CSCI 2100 Tutorial 9

CSCI 2100 Teaching Team, Fall 2021

Outline

- A review on binary heaps (priority queues)
- Regular exercise 8 problem 4
- Special exercise 8 problem 4

Binary Heap (Review)

Let *S* be a set of *n* integers. A binary heap on *S* is a binary tree *T* satisfying:

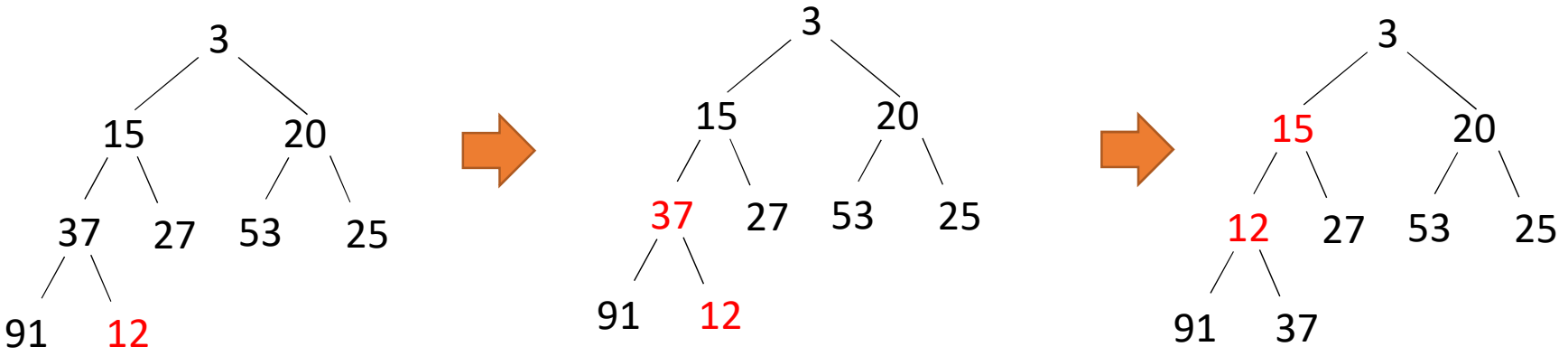
1. *T* is a complete binary tree.

2. Every node *u* in *T* stores a distinct integer in S, called the key of u.

3. If *u* is an internal node, the key of *u* is smaller than those of its child nodes.

Insertion

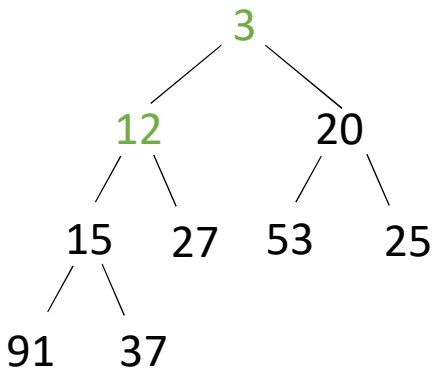
Insert 12 :



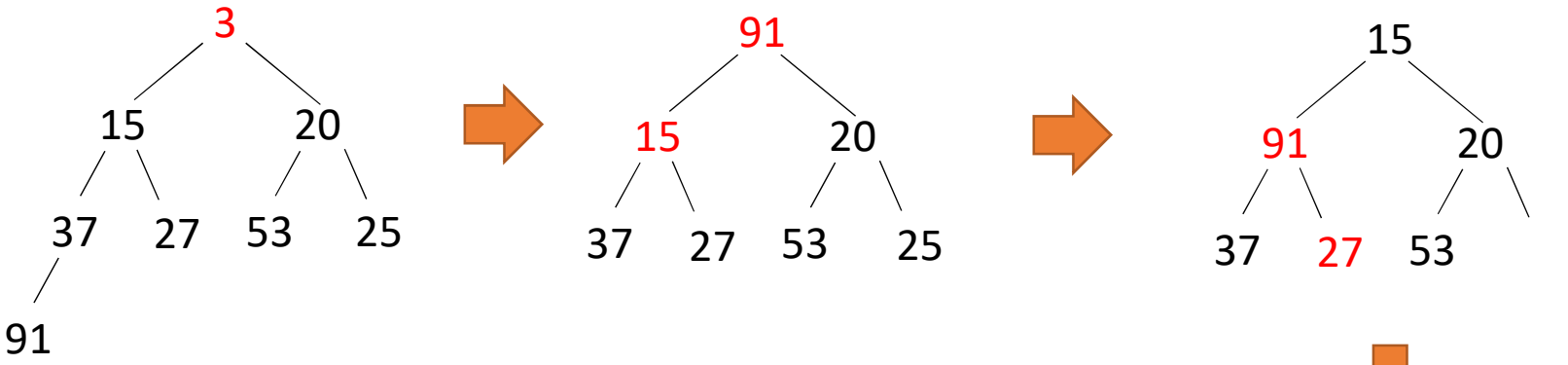
Swap up :

If node *u* has a smaller key than its parent *p*, swap the keys of *u* and *p*.

