Data Structures and Algorithms

Lecture 8: Priority Queue and Heapsort

Department of Computer Science & Technology
United International College

Outline

- Trees and Binary Heaps
- Priority Queue
  - insert
  - deleteMin
- Heapsort

Motivating Example

3 jobs have been submitted to a printer in the order A, B, C.

Job A	100 pages
Job B	10 pages
Job C	1 page

Average waiting time with FIFO:

(100+110+111) / 3 = 107 time units

Average waiting time with shortest-job-first:

(1+11+111) / 3 = 41 time units

Queue

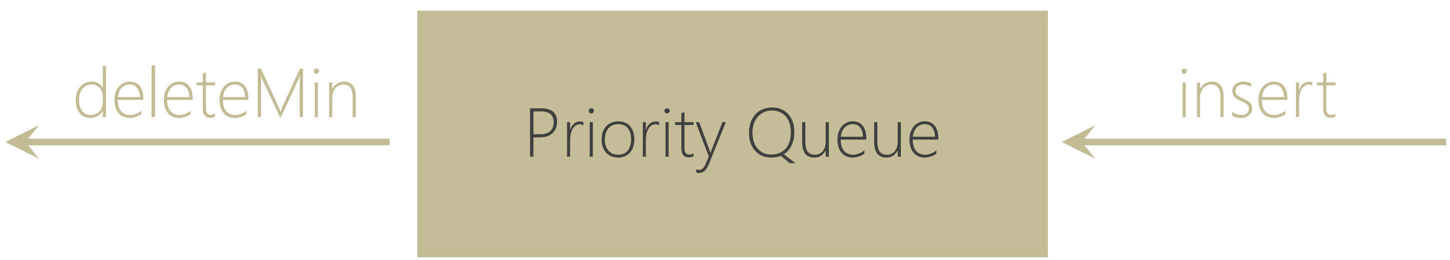
What Data Structure?

A queue capable of insert and deleteMin:

PRIORITY QUEUE

Priority Queue

- Priority queue is a data structure which allows at least two operations
  - `insert`
  - `deleteMin`
    - finds, returns and removes the minimum elements in the priority queue



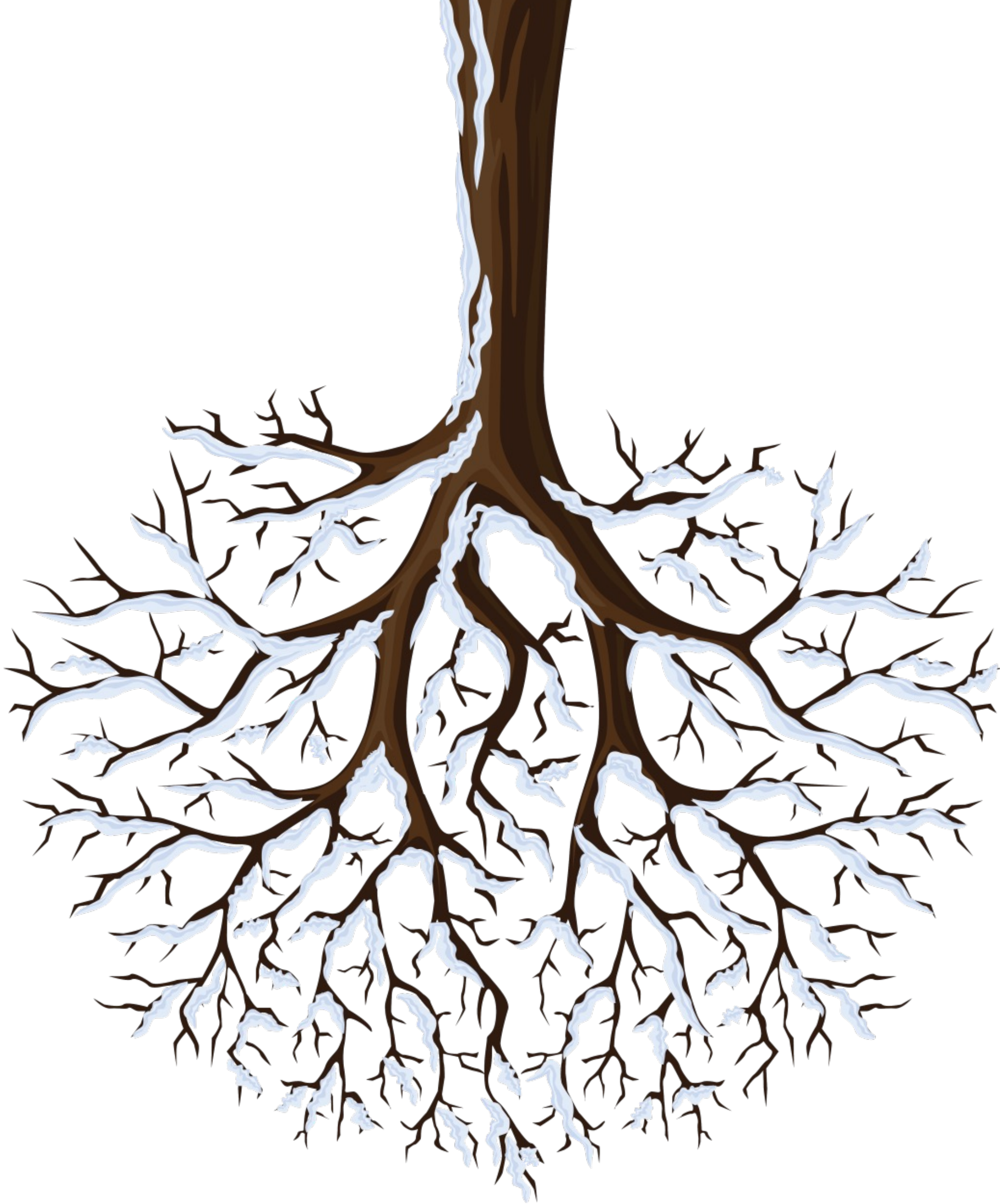
- Applications: external sorting, greedy algorithms

Implementation

- Linked List or Array?
  - insert: O(1)
  - deleteMin: O(n)

