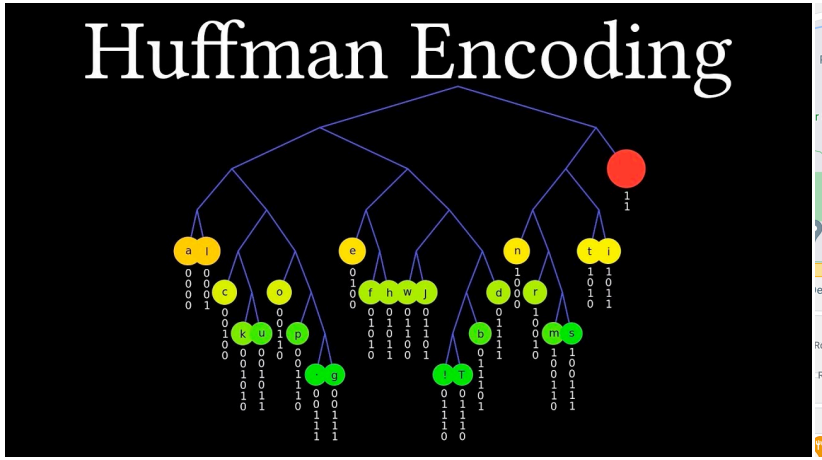
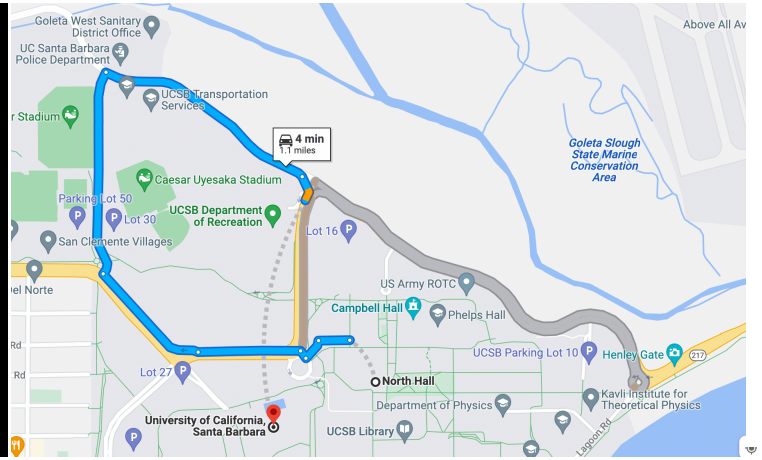
Lecture handout
https://bit.ly/CS24-binaryheap

