LECTURE : 1

⇒ **Review of Data Structures :**

↳ Priority queues : It is a data type for holding a dynamic collection of items, where each item is a key. It supports the following operations :

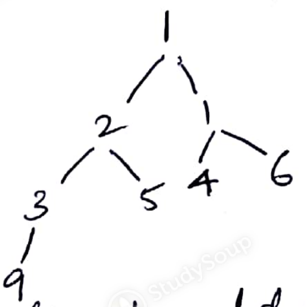
1. Insert(x) — Insert x into collection
2. $x \leftarrow$ extract_min( ) — return and delete object with minimum key.
3. $x \leftarrow$ Find_min( ) — return object with minimum key.
4. Decrease-Key(x, k) — lower object x's key to k.

Heap is a data structure of priority queue datatype.

Time taken for all operations $= O(\log n)$
where $n \rightarrow$ number of keys in the collection

Example :
Keys : 3, 1, 4, 5, 9, 2, 1, 6.

Heap Structure :



(This structure avoids empty spaces in array)

Rule : (Every parent should not be larger than the child.)

Application : To get sorted list, we can extract min for the entire heap.
Time : $n \log n$

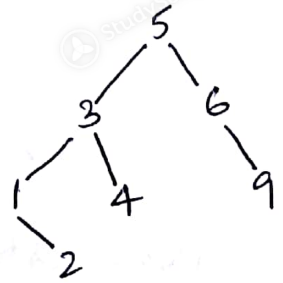
Scanned by CamScanner

↳ **Binary search trees**: It is a rooted binary tree whose nodes represent a dynamic collection of objects that have keys.
It is stored in standard binary representation — where each node has pointers to its parent and its two children.
It maintains search-tree order — At each node, the key is as large as each key in its left subtree, and no larger than any keys in its right subtree.
It supports the following operations:

1. Insert (x) — add object x to collection
2. Delete (x) — remove object x from collection.
3. Search (k) — return object with key k, if present.
4. Minimum () — return object with minimum key.
5. successor (x) — return object y whose key is the next largest key after x's key.

Time for all operations = O(height of tree).
Assumption: keys are distinct.

Example:
keys: 3, 1, 4, 5, 9, 2, 6.



[left side ↓se
right side ↑se]

To insert an object: If object is already present, the new object replaces the current one.

To search a key: Compare key to root.
If root = key, then root is key.
else if key > root, right side search
else left side search.

Scanned by CamScanner

↳ **Dictionaries / hash tables:** It stores a set of keys and values.

It supports the following operations:

Insert (x, k) – Inserts key-object pair into collection

Delete (x) – remove object x from collection

Search (k) – return object with key k, if present.

Assumption: keys are distinct.

Time taken for all operations = O(1) per operation

2 ways of filling an array:
- Chaining (store multiple elements in a cell)
- Open addressing (store single element in a cell)

Example of Chaining:

Keys: 3, 1, 4, 5, 9, 2, 6, 7, 8

A [0 . . . 4] $f(k) = (k \bmod 5)$

A[0] → $\boxed{5}$

A[1] → $\boxed{1}$ → $\boxed{6}$

A[2] → $\boxed{2}$ → $\boxed{7}$

A[3] → $\boxed{3}$ → $\boxed{8}$

A[4] → $\boxed{4}$ → $\boxed{9}$

LECTURE: 2

While writing a proof for a theorem, the proof should be a sequence of small steps. Each step must be:

- a definition or statement of fact,
- first line of block (such as assumption),
- combination of the above.

Example of long-form proof:

Theorem: The number $x = \log_9 12$ is irrational.

Proof:

1. Assume for contradiction that x is rational.
2. By definition of "rational", $x = p/q$ such that $q > 0$ and:

$$p/q = \log_9 12$$

$$9^{p/q} = 12 \quad \text{(raise 9 to power of each side)}$$

$$9^p = 12^q \quad \text{(raise each side to power of } q\text{)}$$

3. Since $q > 0$, 12^q is an even integer.
4. 9^p cannot be an even integer.
5. This is a contradiction.

$\therefore$ x is irrational.

→ Different types of blocks that can be used:

1. "For all" blocks.
   Eg: "let x be an arbitrary positive integer".
2. "If then" blocks.
3. "Proof by contradiction" blocks.
   Eg: "Assume for contradiction that ..."
4. Blocks for proof by cases.
   Eg: "Consider the case that ...".

→ Basic logic :

① "if A then B" ≡ "if not B then not A"
② "if A then B" ≡ "B or not A"
③ "not (A and B)" ≡ "(not A) or (not B)"
④ "not (A or B)" ≡ "(not A) and (not B)"
⑤ "not (not A)" ≡ "A"
⑥ "not $\forall x \in S.\ P(x)$" ≡ "$\exists x \in S$. not $P(x)$"
⑦ "not $\exists x \in S.\ P(x)$" ≡ "$\forall x \in S$. not $P(x)$"

Example of long-form proof :

Theorem : for all positive integers n, if $\log_7 n$ is rational then it is an integer.

Proof :

1. Consider an arbitrary positive integer n.

2.1. Assume that $\log_7 n$ is rational.

2.2. Let p and q be integers with $q > 0$ such that $p/q = \log_7 n$, so

$p/q = \log_7 n$ (raise 7 to power of each side)

$7^{p/q} = n$ (raise each side to power of q) (1)

$\underbrace{7^p = n^q}$ ↳ (for this equality to hold, n has to be a power of 7.)

2.3. Each integer has a unique prime factorization, so Eq.① holds and implies that n is a power of 7.

2.4. $n = 7^i$ for some integer > 0.

2.5. $7^p = (7^i)^q = 7^{iq}$ (taking $\log_7$ of each side)

$p = iq$

$p/q = i$

$\log_7 n = i$

2.6. Therefore $\log_7 n$ is an integer.

3. If ~~log~~ $\log_7 n$ is rational, then $\log_7 n$ is an integer.