Too Slow

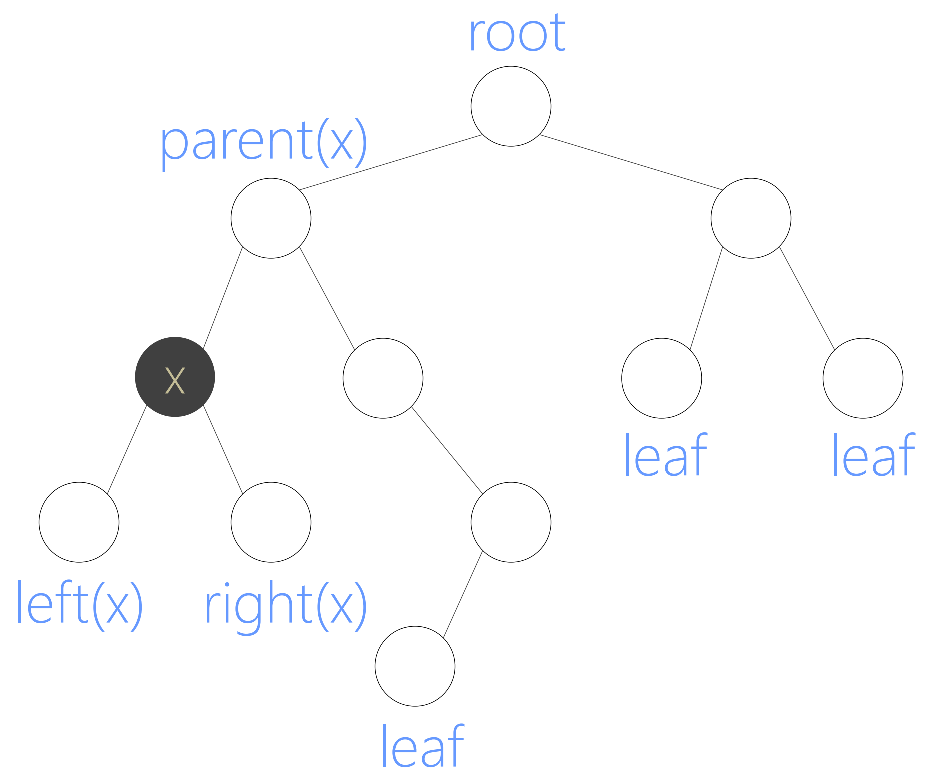
- Goal
  - insert: O(log(n))
  - deleteMin: O(log(n))



Implementation: Trees

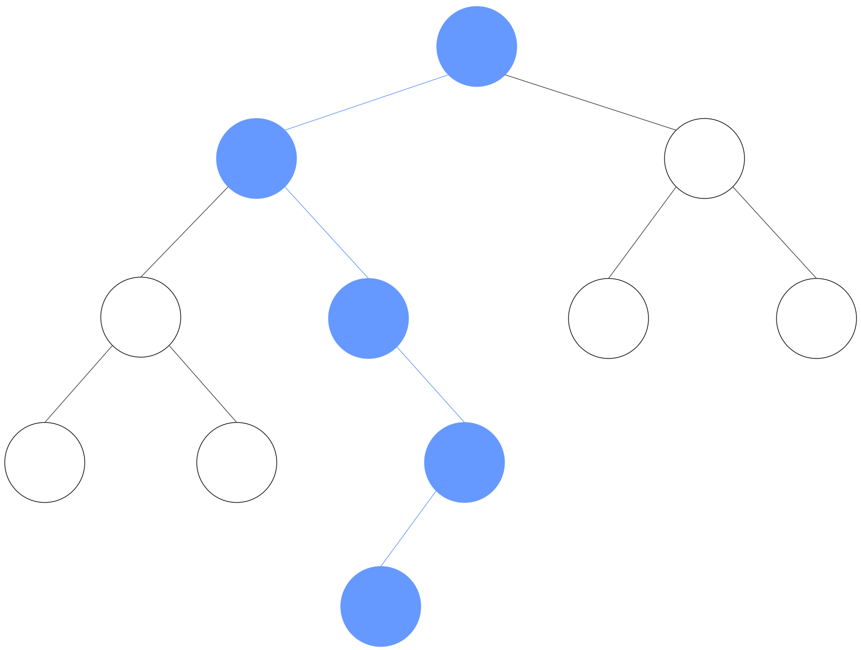
Background: Binary Trees

- Has a root at the topmost level
- Each node has zero, one or two children
- A node that has no child is called a leaf
- For a node x, we denote the left child, right child and the parent of x as left(x), right(x) and parent(x), respectively.



Tree Height (Depth)

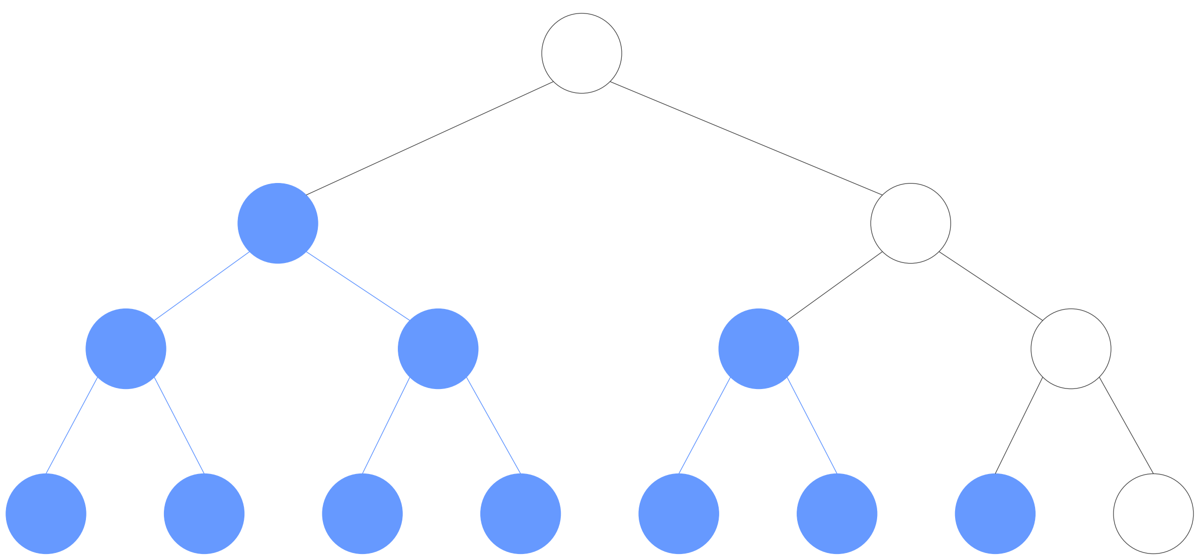
- The number of edges on the longest path from the root to a leaf



height = depth = 4

Perfect Binary Trees

- A perfect binary tree is the tree
  - where a node can have 0 or 2 children and
  - all leaves are at the same depth

