Data Structure & Algorithms

Sunbeam Infotech

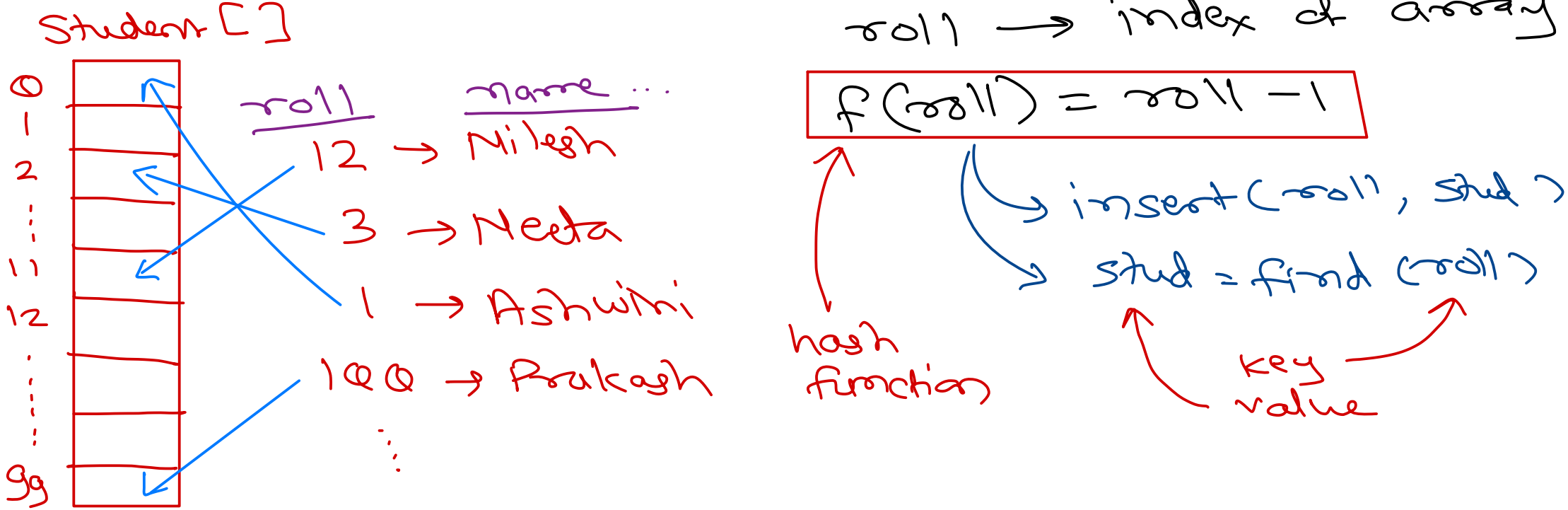
Nilesh Ghule

Hash Table

C++ STL : map<>
Java : HashMap<>
Python / C# : Dictionary

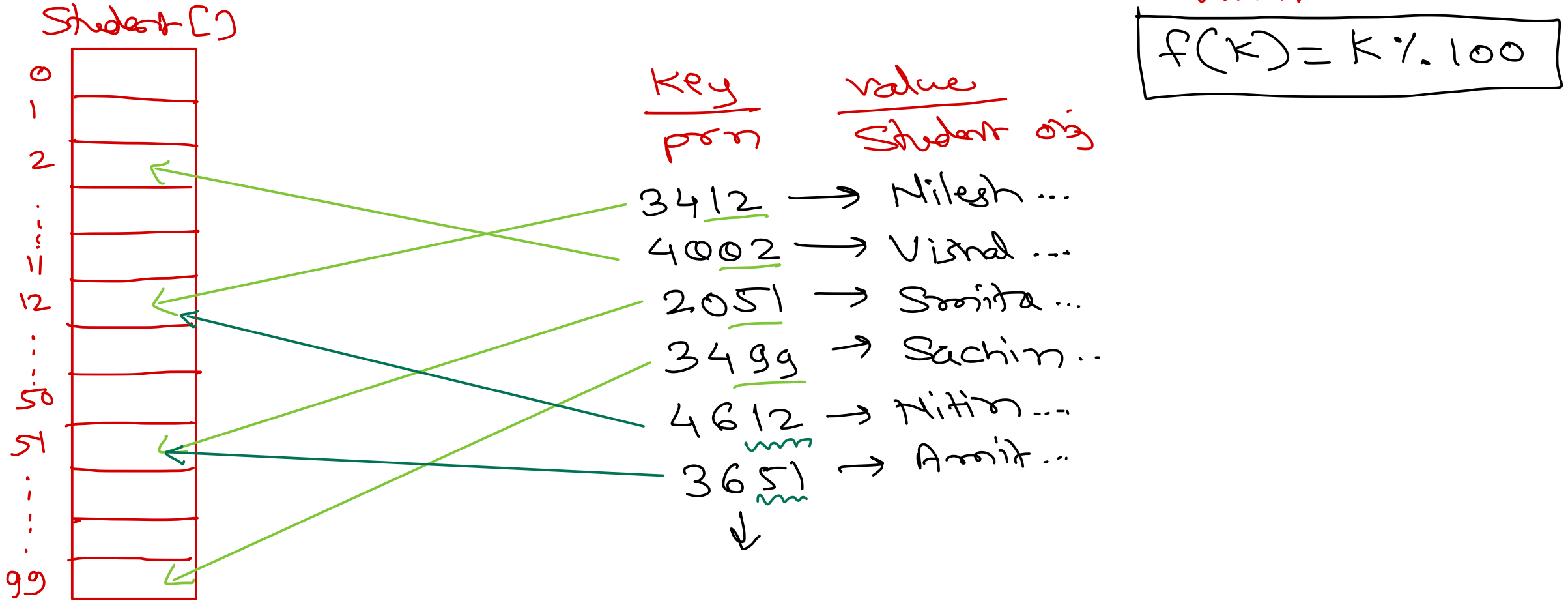
Searching : ① linear search — O(n)
② binary search — O(log n)
③ fibonacci search
④ hashing — O(1)