PRIORITY QUEUES & BINARY HEAP

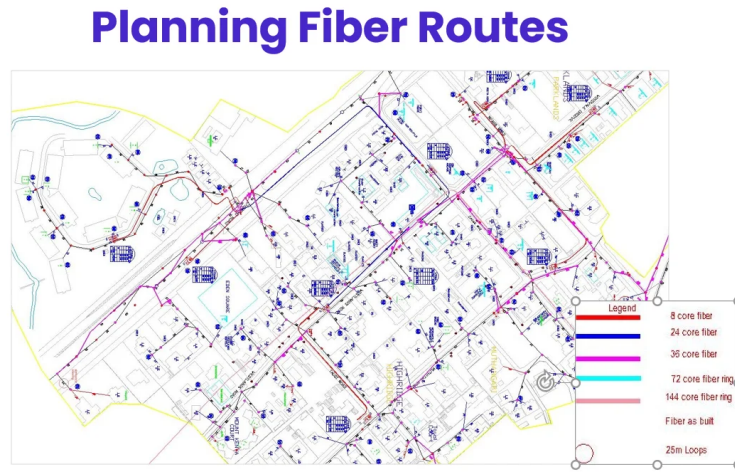
Problem Solving with Computers-II



Data Compression



Navigation



Network Design

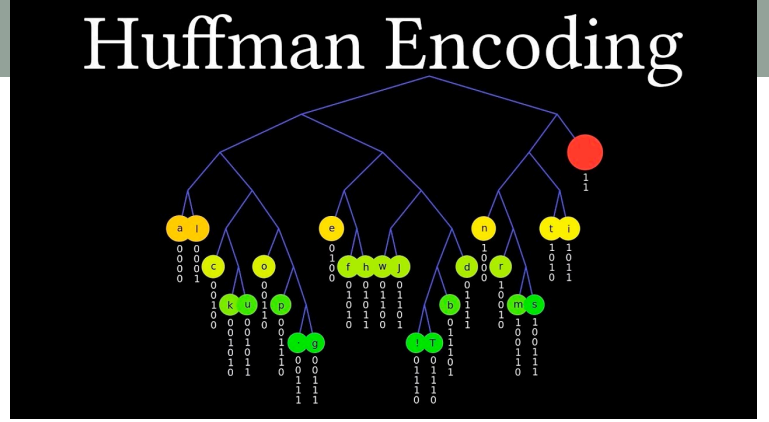
Announcements

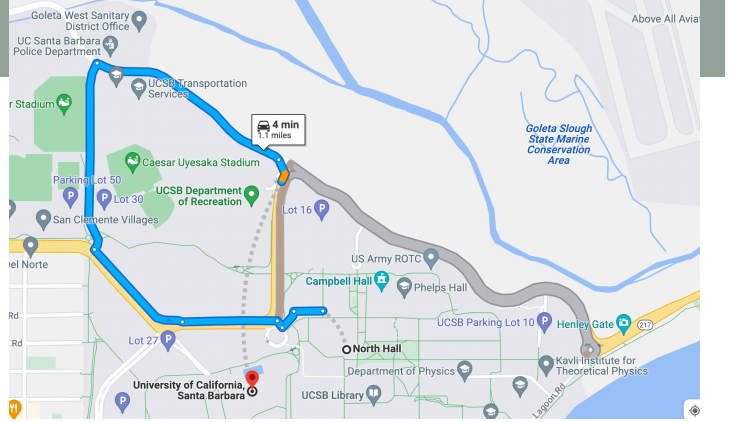
- **PA02 released**
- **Quiz 2 grades are released.**
- **Upcoming lab (lab04): implement a priority queue as a binary heap**

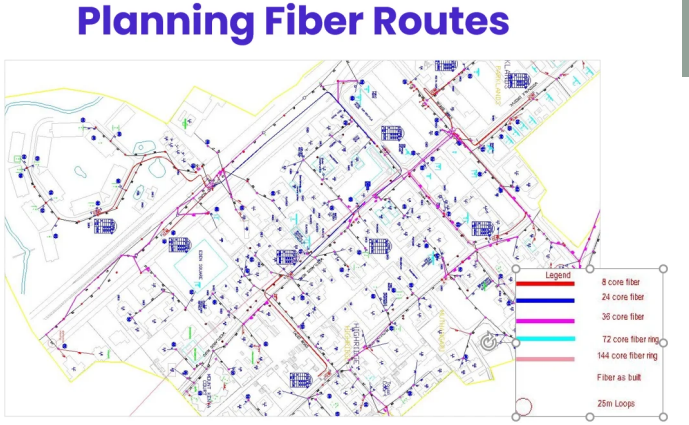
Activity 1: Review of complete binary tree

Link to visualization on VisualAlgo

- What are the keys of the children of the root (98)?
- Who is the parent of 58 (last key in the vector)?
- Who is the parent of 15?