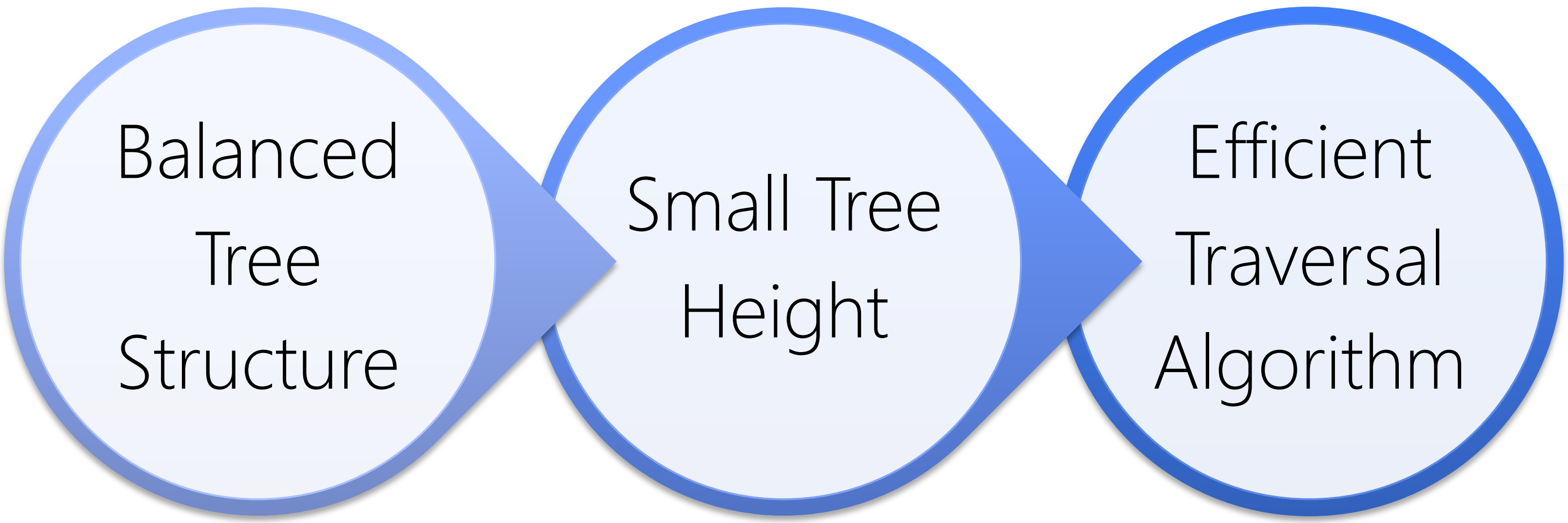


height	# of nodes
0	1
1	3
2	7
3	15
d	$2^{d+1} - 1$

Property

- A perfect binary tree with n nodes has height of $O(\log(n))$
  - Implication: If the number of node access within an algorithm is bounded by the tree height, then its complexity is $O(\log(n))$!
- Side notes
  - The largest depth of a binary tree of n nodes is $O(n)$
  - What is the shape of the tree?

Given the number of nodes (data size):



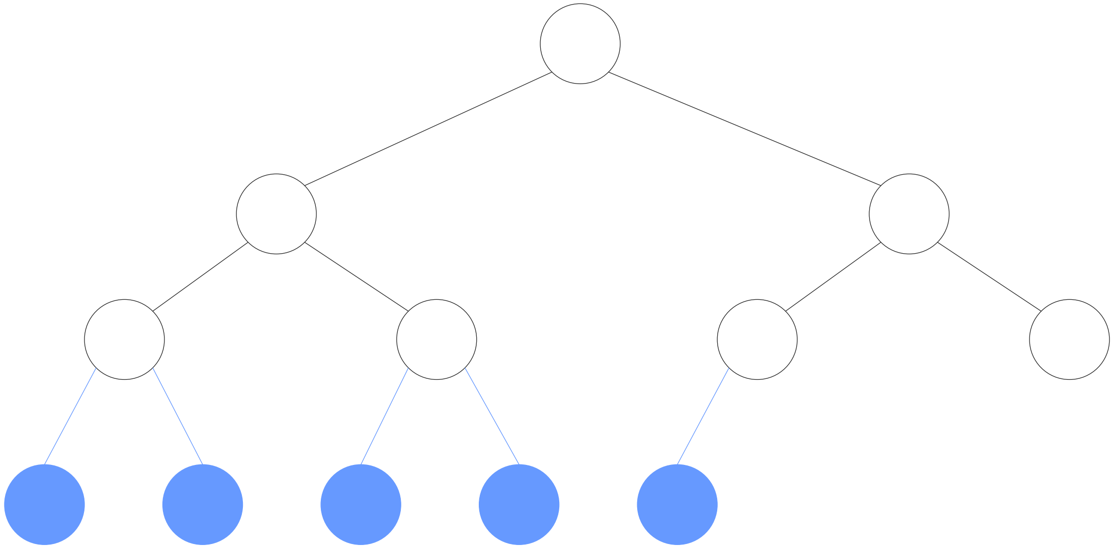
It is always wise to keep a tree balanced.

Not all node sets, however, can form a perfect tree. Hence, we propose

BINARY HEAPS

Binary Heap

- Heaps are "almost perfect binary trees"
  - All levels are full except possibly the lowest level
  - If the lowest level is not full, then nodes must be packed to the left



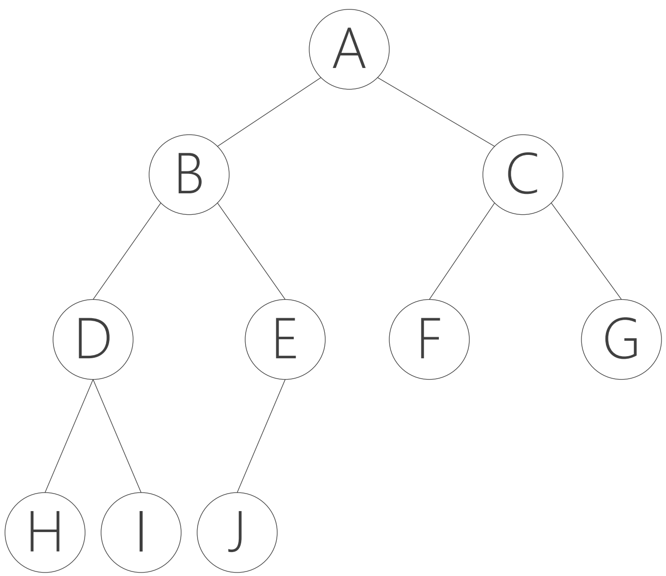
No "hole" from the first node to the last

Property

- Given a binary heap of node number n and height h
  - n is within $[2^h, 2^{h+1}-1]$
  - The height $h=O(\log(n))$
  - The structure is so regular, it can be represented in an array and no links are necessary !!!