Carry out this operation bottom up $O(\log n)$ times.

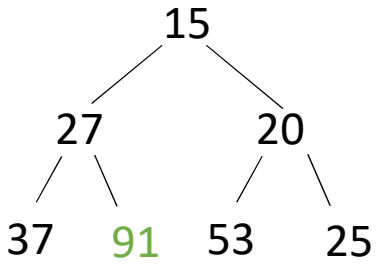


Delete-min



Swap down:

If a node u has a greater key than a child, then (i) identify the child v with a smaller key and (ii) swap the keys of u and v. Carry out this operation top down $O(\log n)$ times



Regular Exercise 8 Problem 4

Problem:

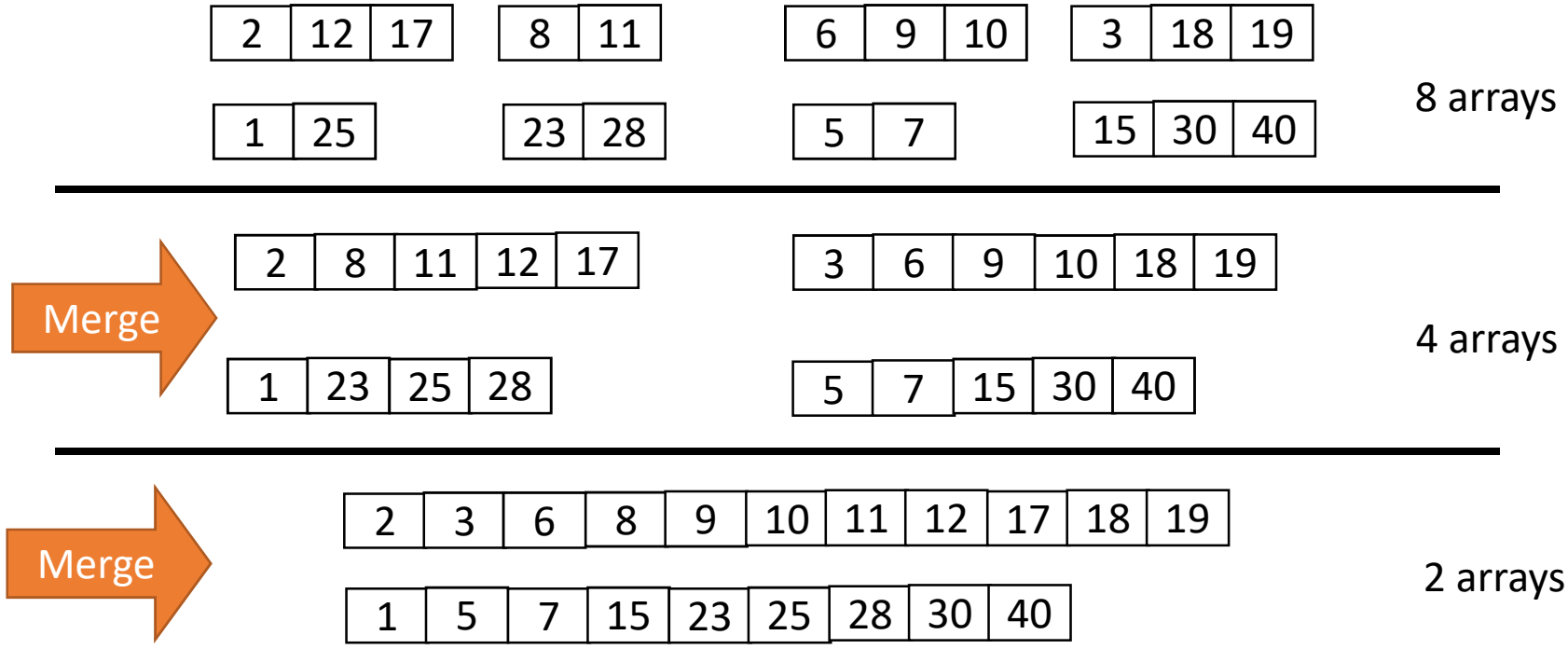
Suppose that we have *k* sorted arrays $A_1, A_2, \ldots, A_k$ of integers. Let *n* be the total number of integers in those arrays.