Data Compression | **Google Maps Navigation** | **Network Design**

Algorithms: **Huffman Coding** | **Shortest Path** | **Minimum Spanning Tree**

ADT: **Priority Queue**

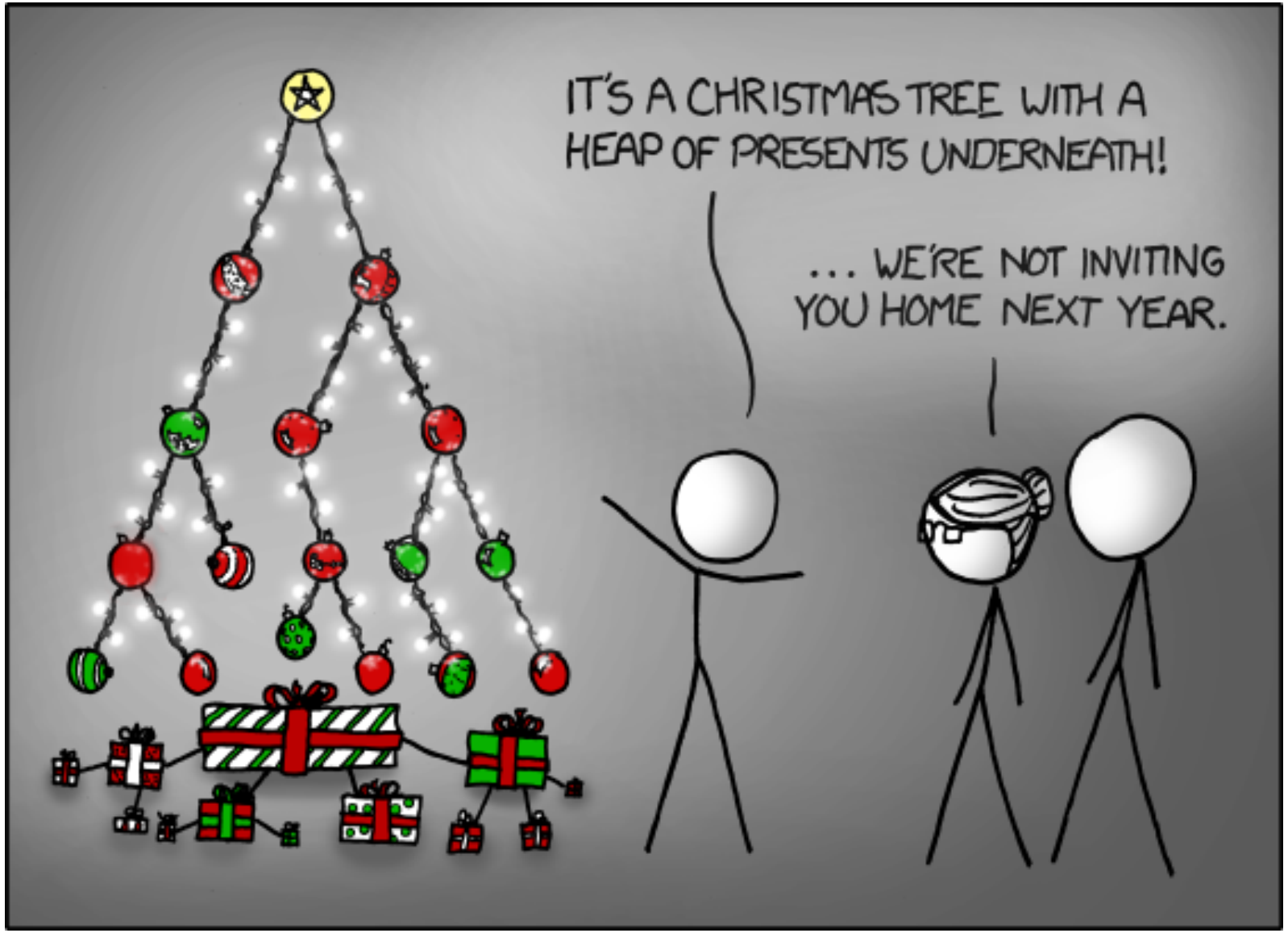
Data structure: **Binary Heap**

Complete Binary Tree

Vector

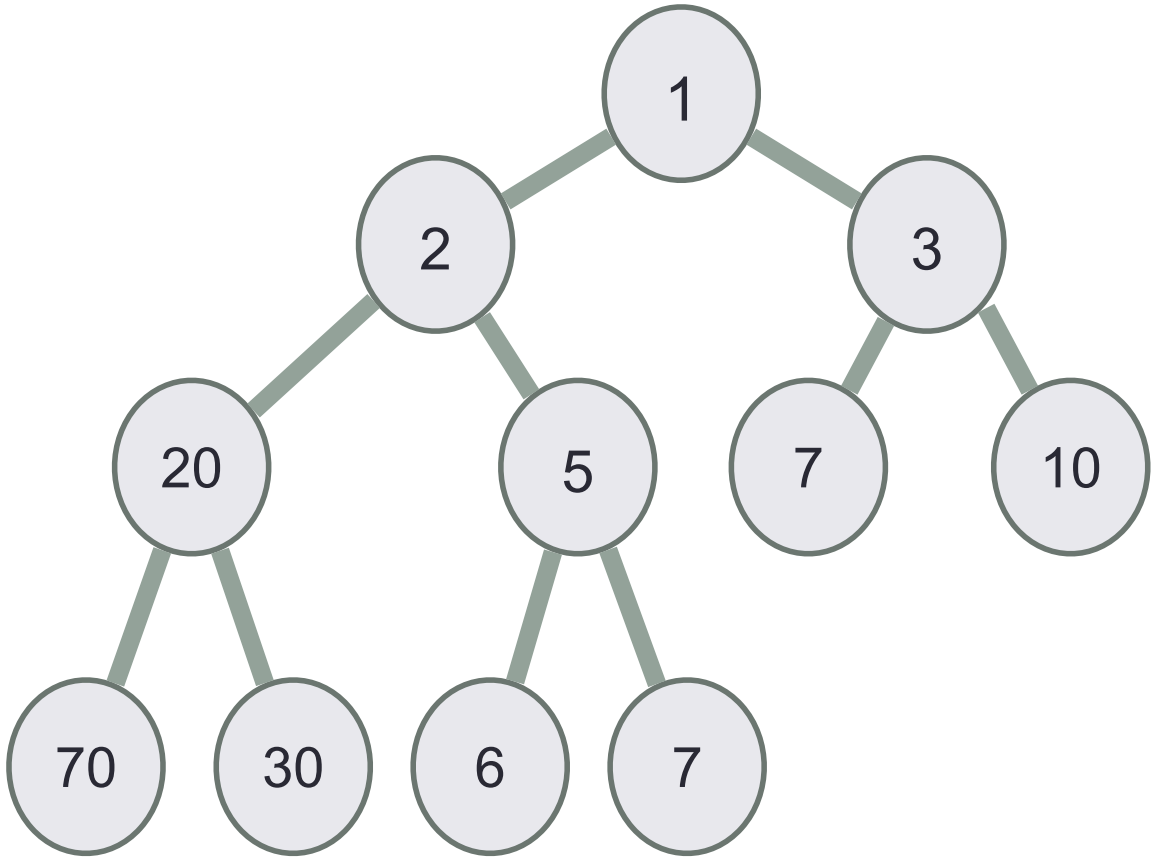
Many algorithms need to compute the min OR max repeatedly.

Priority Queue is used to speed up the running time!



Think of binary heap as a heap of presents!!