Array Implementation of Binary Heaps

Concept: A tree



Implementation: An Array

A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9

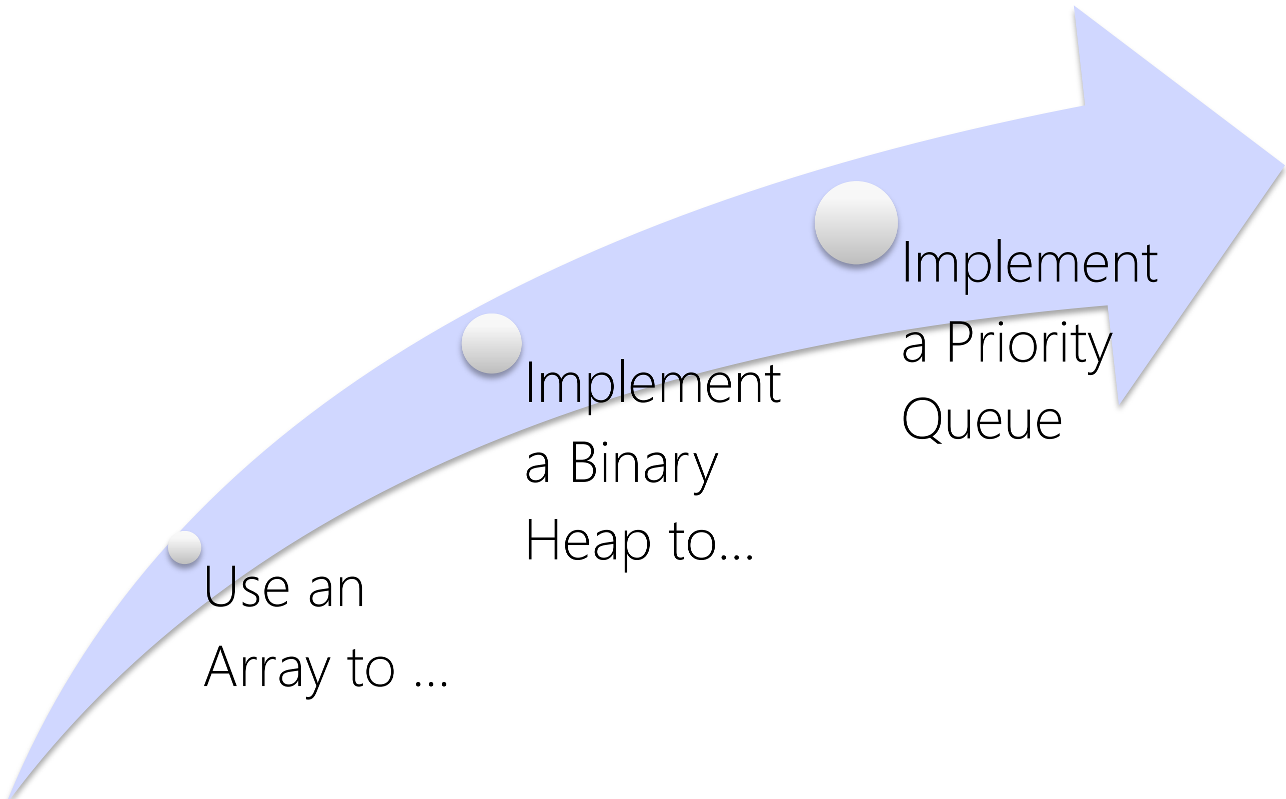
Given a node x at position i

- left(x) is at position 2i+1
- right(x) is at position 2i+2
- parent(x) is at position (i-1)/2

Side Notes

- It's not wise to store normal binary trees in arrays, coz it may generate many holes

Implementation Flow

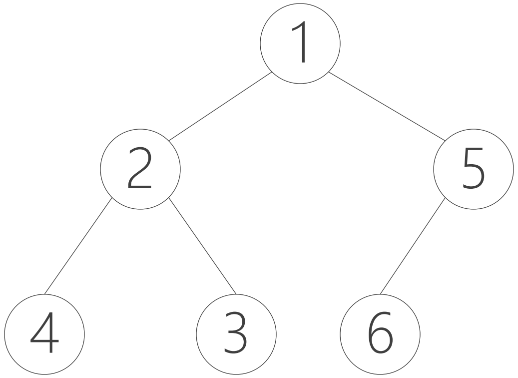


A MINIMUM PRIORITY QUEUE HAS

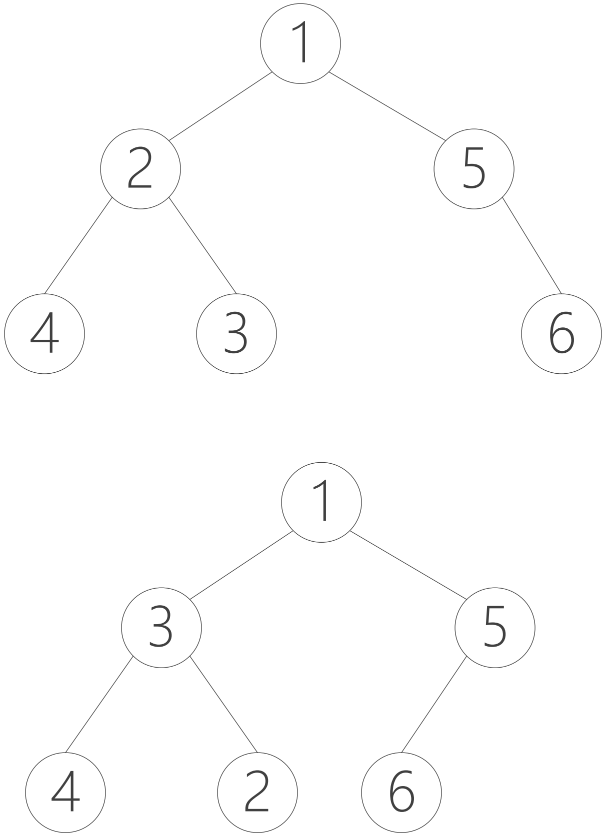
1. Binary Heap Structure
2. Heap Order Property
   - The value at each node is less than or equal to the values at both its descendants
   - The smallest node is always on the top

Use of binary heap is so common for priority queue implementations, thus the word heap is usually assumed to be the implementation of the data structure

A Heap



Not Heaps (WHY?)