- Associative data structure
- Stores key-value so that for a given key, value can be searched in fastest possible time. Ideal time complexity is O(1).
- Example:



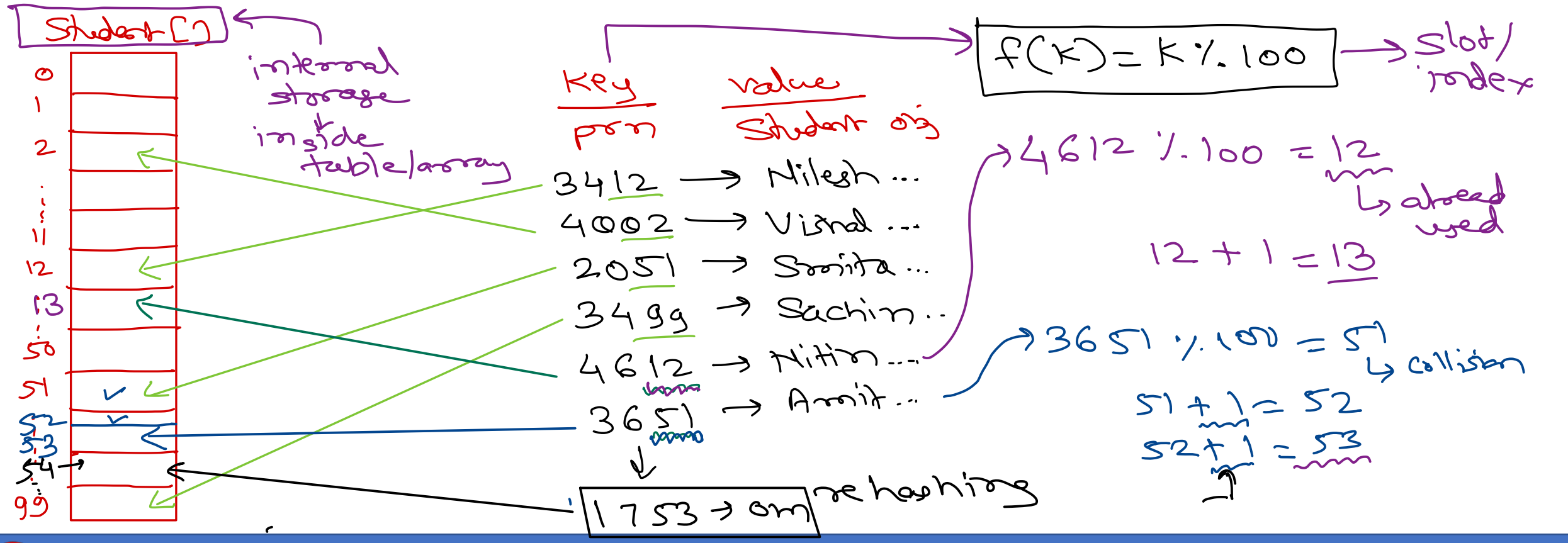
Hash Table

- Hash Function is math function of key, that yields slot in the table.
- If different keys resulting in same slot in the table, it is called as collision.



