UNIVERSITY INSTITUTE OF COMPUTING

MASTER OF COMPUTER APPLICATIONS

DESIGN AND ANALYSIS OF ALGORITHMS

24CAT-611

UNIT-2

DISCOVER . LEARN . EMPOWER

DESIGN AND ANALYSIS OF ALGORITHMS

Course Outcome

CO Number	Title	Level
CO3	Apply and analyze important algorithmic design paradigms and their applications	Understand
CO4	Implement the major graph algorithms to model engineering problems	Understand

- **Divide and Conquer**: General method, Binary search, Advantages and disadvantages of divide and conquer, Decrease and conquer approach: Topological sort

Topics to be covered

- Dijkstra’s Algorithm
- Huffman Trees and Codes
- Heaps and Heap Sort

Transform and Conquer

Transform and conquer is a design paradigm where a given problem is transformed to another domain. This can be done for familiarity of simplicity. The problem is solved in the new domain. Then, the solutions are converted back to the original domain. In short, transform and conquer proposes two stage solution

1. First stage involves the transformation to another problem that is more amenable for solution
2. Second stage involves solving the new problem where the transformed new problem is solved.

Then the solutions are converted back to the original problem.

Variations of Transform and Conquer

Three variants of transform and conquer are as follows:

– Instance Simplification

– Representation Change

– Problem Reduction

Instance simplification

- Instance simplification is one variant where the problem transformed to the same problem of simpler or convenient instance. The illustrated example of roman number to Arabic number system is an example of instance simplification.

Representation Change

- Representation Change is another variety where the strategy involves the transformation of an instance to a different representation. But this is done without affecting the instance. The illustrated example of roman number to Arabic number system is an example of instance simplification.

Problem reduction

- Problem reduction is a strategy that involves a transformation of a problem A to another type of problem B. It is assumed that the solution of problem B already exists. The illustrated example of reduction of computing LCM (Last Common Multiple) in terms of GCD is an example of problem reduction. s GCD. Hypothetically, let us assume that an algorithm exists only for GCD.

Transform and conquer approach

- Transform and conquer approach Instead, the array can be sorted and run length of an element can be calculated on the sorted array. Informally, mode finding using presorting is given as follows:

1) Sort the element of an array.
2) Calculate the run length of an element
3) Print the element having longest length
4) Exit.

Formal algorithm

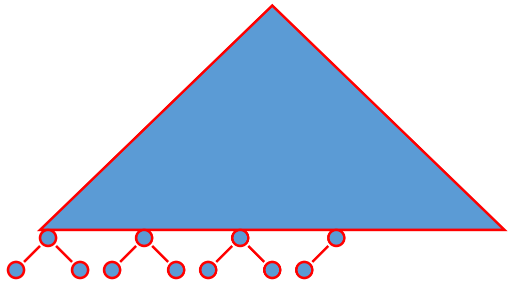
• Formal algorithm is given as follows:

```
Sort A i = 0
frequency = 0
while i ≤ n-1
runlength = 1; runvalue =A[i]
while i+runlength ≤ n-1 and A[i+runlength] = runvalue
runlength = runlength + 1
 if runlength > frequency
frequency =runlength
modevalue =runvalue
i = i + runlength
return modevalue
```

Heaps and Heapsort

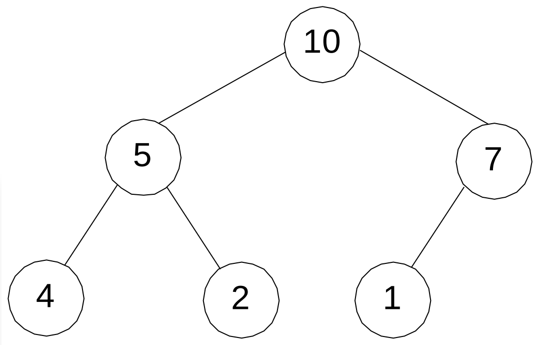
Definition A *heap* is a binary tree with keys at its nodes (one key per node) such that:

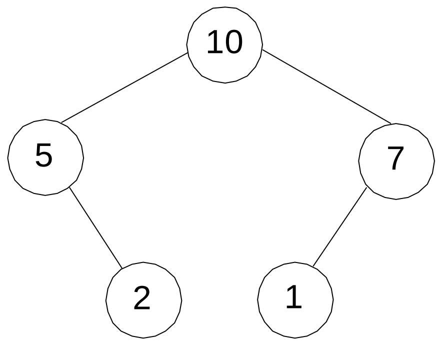
- It is essentially complete, i.e., all its levels are full except possibly the last level, where only some rightmost keys may be missing

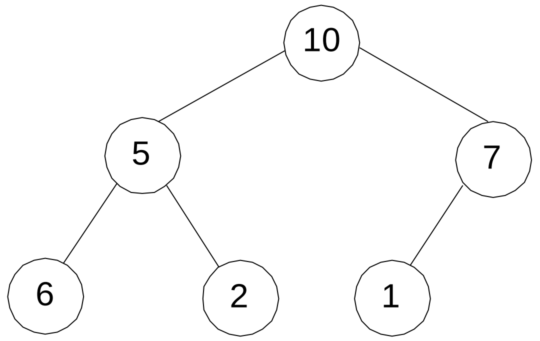


- The key at each node is ≥ keys at its children (this is called a *max-heap*)

Illustration of the heap's definition







a heap **not a heap** **not a heap**

Note: Heap's elements are ordered top down (along any path down from its root), but they are not ordered left to right

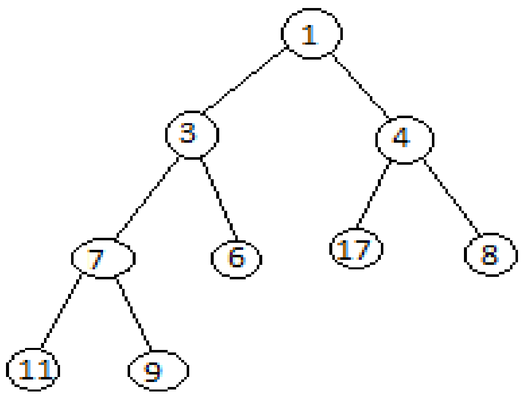
Some Important Properties of a Heap

- The root contains the largest key
- The subtree rooted at any node of a heap is also a heap
- A heap can be represented as an array

Heap Property

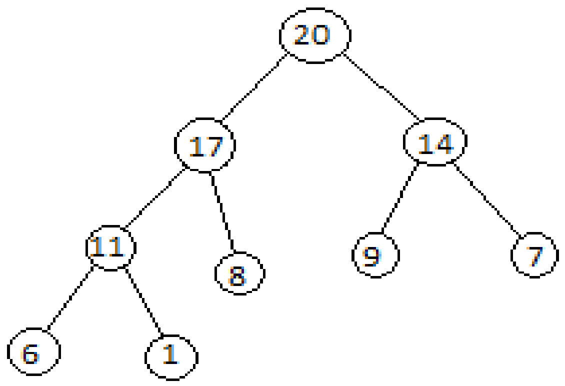
- **Heap Property:** All nodes are either **greater than or equal to** or **less than or equal to** each of its children. If the parent nodes are greater than their child nodes, heap is called a **Max-Heap**, and if the parent nodes are smaller than their child nodes, heap is called **Min-Heap**.

Heap Property



Min-Heap

In min-heap, first element is the smallest. So when we want to sort a list in ascending order, we create a Min-heap from that list, and picks the first element, as it is the smallest, then we repeat the process with remaining elements.