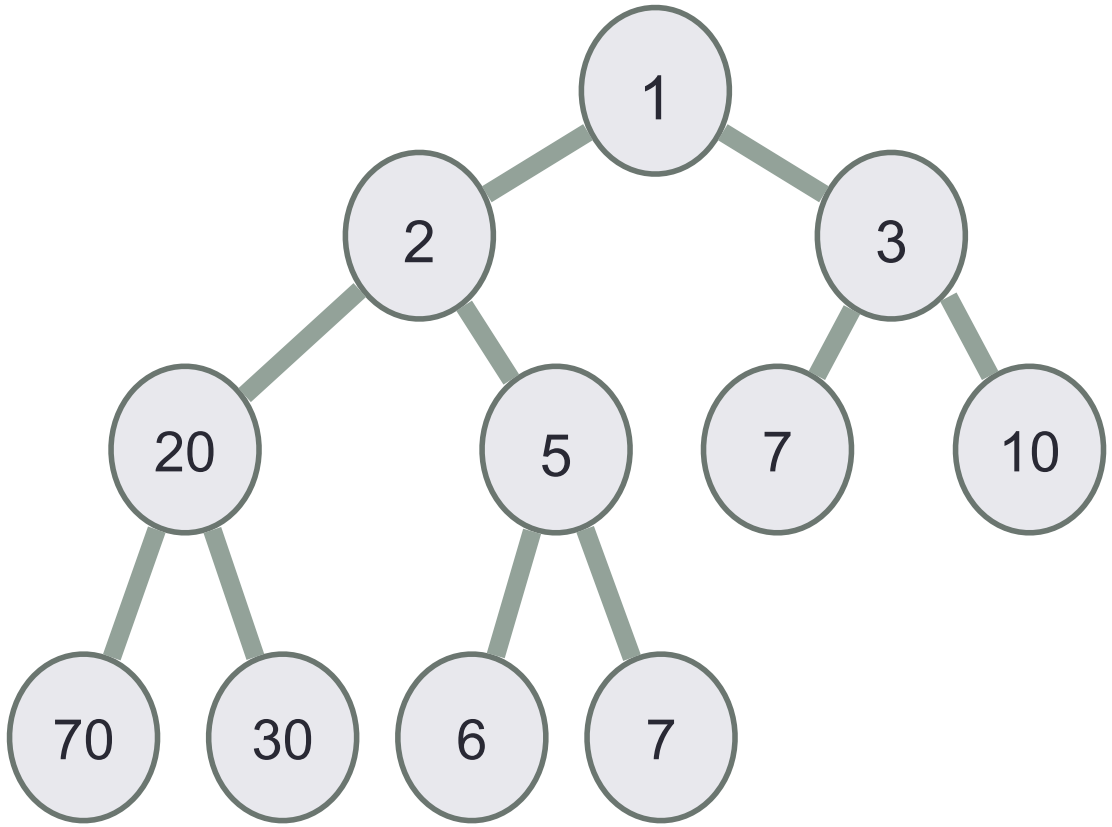
Two important properties of a binary heap tree



(1) Shape property:

(2) Heap property :

Two important properties of a binary heap tree



Example of a min-heap

(1) Shape property:

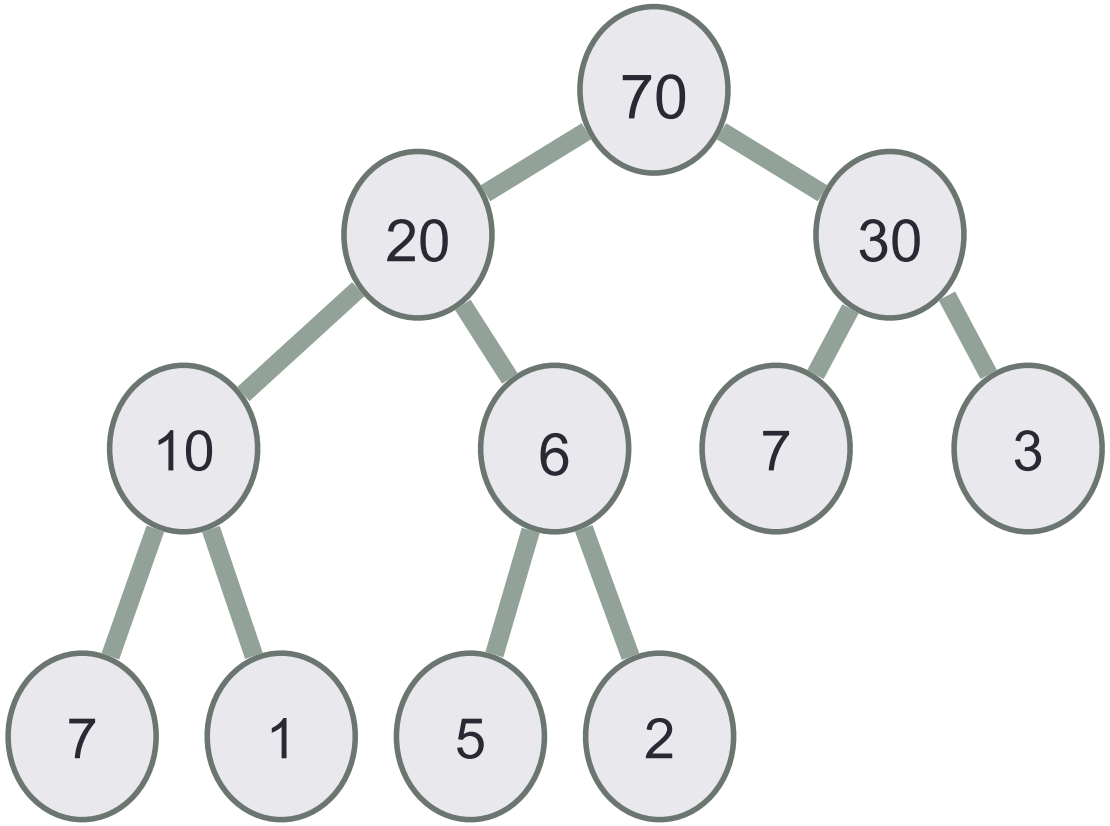
Internally, a heap is a **complete binary tree,** where each node satisfies the heap property

