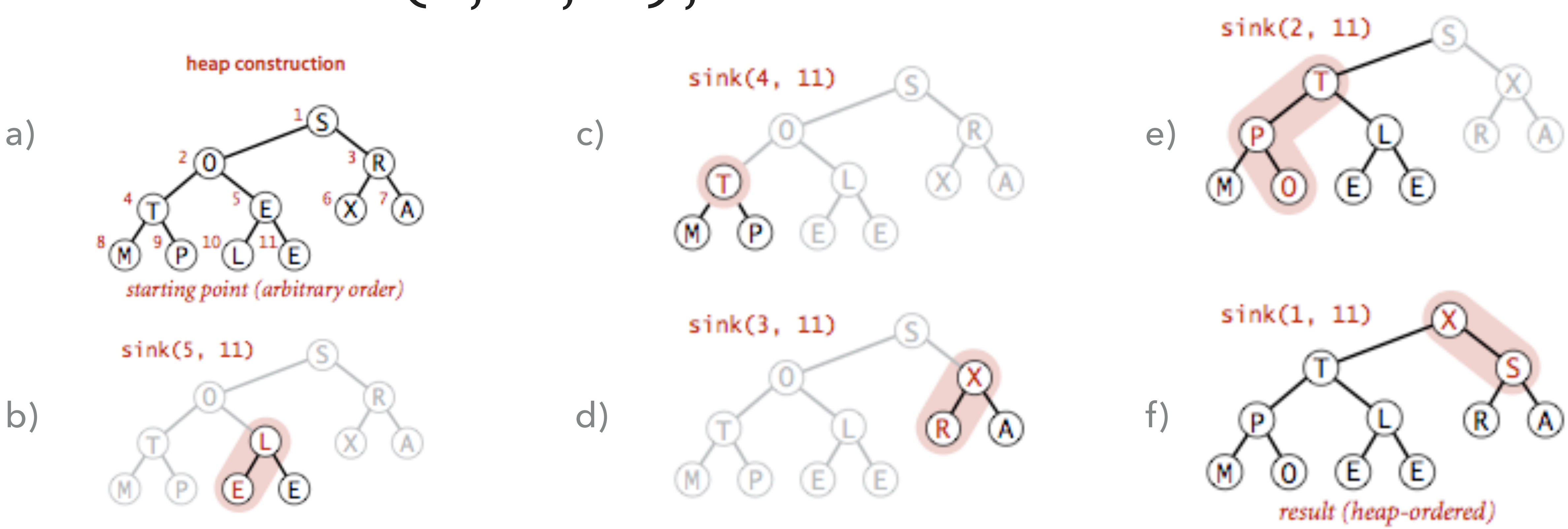
$O(n \log n)$ Heap construction

- Insert n elements, one by one, swim up to their appropriate position.
- We can do better!
- Key insight: After `sink(a,k,n)` completes, the subtree rooted at k is a heap.

```
private static void sink(Comparable[] a, int k, int n) {
    while (2*k <= n) {
        int j = 2*k;
        if (j < n && a[j-1].compareTo(a[j]) < 0){
            j++
        }
        if (a[k-1].compareTo(a[j-1]) >= 0){
             break;
        }
        Comparable temp = a[k-1];
        a[k-1] = a[j-1];
        a[j-1] = temp;