Heap Properties

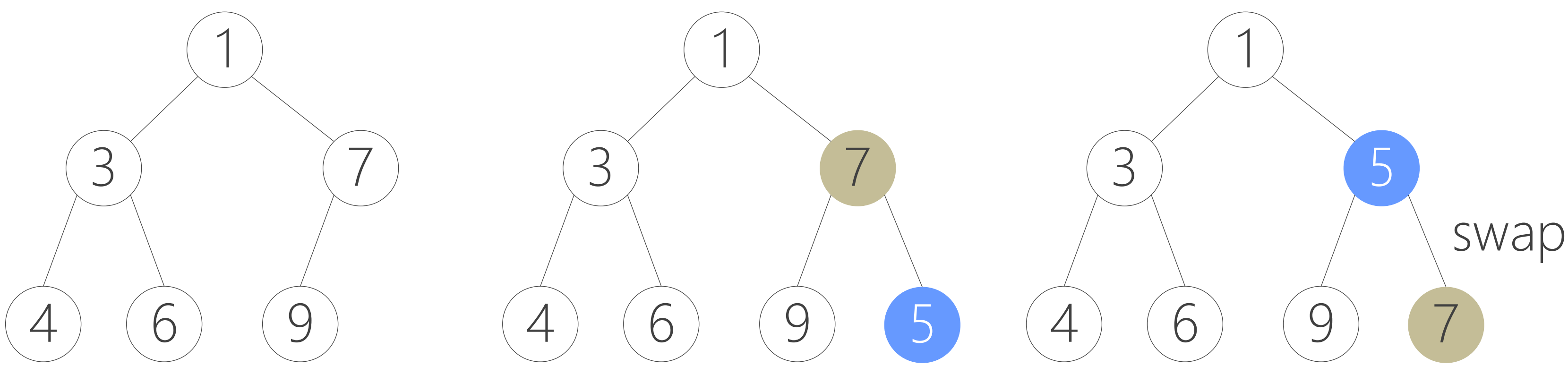
- Heap supports the following operations efficiently
  - Insert in O(logN) time
  - Locate the current minimum in O(1) time
  - Delete the current minimum in O(log N) time
- Note: After each insert/delete operation, the heap must remain a heap

Heap Implementation

MinHeap
- A: int[] - size: int
+ insert(int x): boolean + deleteMin(): int + ...

Insertion

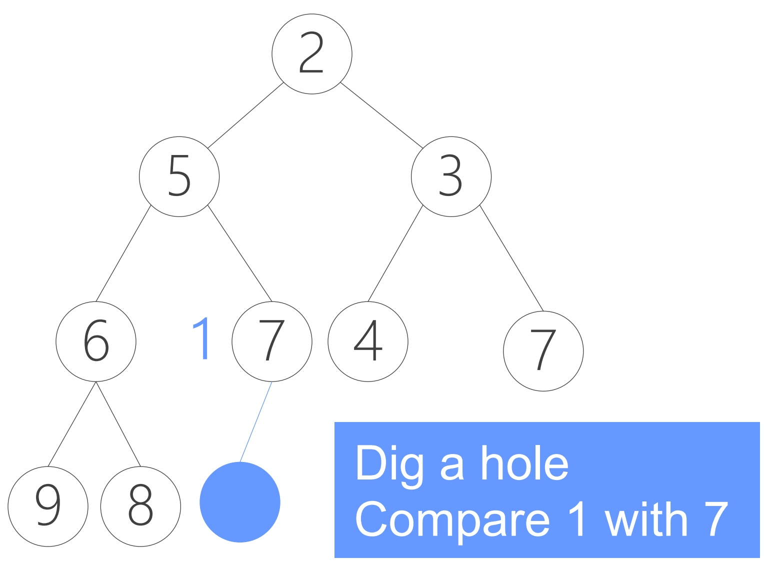
- Algorithm
  1. Add the new element to the next available position at the lowest level
  2. Restore the min-heap property if violated
     - General strategy is percolate up: if the parent of the element is larger than the element, then interchange the parent and child.