(2) Heap property:

In a min-heap, for each node (x):

key(x) <= key(children of x)

Two important properties of a binary heap tree



Example of a max-heap

(1) Shape property:

Internally, a heap is a **complete binary tree,** where each node satisfies the heap property

(2) Heap property:

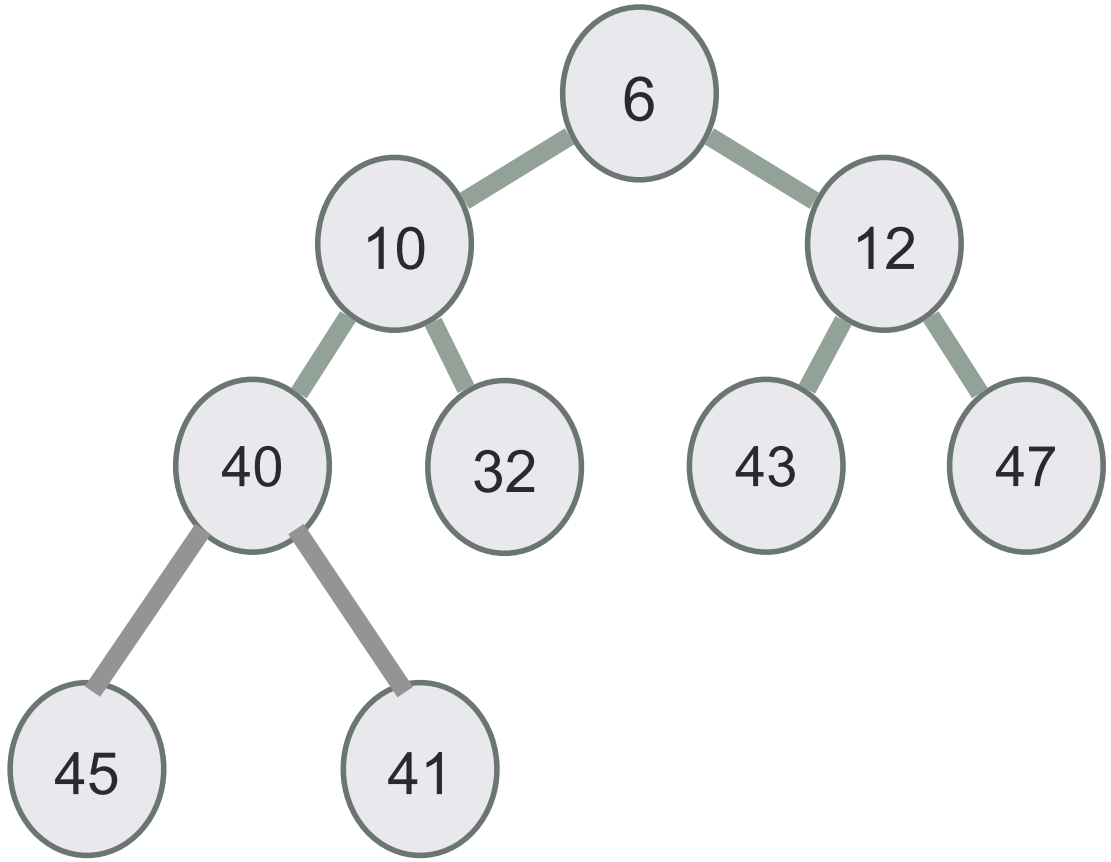
In a max-heap, for each node (x):

key(x) >= key(children of x)

Activity 2: Identifying heaps (iclicker)