        k = j;
    }
}
```

$O(n)$ Heap construction

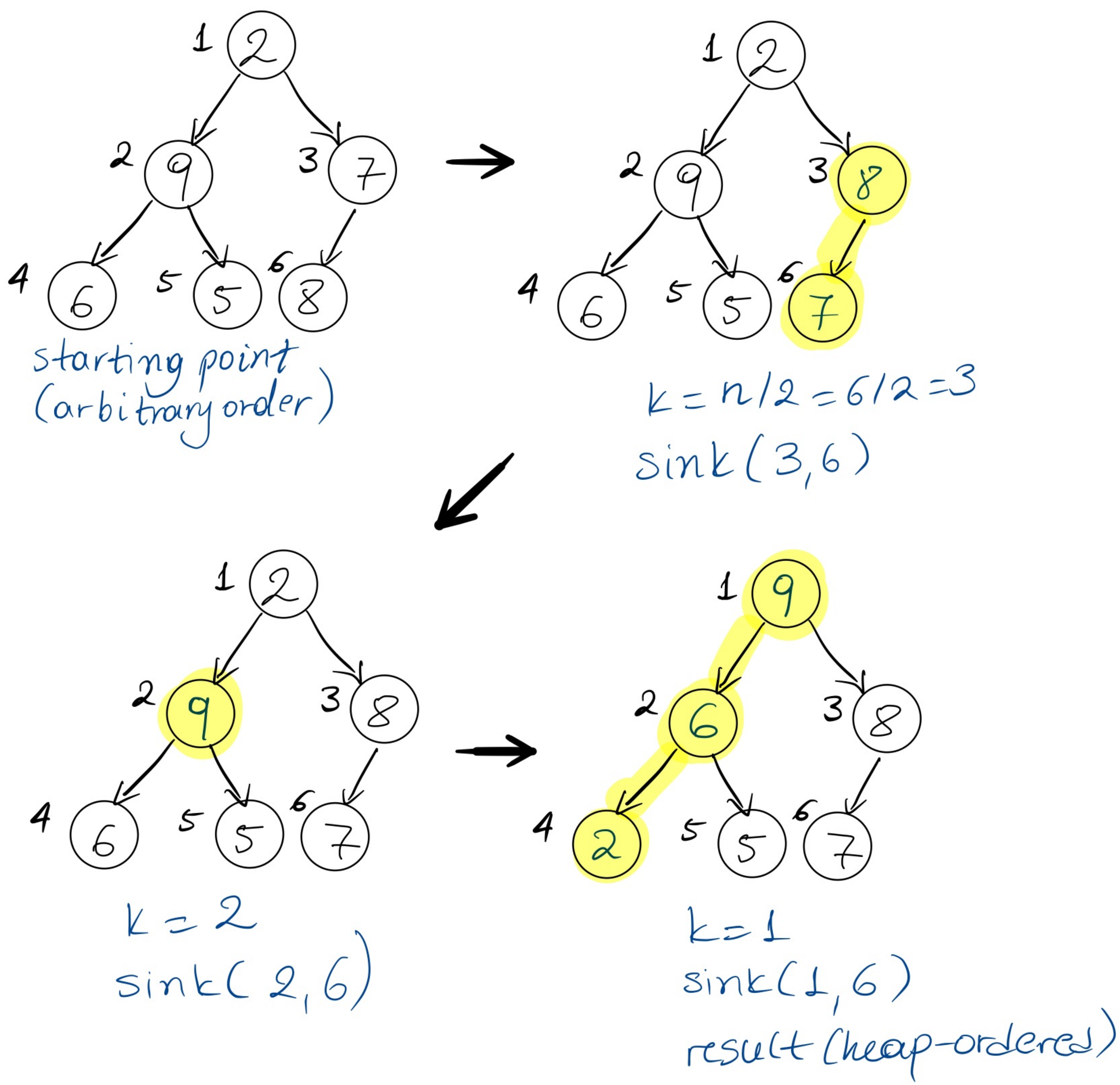
- Insert all nodes as is in indices 1 to n. We will turn this binary tree into a heap.
- Ignore all leaves (indices n/2+1,…,n). Sink each internal node
- ```
  for(int k = n/2; k >= 1; k--)
          sink(a, k, n);
  ```


  ```

Practice Time - Worksheet #17

- Run the first step of heapsort, heap construction, on the array [2,9,7,6,5,8].

Answer: Heap construction

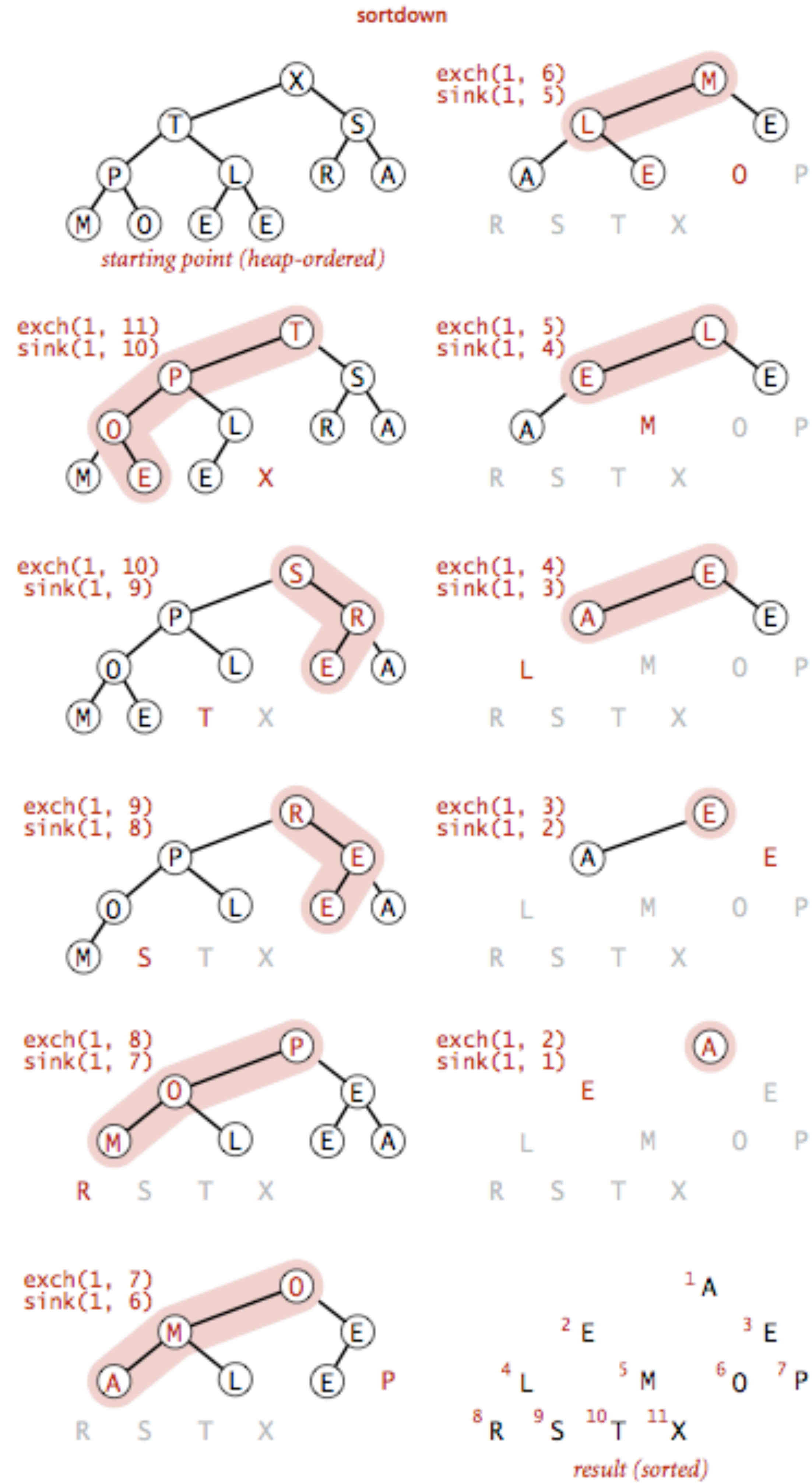


Sortdown

- Remove the maximum, one at a time, but leave in array instead of nulling out.