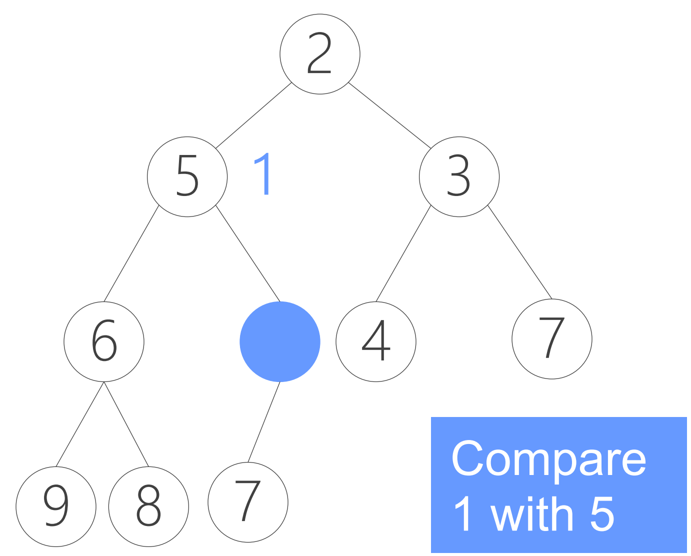


Insert 5

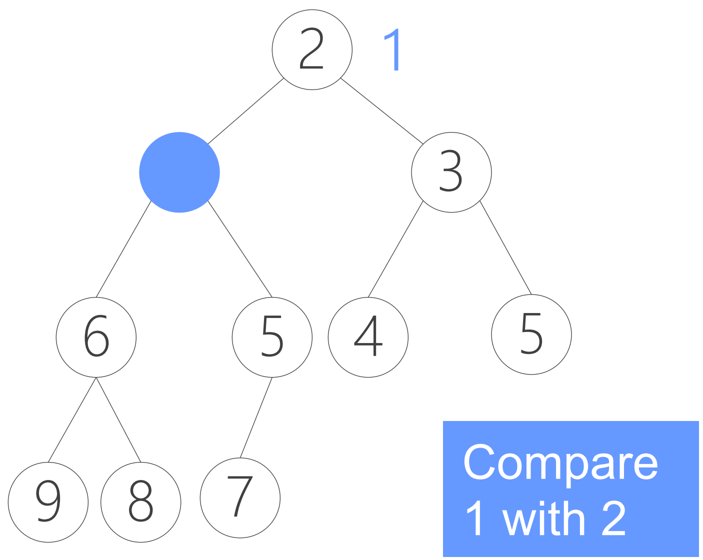
An Implementation Trick

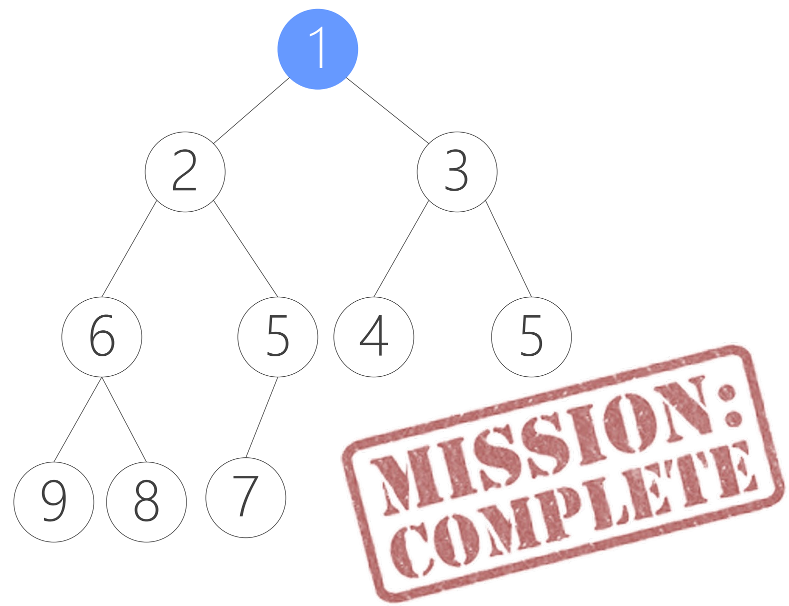
- Implementation of percolation in the insert routine
  - by performing repeated swaps: 3 assignment statements for a swap.
  - 3d assignments if an element is percolated up d levels
  - An enhancement: Hole digging with d+1 assignments
- Insert 1…





Insertion Complexity





Time Complexity = O(height) = O(logN)

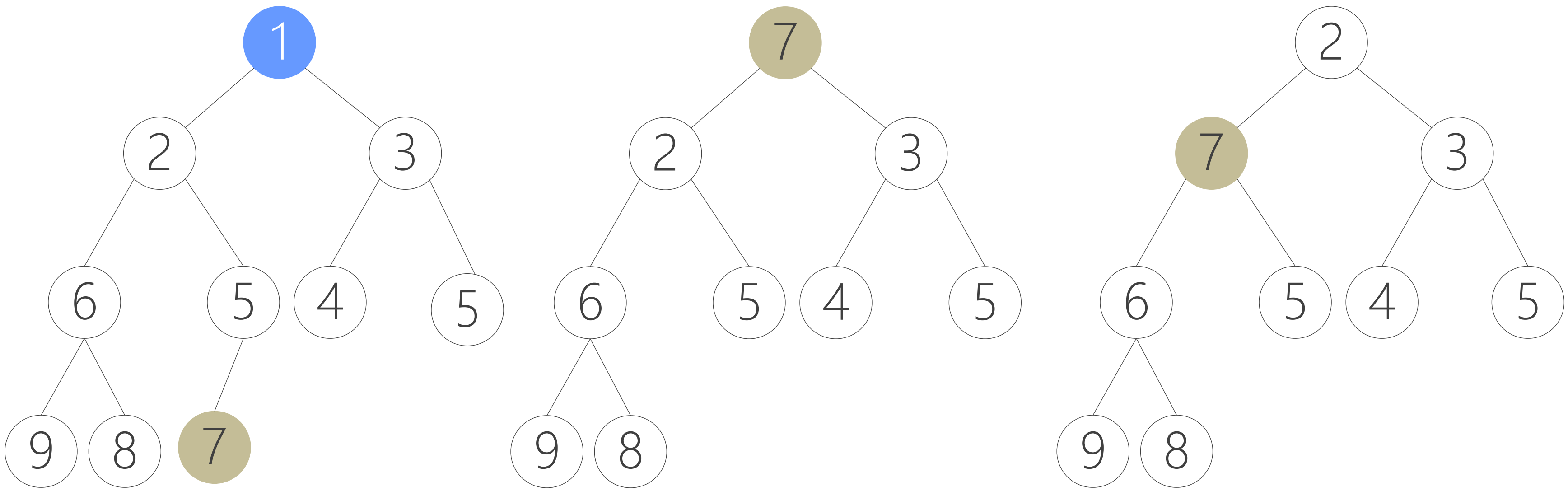
Insertion Pseudo Code

```
insert(x)
IF ISFULL(A)
    return False
// percolate up
hole = size ++
WHILE hole>0 AND x<A[(hole-1)/2]
    A[hole] = A[(hole-1)/2]
    hole = (hole-1)/2
A[hole] = x
return True
```

deleteMin

- Same Strategy
  - First, maintain binary heap structure
    - Replace the root with the value of the last node
  - Then, maintain heap order property
    - Percolate down

deleteMin Example

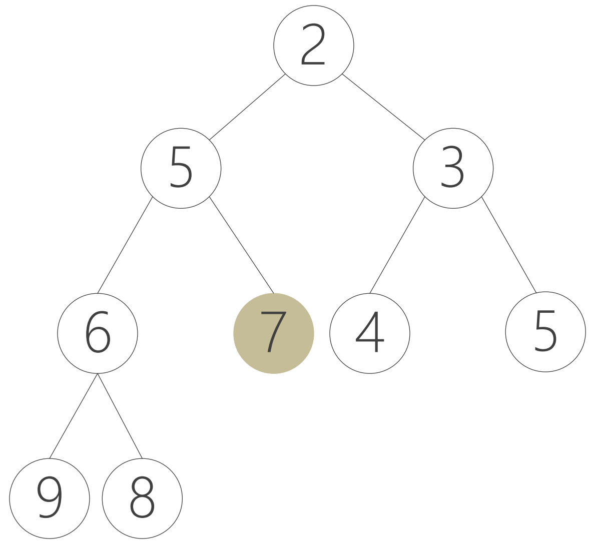


deleteMin() called

replace the root

swap with the smallest in the family

deleteMin Example



swap with the
smallest in the family

The “dig a hole” trick can be applied on delete, too!