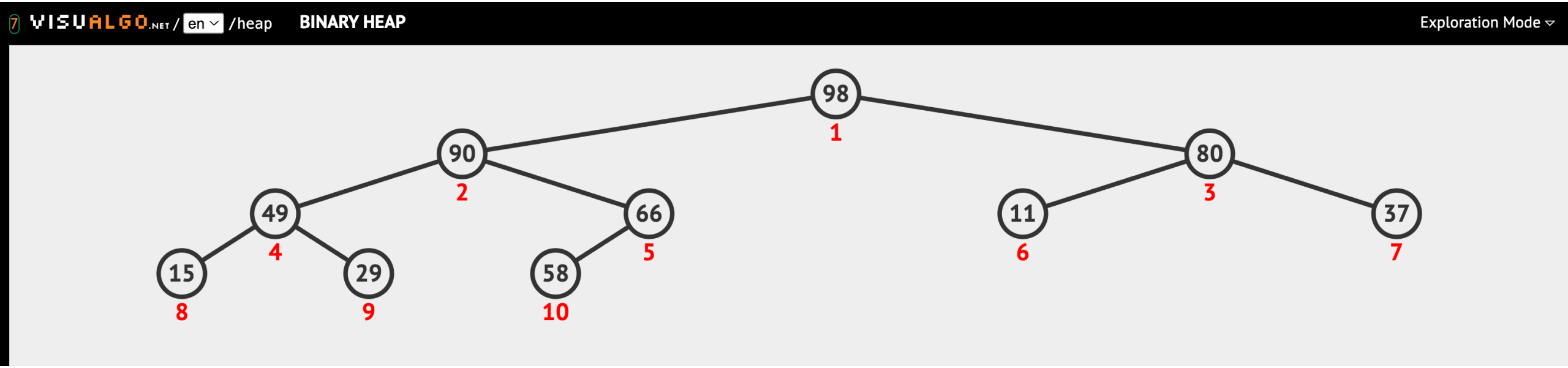
Starting with the following min-Heap which of the following operations will result in something that is NOT a min Heap

A. Swap the keys 40 and 32
B. Swap the keys 32 and 43
C. Swap the keys 43 and 40
D. Insert 50 as the left child of 45
E. C&D



Visualizing the operations of a heap

- VisualAlgo (Heap): https://visualgo.net/en/heap
- Heap Operations:
  - top(): get the root (either min or max key): O(1)
  - push(key): insert a key - O(log n)
  - pop(): remove the top element- O(log n)



Activity 3: **Practice inserting the values** 5, 7, 1, 3, 2, 20 into an initially empty max-heap. Show how keys bubble up on an insert to restore the heap property. Write the resulting vector representation of the max heap

```
procedure push(x: key value)
   insert x in the first open spot in the tree
   while(x has a parent && parent(x) < x): //Bubble up!
        swap(x, parent(x))
  done
```

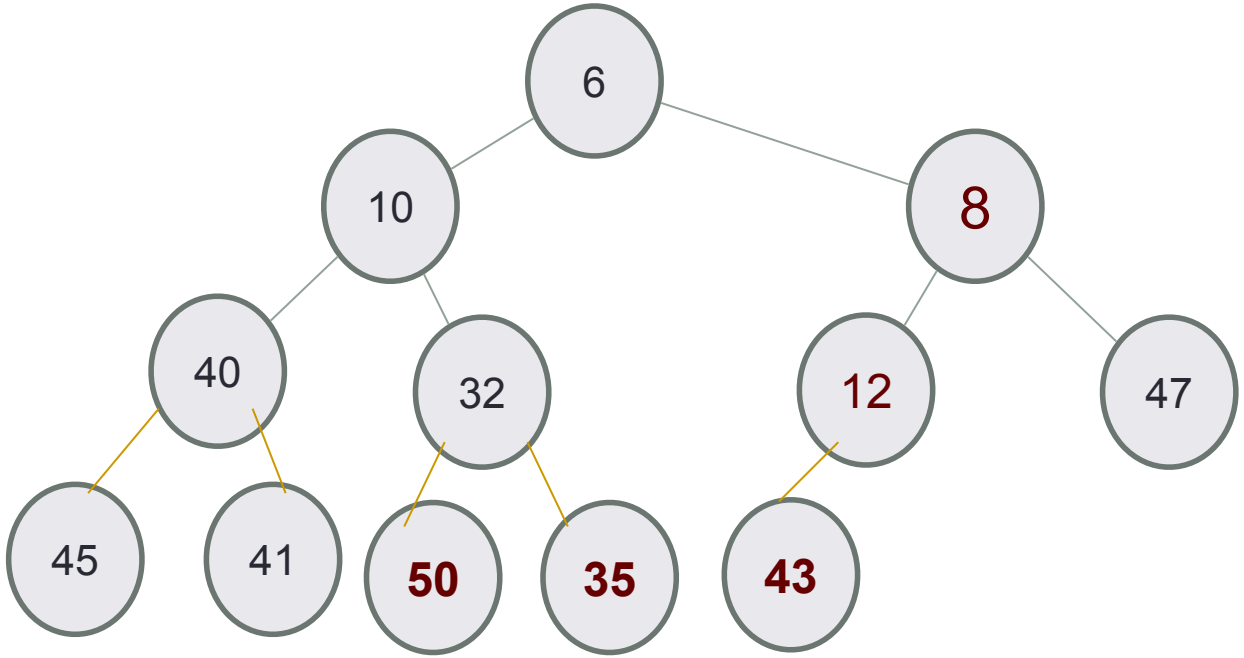
pop(): delete the key at the top()

procedure pop()

Step1: Replace the key of root with key of last node in the last level.

Step 2: Starting at the root “Bubble down” until the heap property is restored

return {the top element was deleted}



"Bubble down" in a min-heap:

- swap key of node with child that has the smallest key value until the heap property is restored

“Bubble down” in a max heap:

- swap key of node with child that has the largest key value until the heap property is restored

.