- ```
  while(n>1){
     exch(a, 1, n--);
     sink(a, 1, n);
  }
  ```
- Key insight: After each iteration the array consists of a heap-ordered subarray followed by a sub-array in final order.
  ```

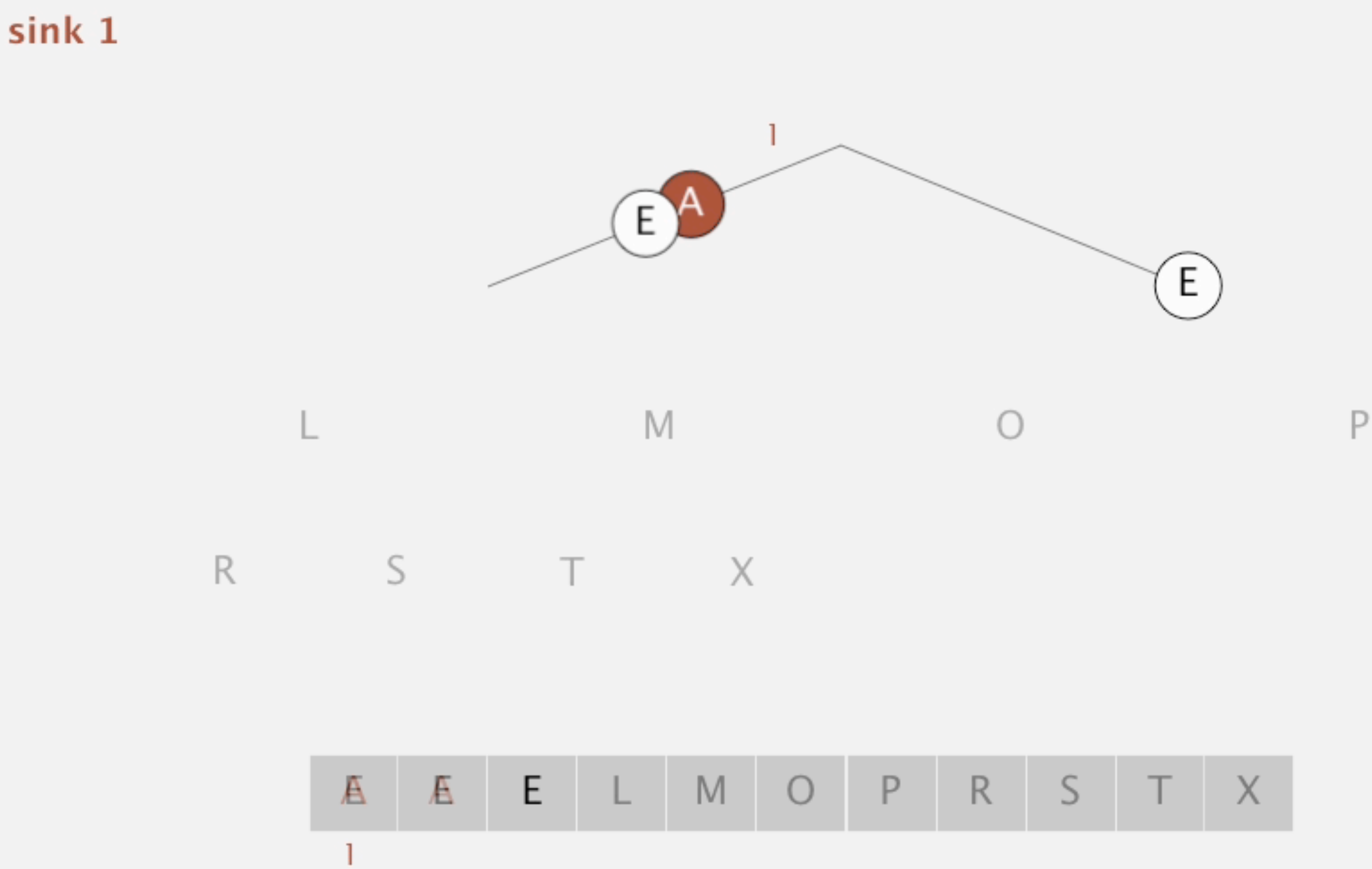
Sortdown

- ```
  while(n>1){
     exch(a, 1, n--);
     sink(a, 1, n);
  }
  ```


  ```

Heapsort demo

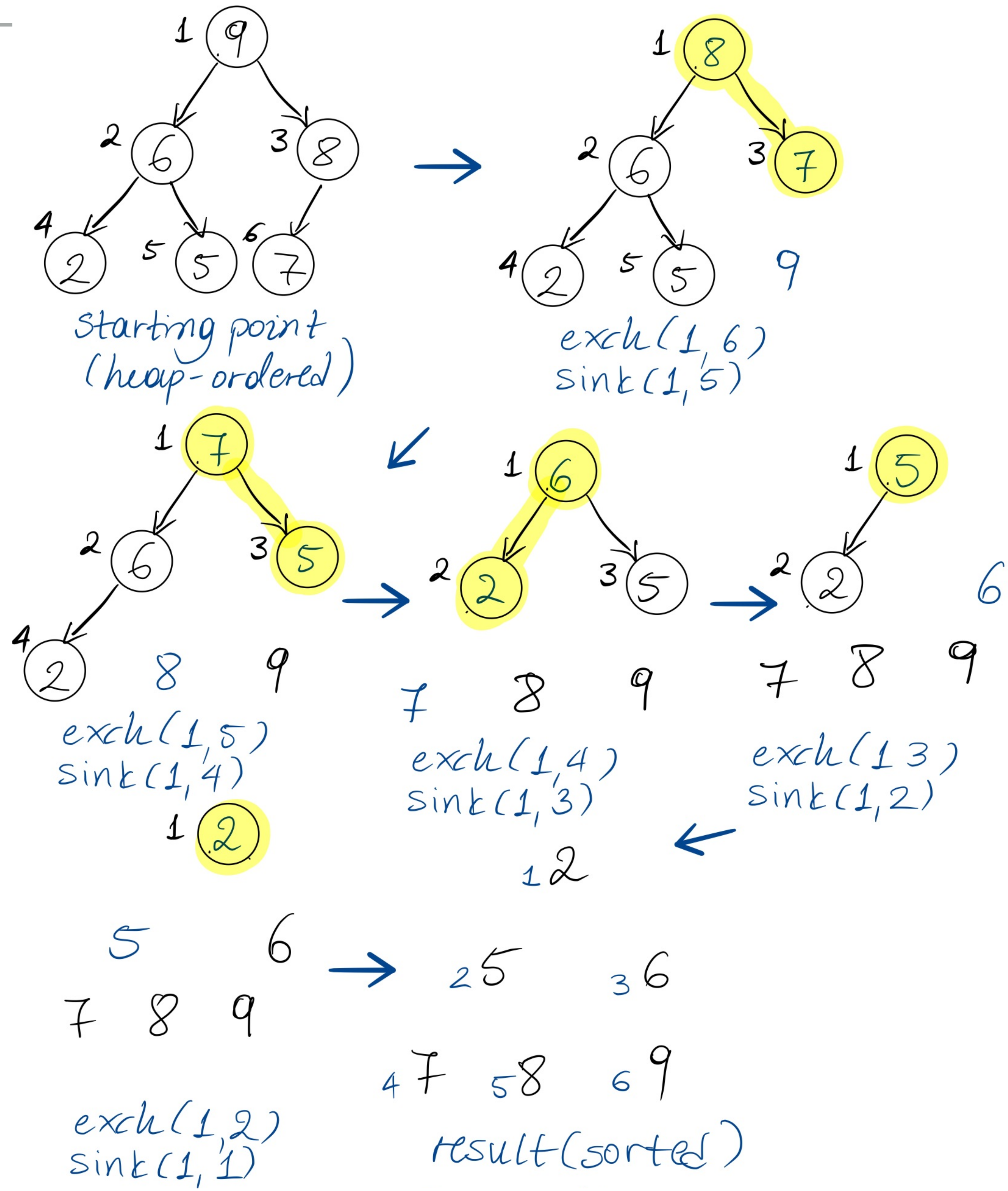
Sortdown. Repeatedly delete the largest remaining item.



Practice Time

- Given the heap you constructed before, run the second step of heapsort, sortdown, to sort the array [2,9,7,6,5,8].

Answer: Sortdown



Heapsort analysis

- Heap construction (the fast version) makes $O(n)$ exchanges and $O(n)$ compares.
- Sortdown and therefore the entire heapsort $O(n \log n)$ exchanges and compares.
- In-place sorting algorithm with $O(n \log n)$ worst-case!
- Remember:
  - mergesort: not in place, requires linear extra space.
  - quicksort: quadratic time in worst case.
- Heapsort is optimal both for time and space in terms of Big-O, but:
  - Inner loop longer than quick sort.
  - Poor use of cache because it accesses memory in non-sequential manner, jumping around.