HOLE TRICK

deleteMin Pseudo Code

```
deleteMin()
IF ISEMPTY(A)
    return -1
min = A[0], hole = 0, x=A[--size]
// percolate down
WHILE A[hole] has children
    sid = index of A[hole]'s smaller child
    IF x<=A[sid]
        BREAK
    A[hole] = A[sid]
    hole = sid
A[hole] = x
return min
```

A sorting algorithm based on heaps

HEAPSORT

Heapsort

1. Build a binary heap of n elements
   – the minimum element is at the top of the heap

O(nlog(n)) time

2. Perform n DeleteMin operations
   – the elements are extracted in sorted order

O(nlog(n)) time

3. Record these elements in a second array and then copy the array back

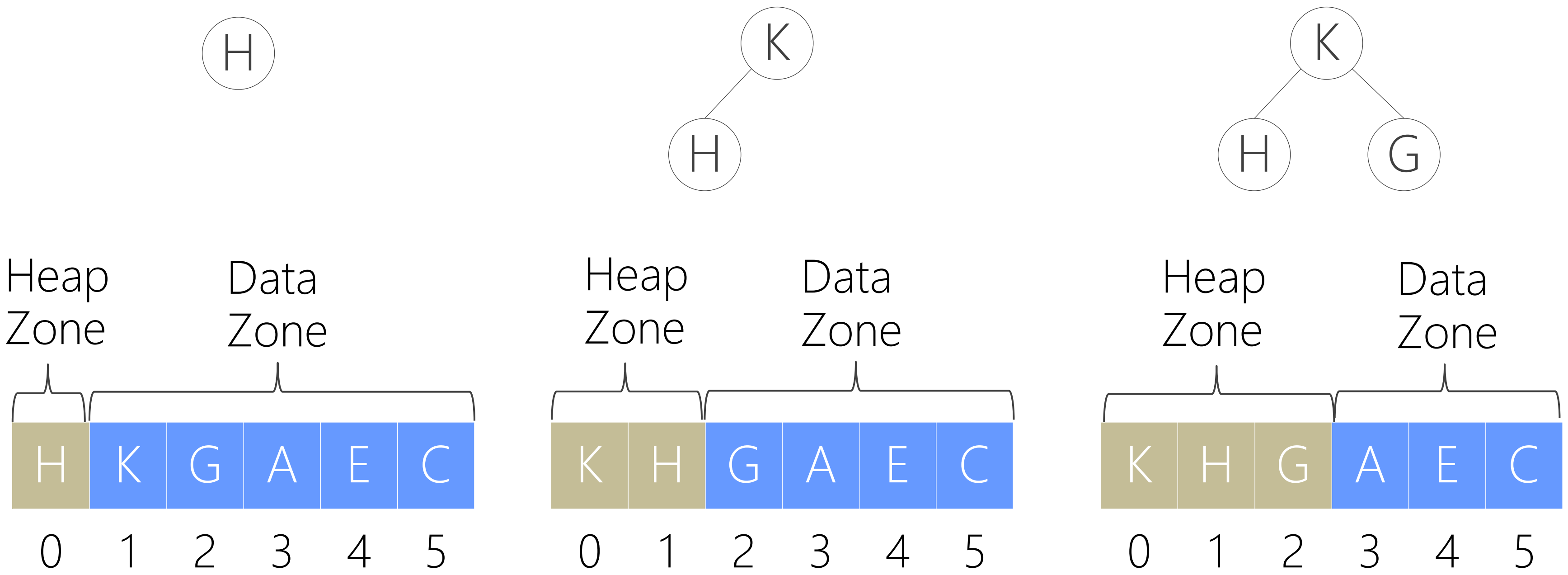
O(n) time
O(n) storage

Can we do better?