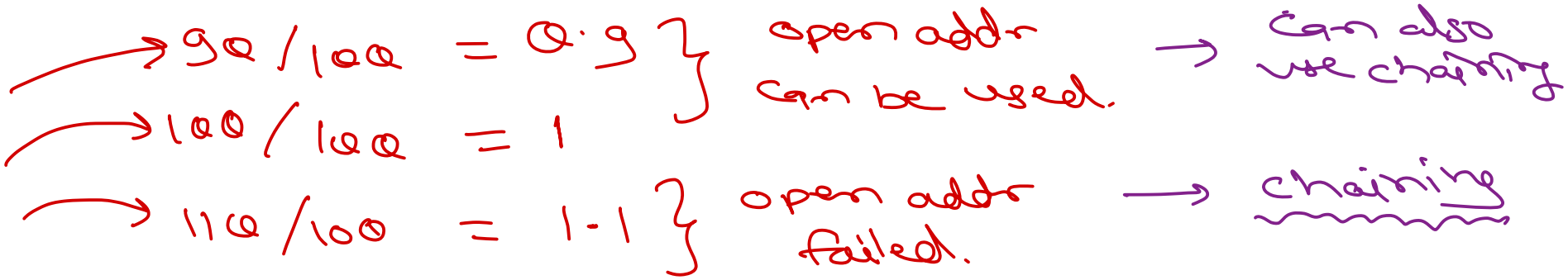
Hash Table

- Collision handling methods: Open addressing or Chaining
- Open addressing
  - Rehashing: Linear probing, Quadratic probing, …



Hash Table

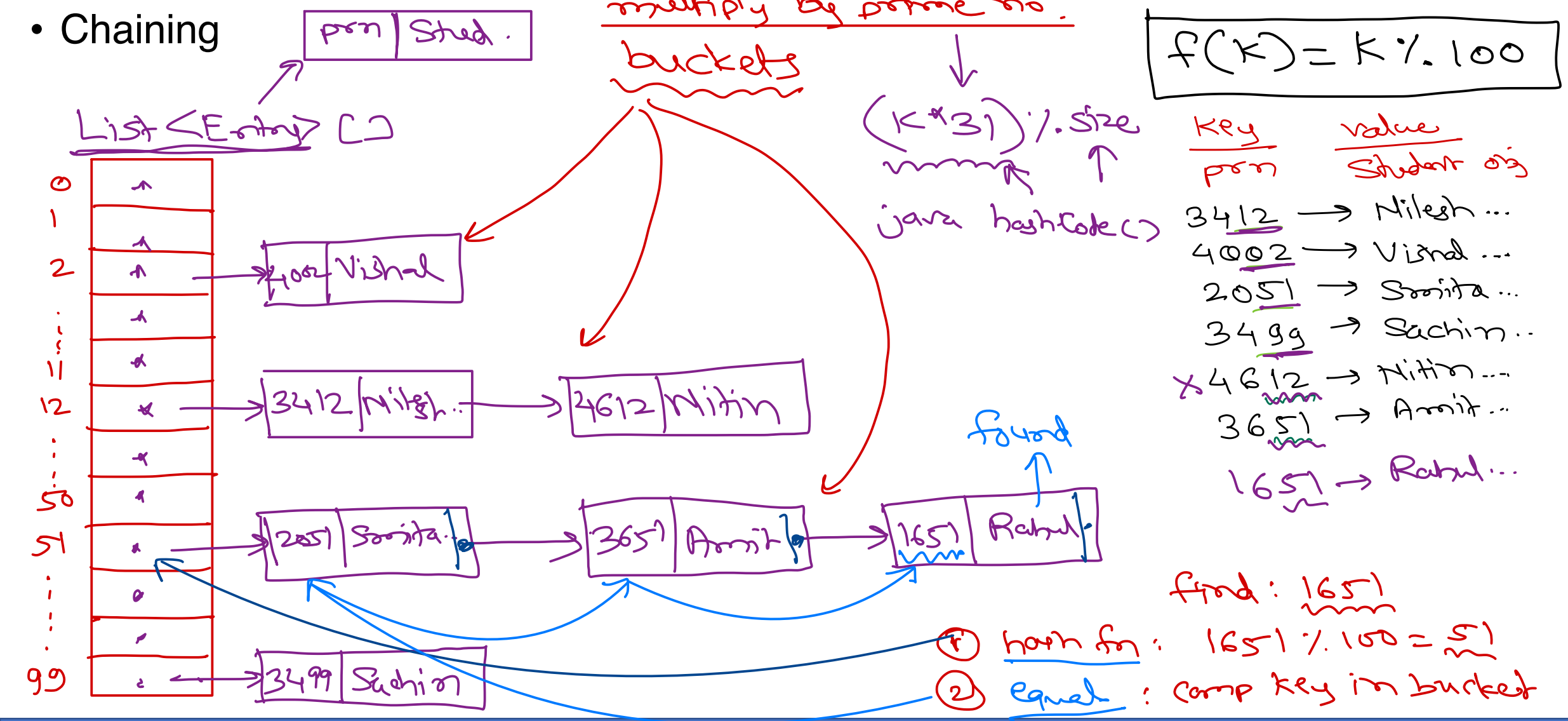
- Load factor = Number of entries / Number of slots
- Cases
  - Load factor < 1
  - Load factor = 1
  - Load factor > 1