    - more in CS105!
  - Not stable.

Sorting: Everything you need to remember about it!

Which Sort	In place	Stable	Best	Average	Worst	Remarks
Selection	X		$O(n^2)$	$O(n^2)$	$O(n^2)$	n exchanges
Insertion	X	X	$O(n)$	$O(n^2)$	$O(n^2)$	Use for small arrays or partially ordered
Merge		X	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Guaranteed performance; stable
Quick	X		$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$n \log n$ probabilistic guarantee; fastest!
Heap	X		$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Guaranteed performance; in place

Lecture 17: Heapsort

- Heapsort

Readings:

- Recommended Textbook:
  - Chapter 2.4 (Pages 308-327), 2.5 (336-344)
- Website:
  - Priority Queues: https://algs4.cs.princeton.edu/24pq/
- Visualization:
  - Create (compare the n and nlogn approaches) and heapsort: https://visualgo.net/en/heap

Worksheet

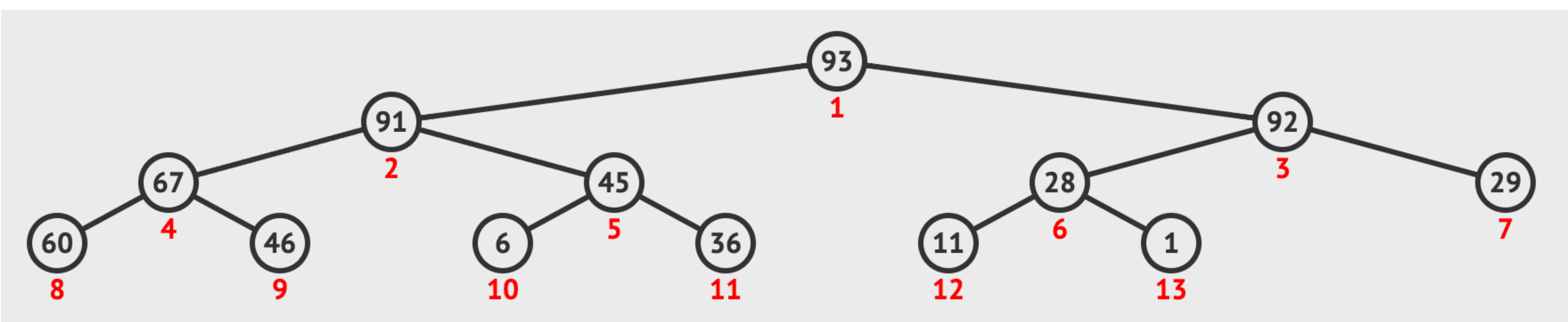
- Lecture 17 worksheet

Practice Problem 1

- Given the array [93,36,1,46,91,92,29,60,67,6,45,11,28], apply heap sort. Visualize what the heap will initially look like (apply the O(n) algorithm) and visualize it at the end of each deletion.

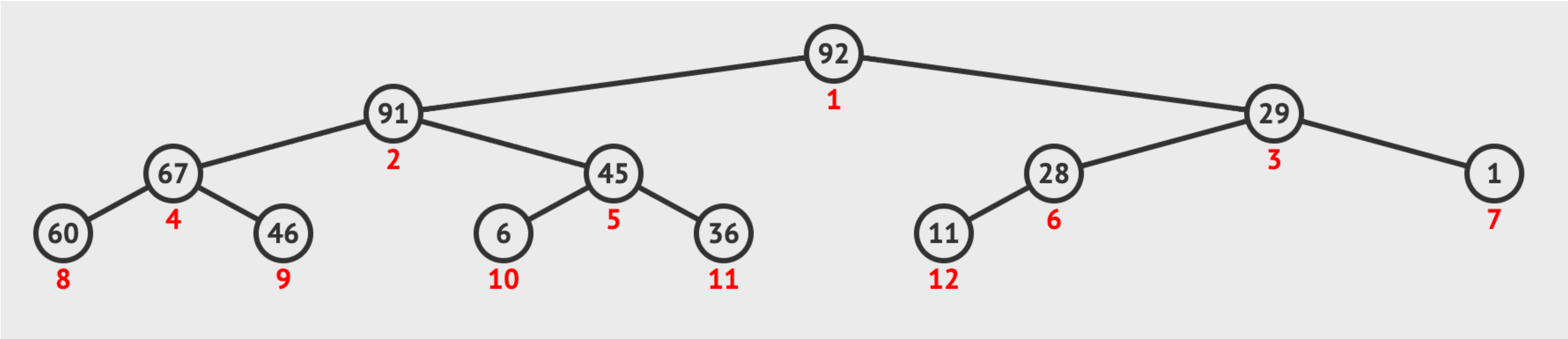
ANSWER 1

- Given the array [93,36,1,46,91,92,29,60,67,6,45,11,28], apply heap sort. Visualize what the heap will initially look like (apply the O(n) heap construction algorithm) and visualize all the steps of the sortdown.