Activity 4: Starting with the vector representation of the max heap: [20, 5, 7, 3, 2, 1]. Trace the "bubble down" operation on the vector to pop() the top key.

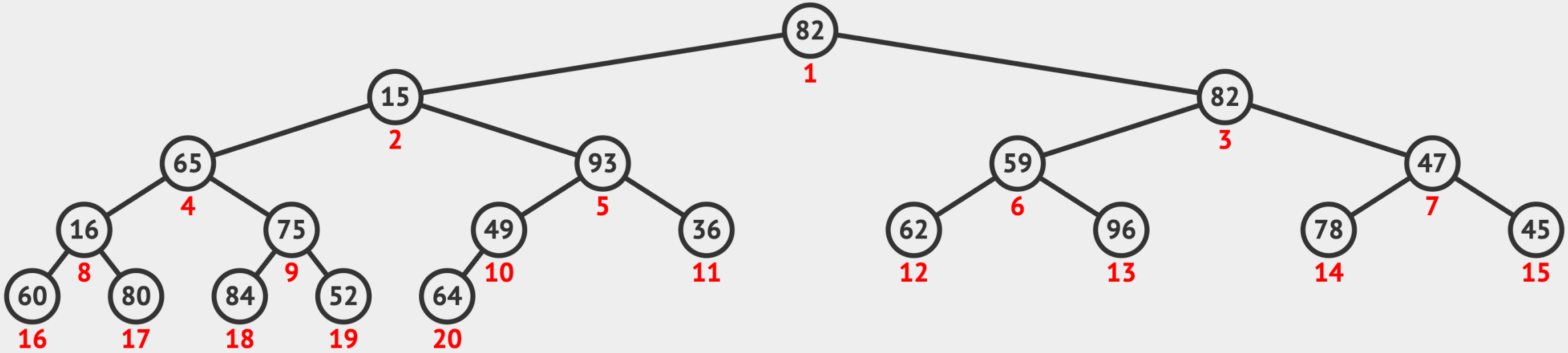
procedure pop()

Step1: Replace key of root with key of last node in the last level.

Step 2: Starting at the root “Bubble down” until the heap property is restored

return {the top element was deleted}

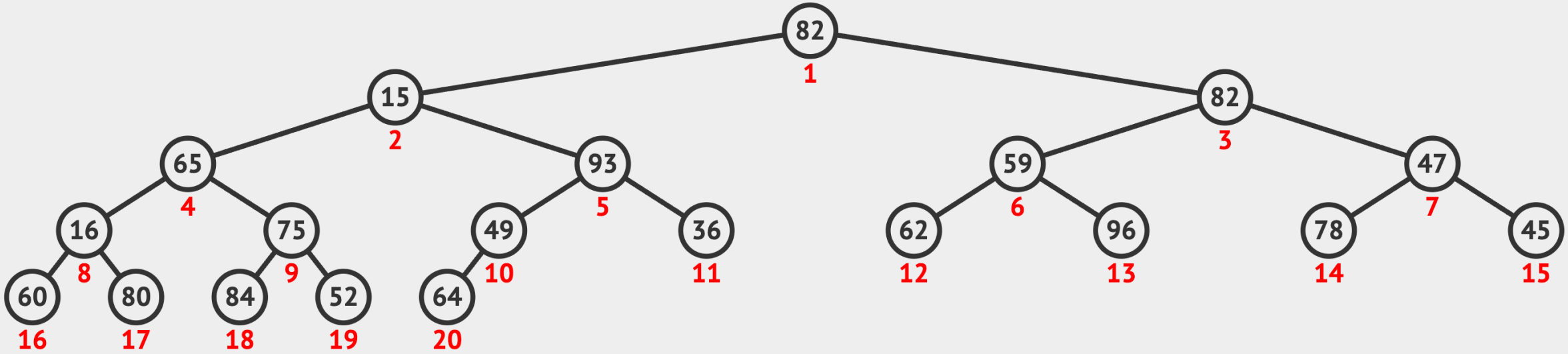
Heapify: Fast construction of a heap



Efficient Approach to Construct a Heap from vector with n keys:
Start from the last non-leaf node and work upwards to the root (index 0).
For each node, perform “bubble-down” to restore the heap property in its subtree.