Hash Table

hash fn → uniform probability distribution

- Chaining



Hash Table

key = roll value = Student

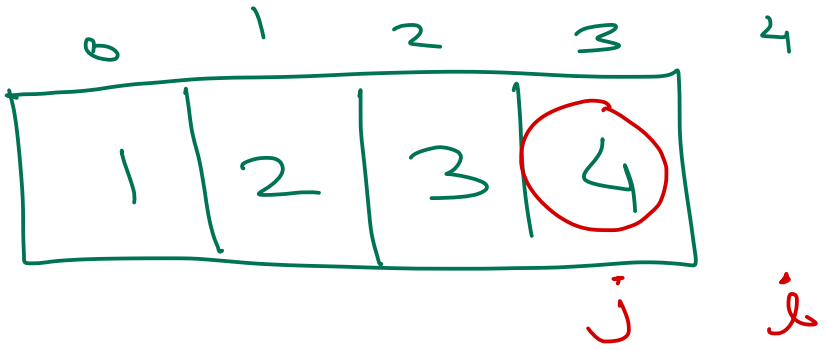
int obj

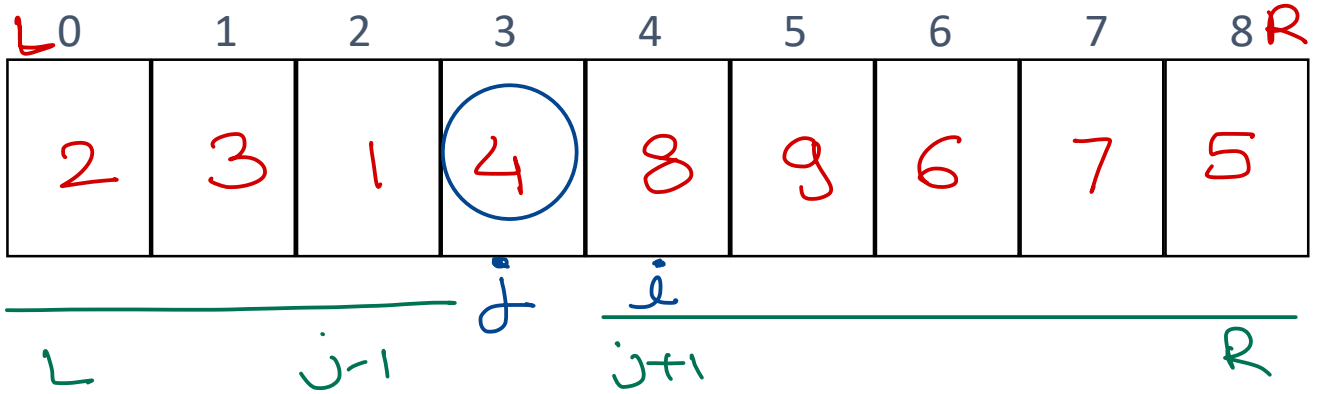
```
class Entry {
    int key;
    Student value;
    =
};
```

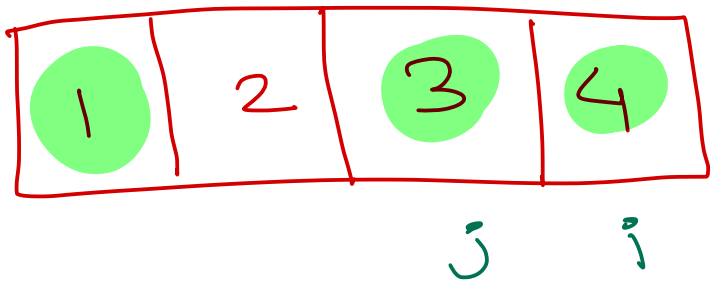
```
class HashTable {
    List<Entry> [] table;
    ctor:
        → all buckets are empty.
    put : (roll, stud)
    get : stud ← (roll)
};
```

Quick Sort

4 7 9 2 8 1 6 3 5







O(n log n) → pivot find pn is middle.