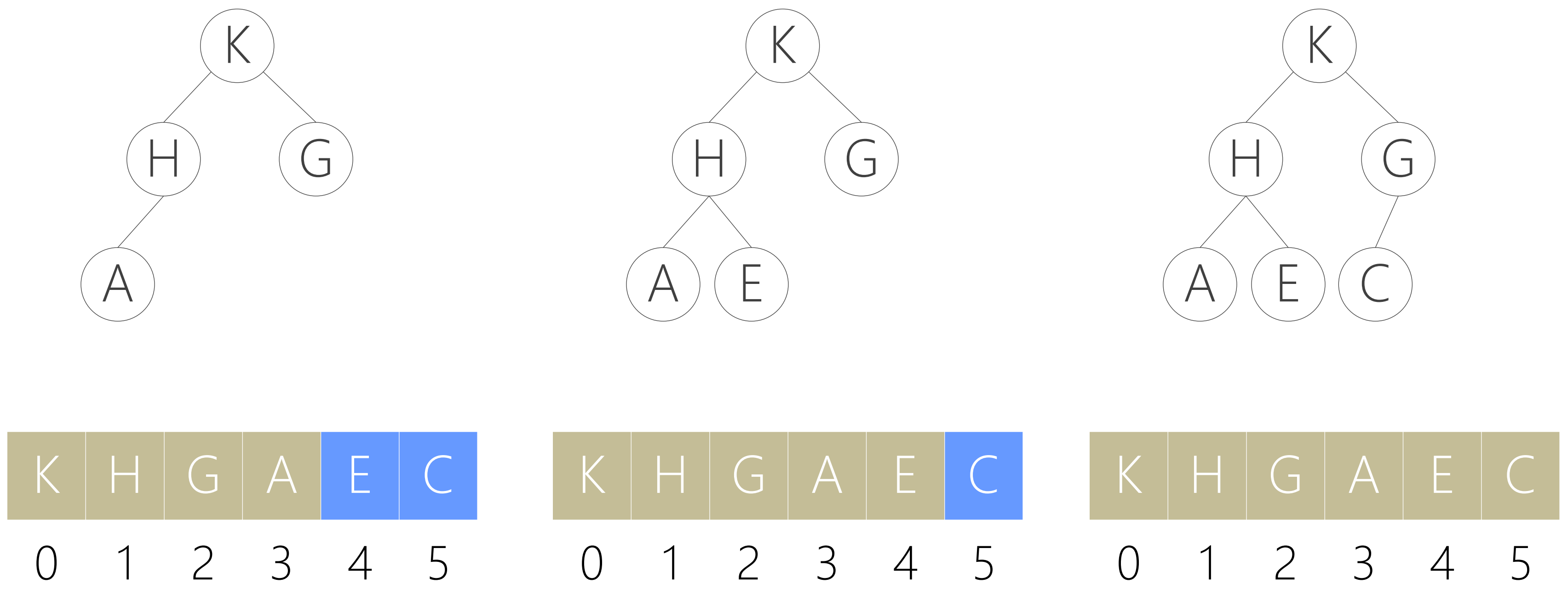
Heapsort: No Extra Storage

- Observation: after each deleteMin, the size of heap shrinks by 1
  - We can use the last cell just freed up to store the element that was just deleted
  - after the last deleteMin, the array will contain the elements in decreasing order
- Further observation:
  - To sort the elements in decreasing order, use a min heap
  - To sort the elements in increasing order, use a max heap
    - Max Heap: the parent has a larger element than the child

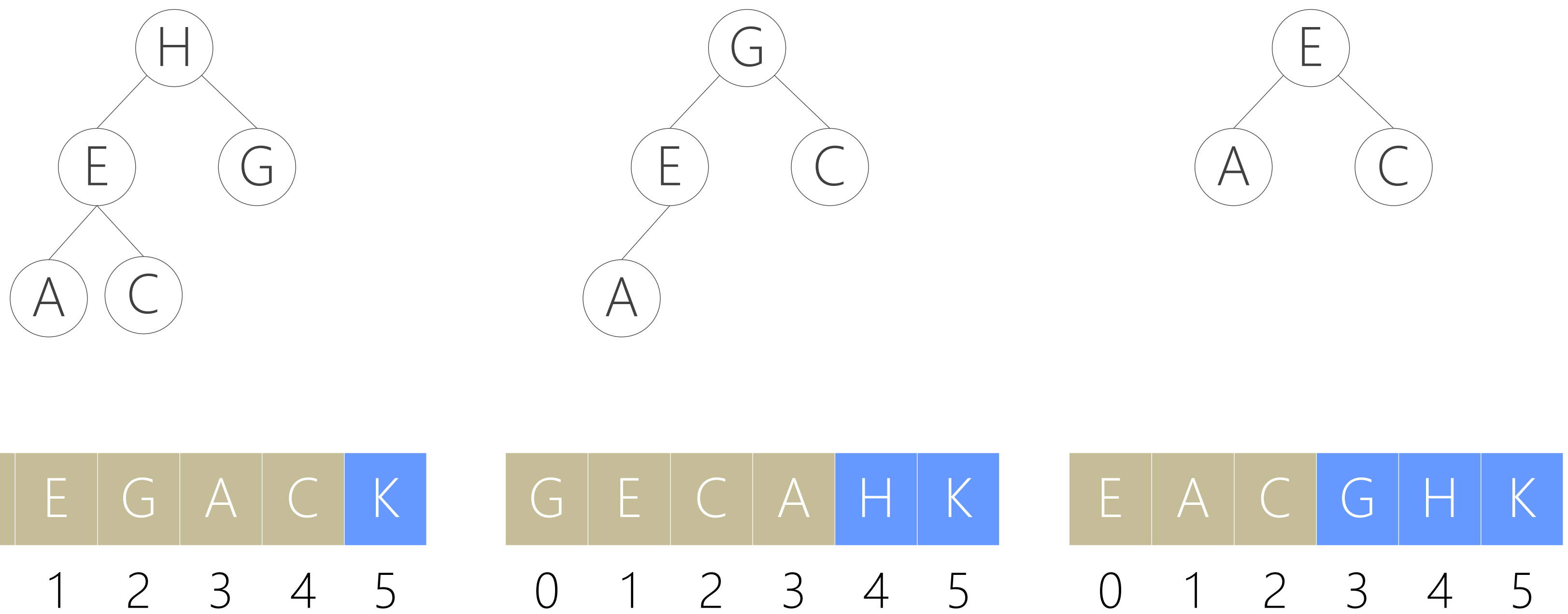
Example: Heap Build-up



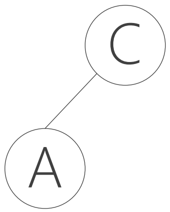
Example: Heap Build-up

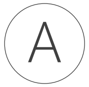


Example: deteleMax



Example: deteleMax





C	A	E	G	H	K
0	1	2	3	4	5

A	C	E	G	H	K
0	1	2	3	4	5

A	C	E	G	H	K
0	1	2	3	4	5

Heapsort Pseudo Code

```
HEAPSORT(A)
1.    heap = new MaxHeap(A)
2.    FOR Each x in A
3.        heap.insert(x)
4.    FOR i=size-1 TO 0
5.        A[i] = heap.deleteMax()
```

Task

- Create a class, MaxHeap, as described below and submit MaxHeap.java to iSpace

MaxHeap
- A: int[] - size: int
+ MaxHeap(int A[]) + insert(int x): boolean + deleteMax(): int + <u>heapSort(int A[]): void</u> + <u>main(): void</u>

Methods

- public MaxHeap(int A[])
  - Constructs a MaxHeap which uses an existing array, *A*.
- public boolean insert(int x)
  - Adds *x* into the heap
  - Returns true if the operation is successful, and false otherwise
- public int deleteMax()
  - Removes and returns the (old) maximum element of the heap
  - Returns -1 if the operation fails

Methods

- public static void heapSort(int[] A)
  - *A* is an array of integers
  - Sort *A* using heapsort
- public static void main(String[] args)
  - Generate an array *A* consisting of 10^5 random integers which are in range [0, 999]
  - Sort *A* using heapSort
  - Print the elapsed time in milliseconds during which the sort function runs