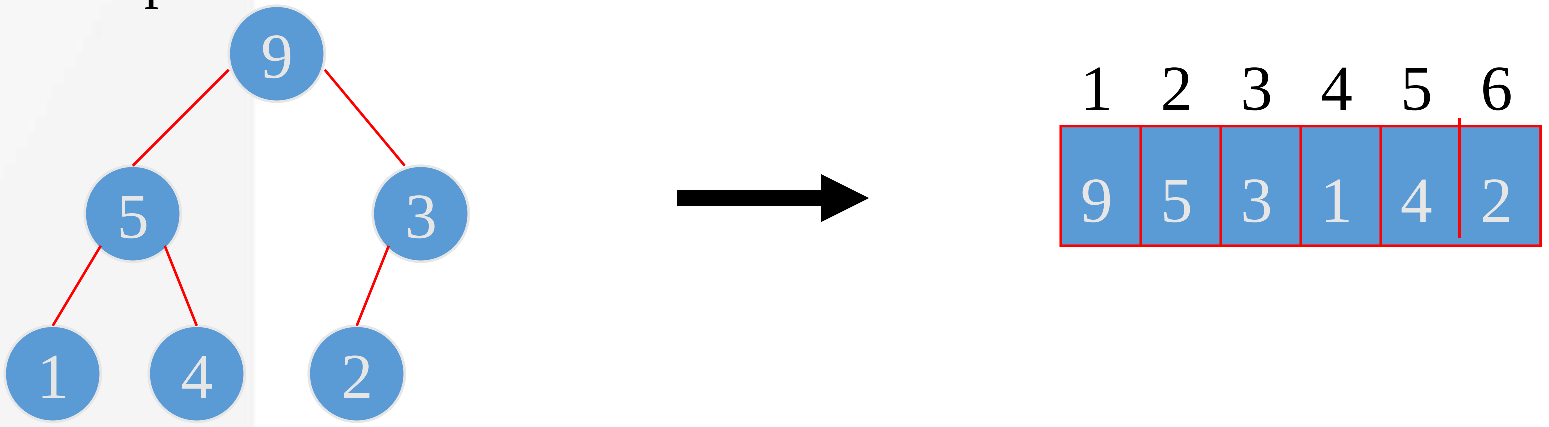
Max-Heap

In max-heap, the first element is the largest, hence it is used when we need to sort a list in descending order.

Heap's Array Representation

Store heap's elements in an array (whose elements indexed, for convenience, 1 to *n*) in top-down left-to-right order

Example:



- Left child of node *j* is at 2*j*
- Right child of node *j* is at 2*j*+1
- Parent of node *j* is at $\lfloor j/2 \rfloor$
- Parental nodes are represented in the first $\lfloor n/2 \rfloor$ locations

Heap Construction (bottom-up)