- Heap construction

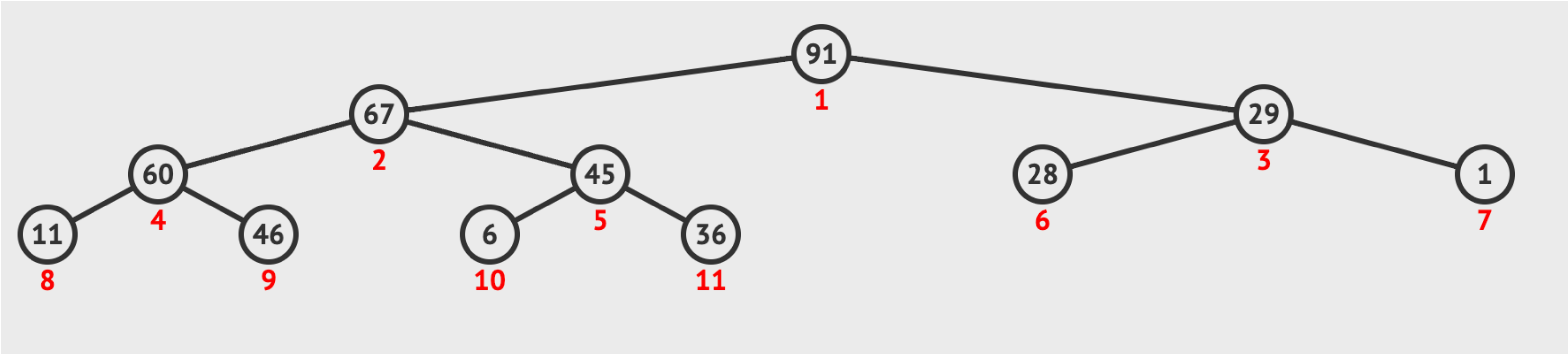


ANSWER 1

- Extract max (93)

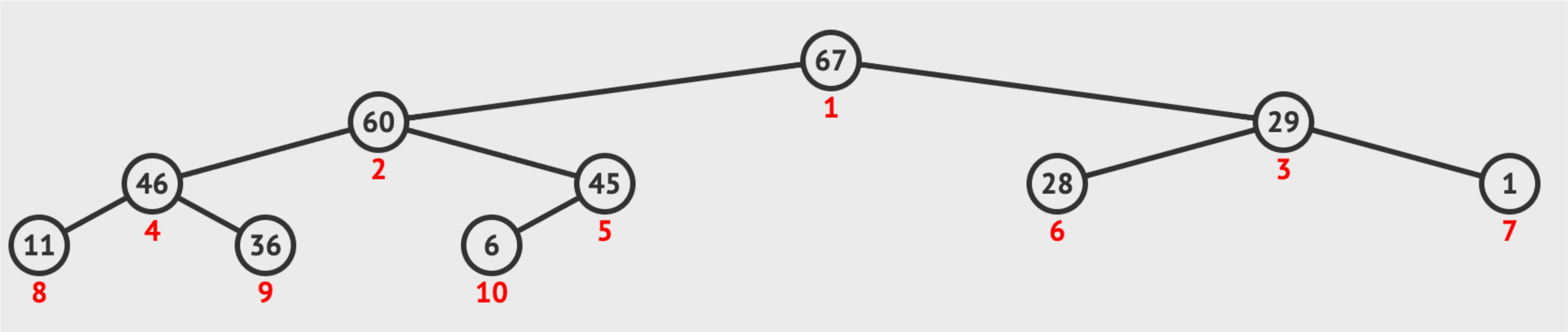


- Extract max (92)

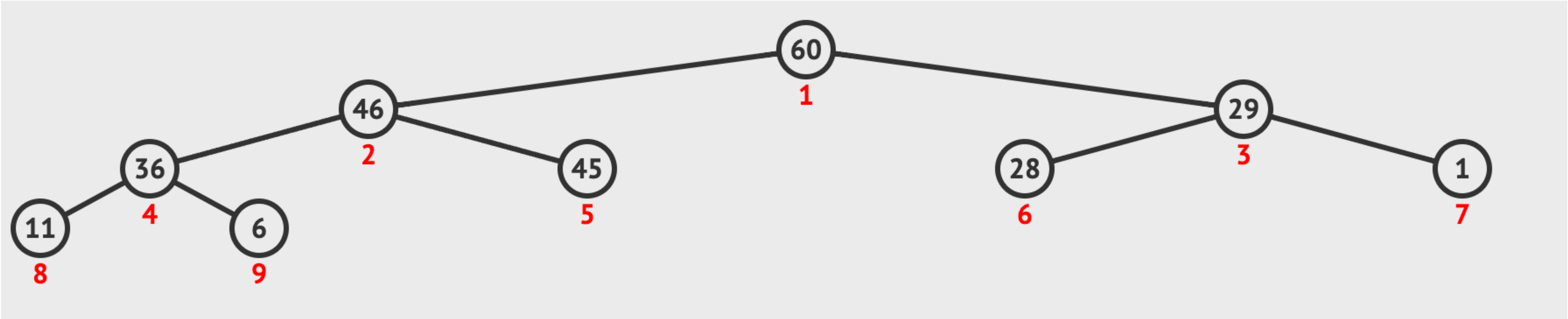


ANSWER 1

- Extract max (91)

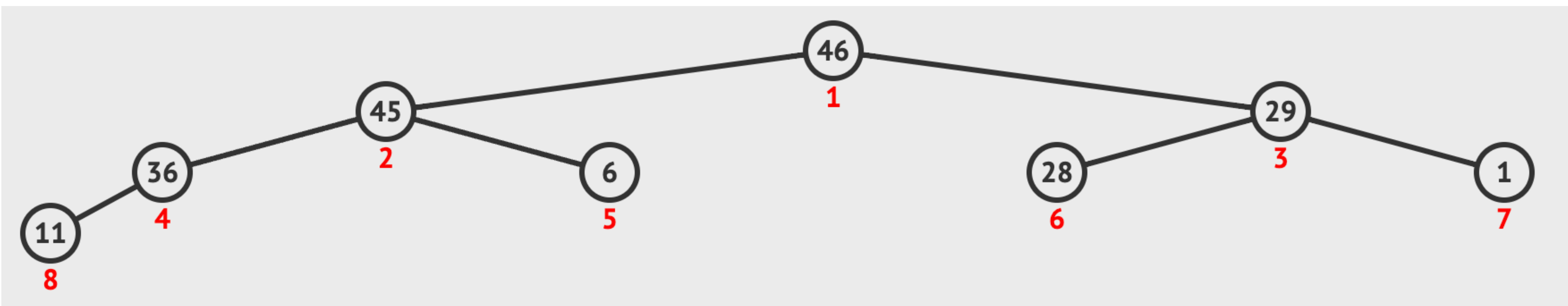