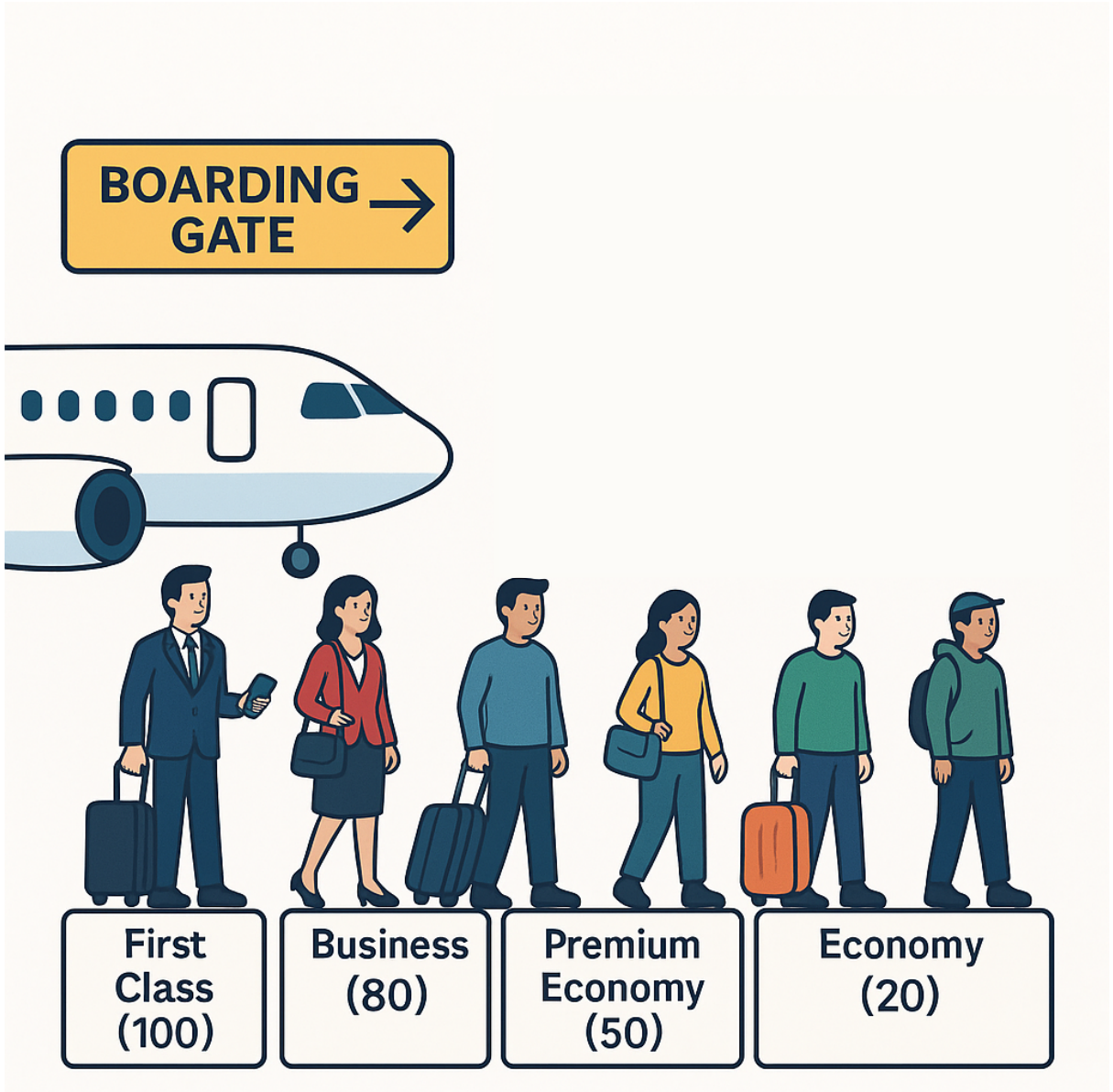
Link to visualization

Heapify: Run Time Complexity




```
void buildHeap(vector<int>& v) {
    int n = v.size();
    // Start from last non-leaf node → root
    for (int i = n/2 - 1; i >= 0; i--) {
        bubbleDown(v, i);
    }
}
```

C++ Priority Queue ADT




```
priority_queue<int> pq;

pq.push(20);
pq.push(20);
pq.push(80);
pq.push(50);
pq.push(100);

cout << pq.top();

pq.pop();
```

Configuring std::priority_queue

```
template <
      class T,
      class Container= vector<T>,
      class Compare = less <T>
      > class priority_queue;
```

The template for priority_queue takes 3 arguments:

1. Type elements contained in the queue.
2. Container class used as the internal store for the priority_queue, the default is **vector<T>**
3. Class that provides priority comparisons, the default is **less**

Configuring std::priority_queue

```
//Configure for a max-heap
priority_queue<int, vector<int>, std::less<int>> pq;

//Configure for a min-heap
priority_queue<int, vector<int>, std::greater<int>> pq;
```

215. Kth Largest Element in an Array

Medium | Topics | Companies

Given an integer array `nums` and an integer `k`, return *the* `k^th` *largest element in the array*.

Note that it is the `k^th` largest element in the sorted order, not the `k^th` distinct element.

Can you solve it without sorting?

Example 1:

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```

Example 2:

```
Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4
```

Activity (10 mins): Brainstorm and describe a possible solution in plain English

Trace your approach on the example inputs

Kth Largest Element in an Array (medium):
https://leetcode.com/problems/kth-largest-element-in-an-array/description/

Leetcode practice (LP04)

LP04 (PQ + Hashtables): https://ucsb-cs24.github.io/s25/lp/lp04/

Priority Queues must know problems:

1. Kth Largest Element in an Array (medium): https://leetcode.com/problems/kth-largest-element-in-an-array/description/

2. Top K Frequent Elements (medium): https://leetcode.com/problems/top-k-frequent-elements/description/

* Practice configuring a PQ in different ways using a comparison class