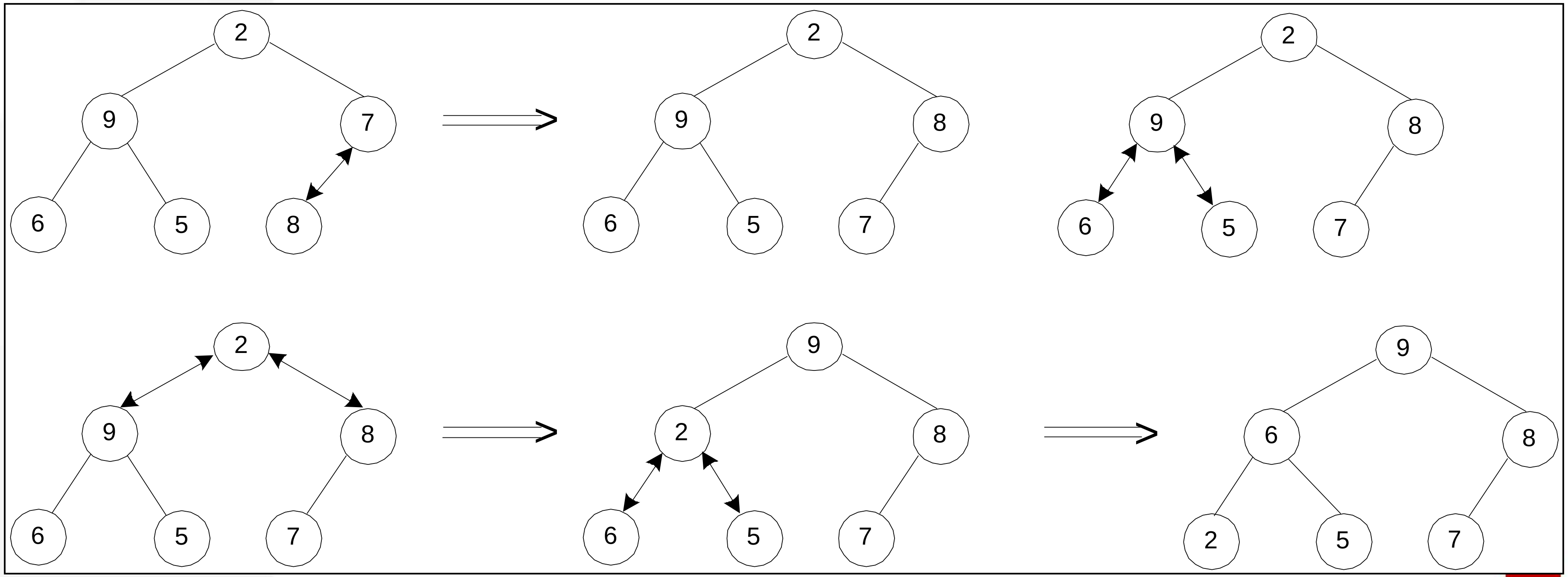
Step 0: Initialize the structure with keys in the order given

Step 1: Starting with the last (rightmost) parental node, fix the heap rooted at it, if it doesn’t satisfy the heap condition: keep exchanging it with its larger child until the heap condition holds

Step 2: Repeat Step 1 for the preceding parental node

Example of Heap Construction

Construct a heap for the list 2, 9, 7, 6, 5, 8



Heapsort

Stage 1: Construct a heap for a given list of *n* keys

Stage 2: Repeat operation of root removal *n*-1 times:

- Exchange keys in the root and in the last (rightmost) leaf
- Decrease heap size by 1
- If necessary, swap new root with larger child until the heap condition holds

Example of Sorting by Heapsort

Sort the list 2, 9, 7, 6, 5, 8 by heapsort

Stage 1 (heap construction)

1 9 $\underline{7}$ 6 5 8

2 $\underline{9}$ 8 6 5 7

$\underline{2}$ 9 8 6 5 7

9 $\underline{2}$ 8 6 5 7

9 6 8 2 5 7

Stage 2 (root/max removal)

$\underline{9}$ 6 8 2 5 7

7 6 8 2 5 | 9

$\underline{8}$ 6 7 2 5 | 9

5 6 7 2 | 8 9

$\underline{7}$ 6 5 2 | 8 9

2 6 5 | 7 8 9

$\underline{6}$ 2 5 | 7 8 9

5 2 | 6 7 8 9

$\underline{5}$ 2 | 6 7 8 9

2 | 5 6 7 8 9

Insertion of a New Element into a Heap

- Insert the new element at last position in heap.
- Compare it with its parent and, if it violates heap condition, exchange them
- Continue comparing the new element with nodes up the tree until the heap condition is satisfied

Example: Insert key 10

Efficiency: O(log *n*)

References

1) http://epgp.inflibnet.ac.in/epgpdata/uploads/epgp_content/S000007CS/P001064/M014960/ET/1459249858E17.pdf
2) https://www.studytonight.com/data-structures/heap-sort
3) https://www.interviewbit.com/tutorial/heap-sort-algorithm/

Books:

1. Introduction to Algorithms by Coreman, Leiserson, Rivest, Stein.
2. Fundamentals of Algorithms by Ellis Horwitz, Sartaj Sahni, Sanguthevar Rajasekaran

CU
CHANDIGARH

NAAC GRADE A+

THANK YOU