- Extract max (67)

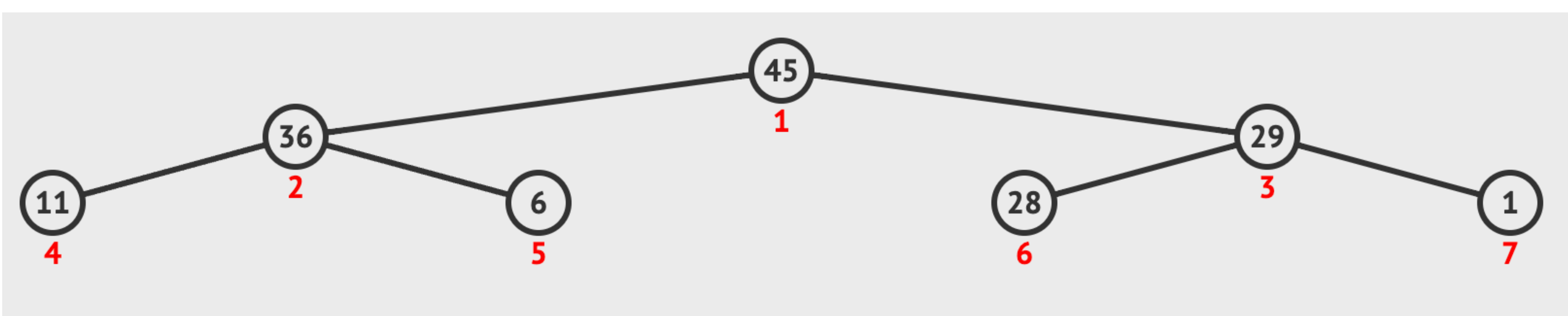


ANSWER 1

- Extract max (60)

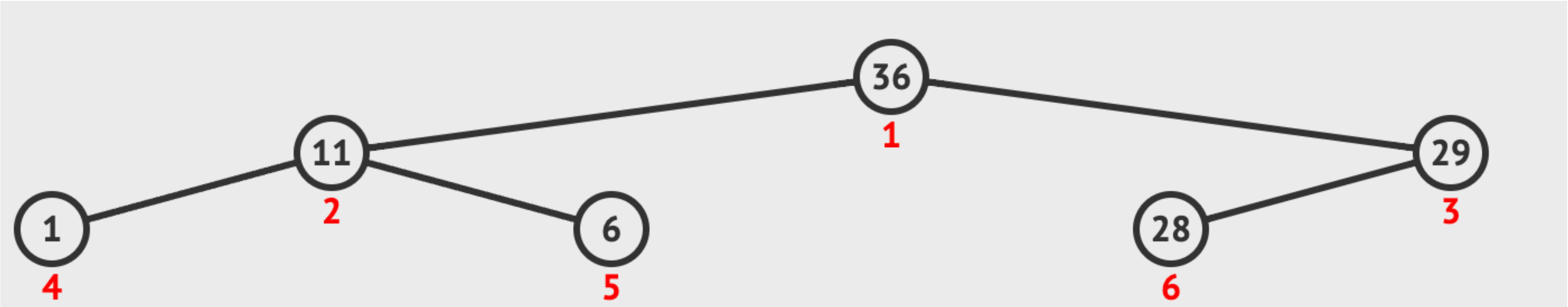


- Extract max (46)

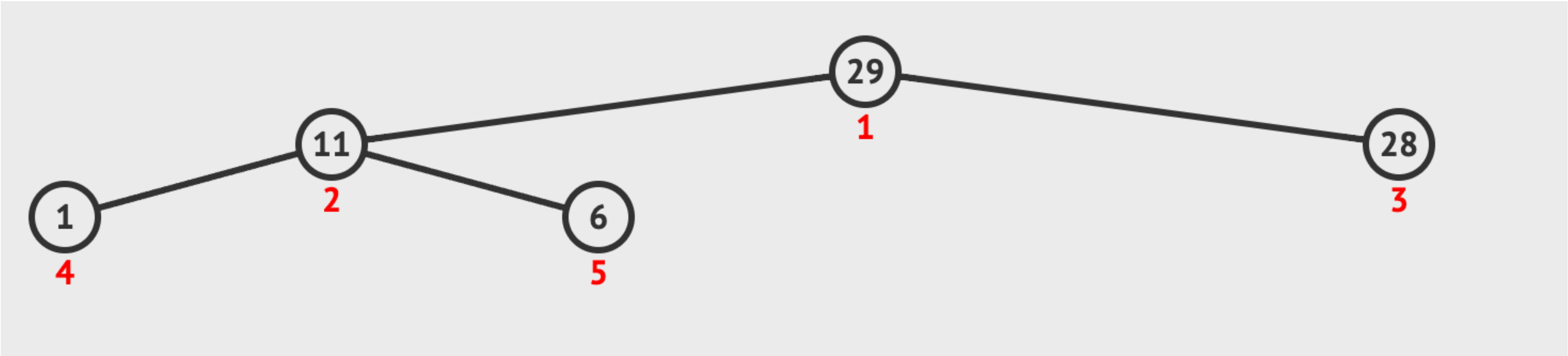


ANSWER 1

- Extract max (45)