Describe an algorithm to produce an array that sorts all the *n* integers in $O(n \log k)$ time.

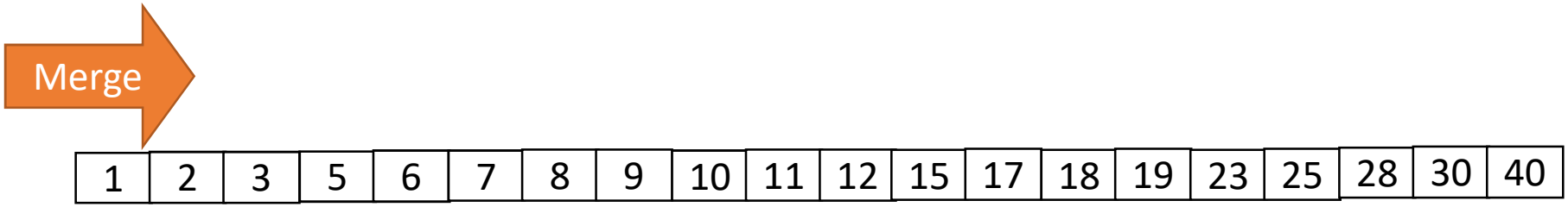
Solution 1: Merging

- Input

  k = 8, n = 20



Solution 1: Merge Operation



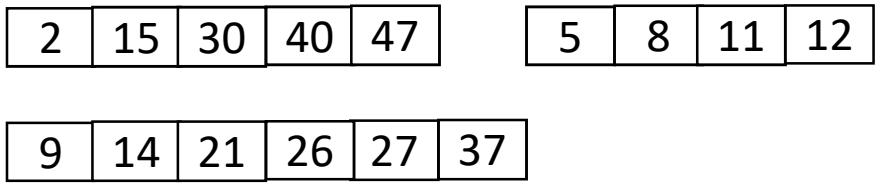
Need $O(\log k)$ passes. Each pass takes $O(n)$ time on n integers (the cost of merging is proportional to the number of elements involved).

Therefore, the total time complexity is $O(n \log k)$.

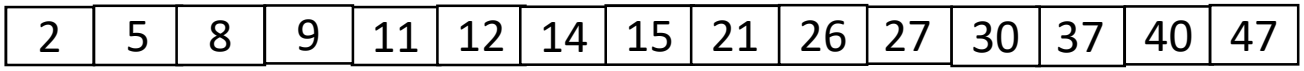
Solution 2: Binary Heap

- Input:

  k = 3, n = 15



- Output

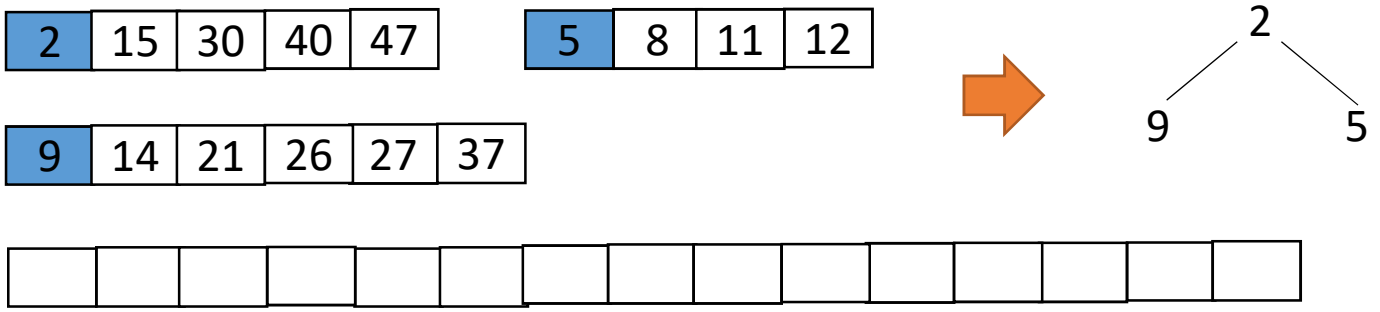


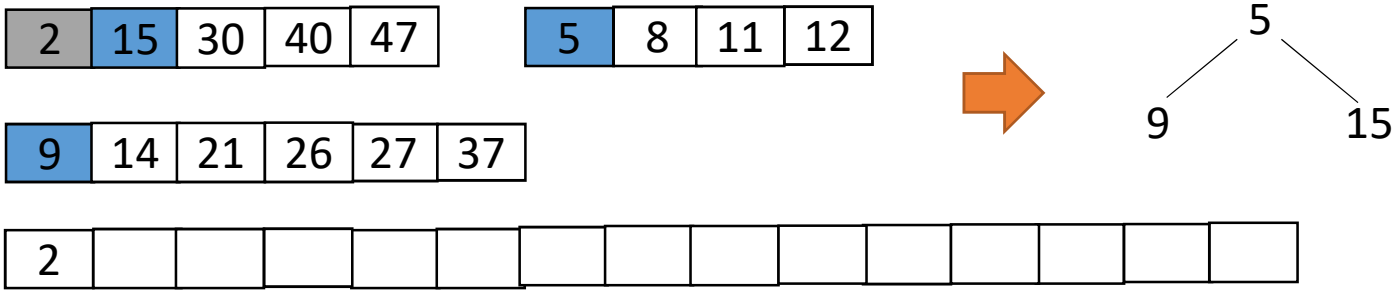
Solution 2: Binary Heap

Ideas:

- At all times, use a heap to store, for each array, the smallest element that has not been output.
- A binary heap of size k can perform delete-min and insertion in $O(log\ k)$ time.
- Perform a delete-min to obtain the next integer to output.
- After delete-min, insert a new integer into the heap from the integer's origin array.

Solution 2: Binary Heap





Solution: Binary Heap

Initialization cost:

creating the output array: $O(n)$

Processing cost:

n insertions: $O(n\log k)$ n delete-min: $O(n\log k)$

Total time complexity:

$O(n\log k)$

Special Exercise 8 Problem 4

Problem:

Let *S* be a dynamic set of integers. At first, *S* is empty. Then, new integers are added to *S* one by one, but never deleted. Let *k* be a fixed integer. Describe an algorithm that achieves the following guarantees:

- Space consumption $O(k)$.
- Insert(*e*): Insert a new element *e* into *S* in $O(log\ k)$ time.
- Report-top-*k*: Report the set of *k* largest integers in *S* in O(*k*) time.

Special Exercise 8 Problem 4

Example:

Suppose that $k = 3$, and the sequence of integers inserted is 83, 21, 66, 5, 24, 76, 92, 33, 43,…

After the insertion of 24, we should report 83, 66, 24 (in any order). After the insertion of 76, report 83, 66, 76.

Solution

Intuition:

- A min-heap *H* of size *k* takes $O(k)$ space.
- *H* performs insertion and delete-min in $O(log\ k)$

time.

- *H* always contains the *k* largest integers in *S*.
  - If the incoming integer *m* is larger than the root of *H,* perform delete-min and insert*(m)*. Otherwise, we do nothing.

Solution

- Input:

83, 21, 66, 5, 24, 76, 92, 33, 43, …, and k=3

…

Solution

Maintain a binary heap *H* with *k* integers.

1. Insert first *k* integers into *H*. Each insertion takes $O(log\ k)$ time.
2. For a newly added integer *e* from the sequence, compare it with the integer e_r stored at the root *r* of *H*:

   (1) If *e* > e_r, perform delete-min and insert*(e)*, which take $O(log\ k)$ time in total.

   (2) Otherwise, ignore *e.*

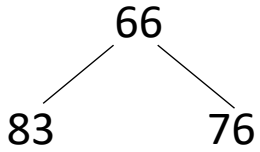
Solution

Report-top-*k*:

Report all integers in *H* by traversing the heap.

Traversal:

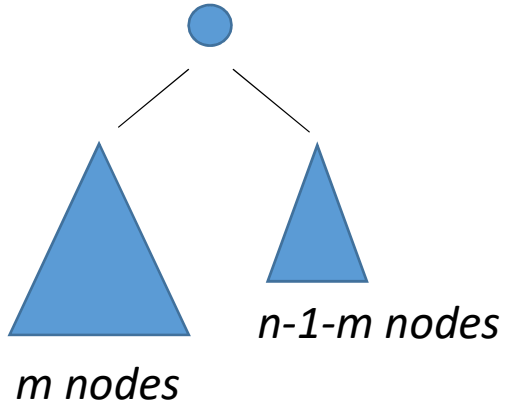
First report the root of *H*. Then report the left subtree and right subtree recursively.



Output sequence: 66, 83, 76

Solution

Cost of traversing a tree of *n* nodes:



$f(n) = O(1) + f(m) + f(n - 1 + m)$

Solving the recurrence gives $f(n) = O(n)$.

A challenging problem for you

- For this problem, we can actually achieve
  - O(*k*) space
  - $O(1)$ amortized insertion time
  - O(*k*) top-k report time.

- Hint: *k*-selection.