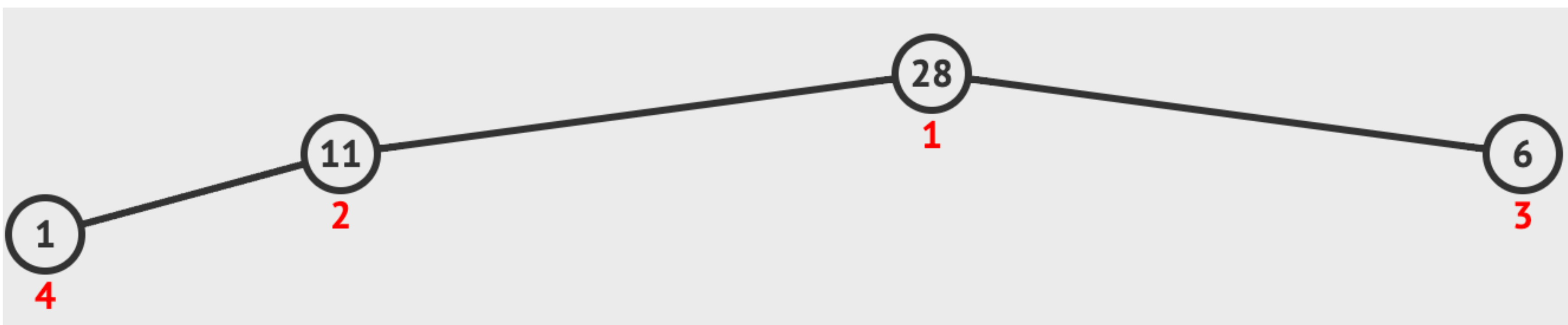


- Extract max (36)

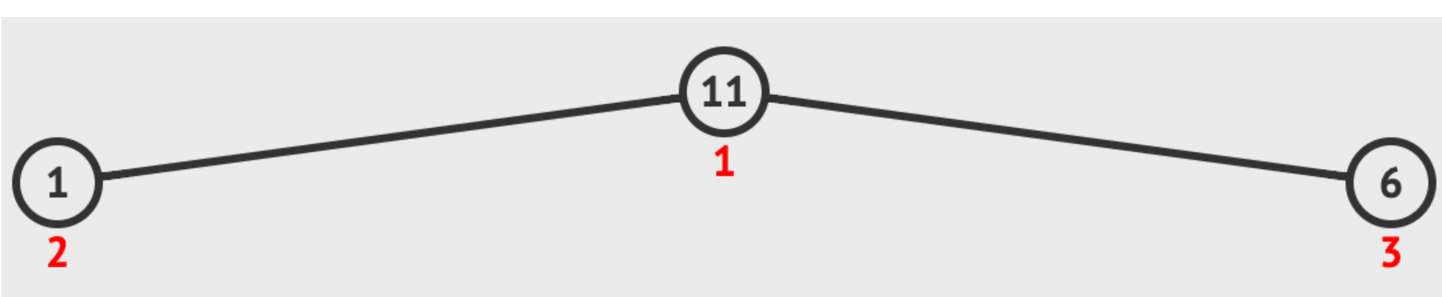


ANSWER 1

- Extract max (29)



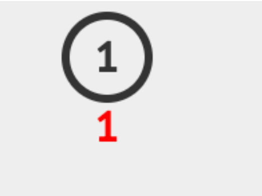
- Extract max (28)



ANSWER 1

- Extract max (11)

- Extract max (6)



- Extract max (1)