n → n-1 ⟹ O(n²)

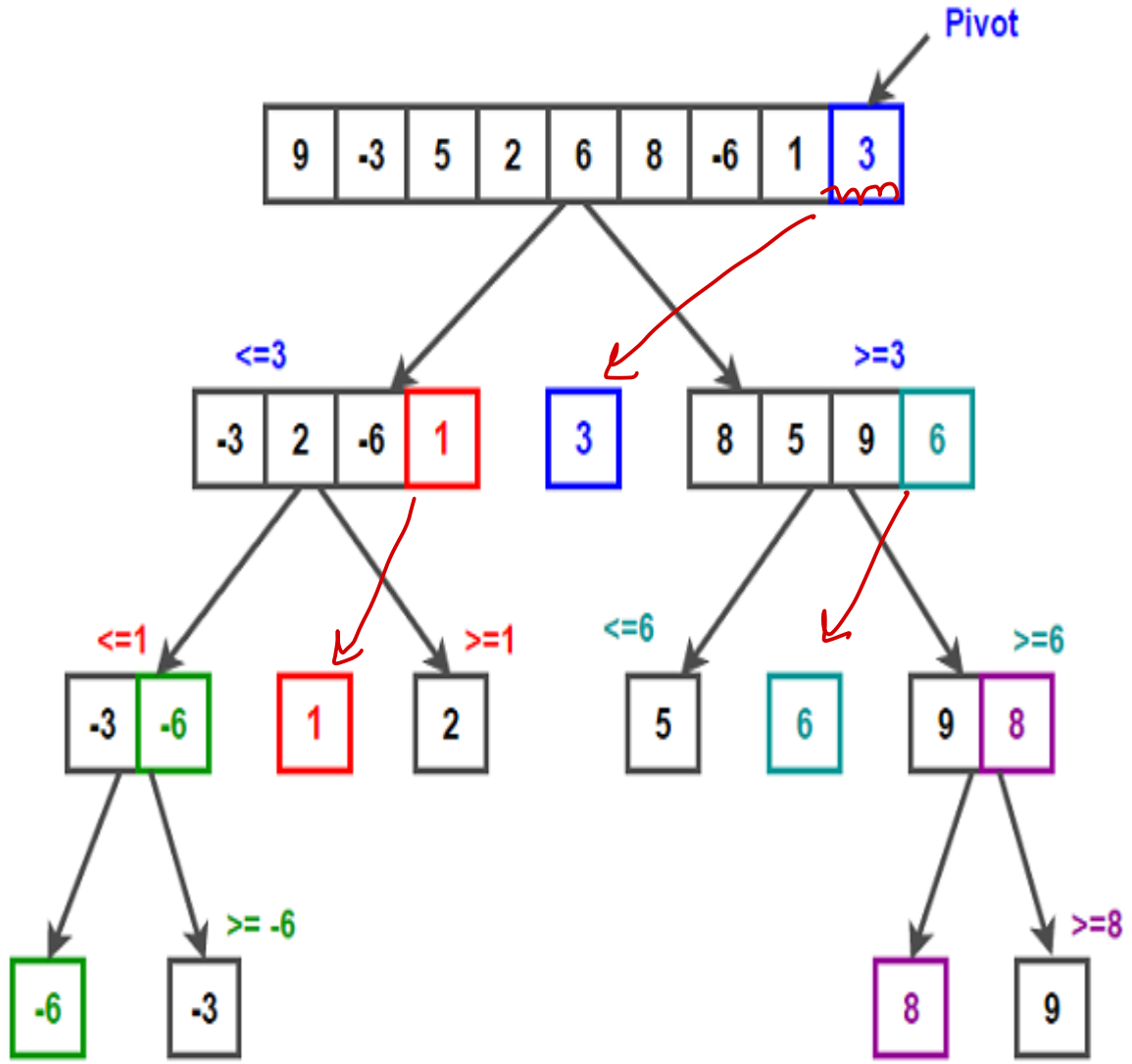
Quick Sort – Time complexity

- Quick sort pivot element can be
  - First element or Last element
  - Random element
  - Median of the array → best case
- Quick sort time
  - Time to partition as per pivot – T(n)
  - Time to sort left partition – T(k)
  - Time to sort left partition – T(n-k-1)
- Worst case
  - $T(n) = T(0) + T(n-1) + O(n) \Rightarrow O(n^2)$
- Best case
  - $T(n) = T(n/2) + T(n/2) + O(n) \Rightarrow O(n \log n)$
- Average case
  - $T(n) = T(n/9) + T(9n/10) + O(n) \Rightarrow O(n \log n)$

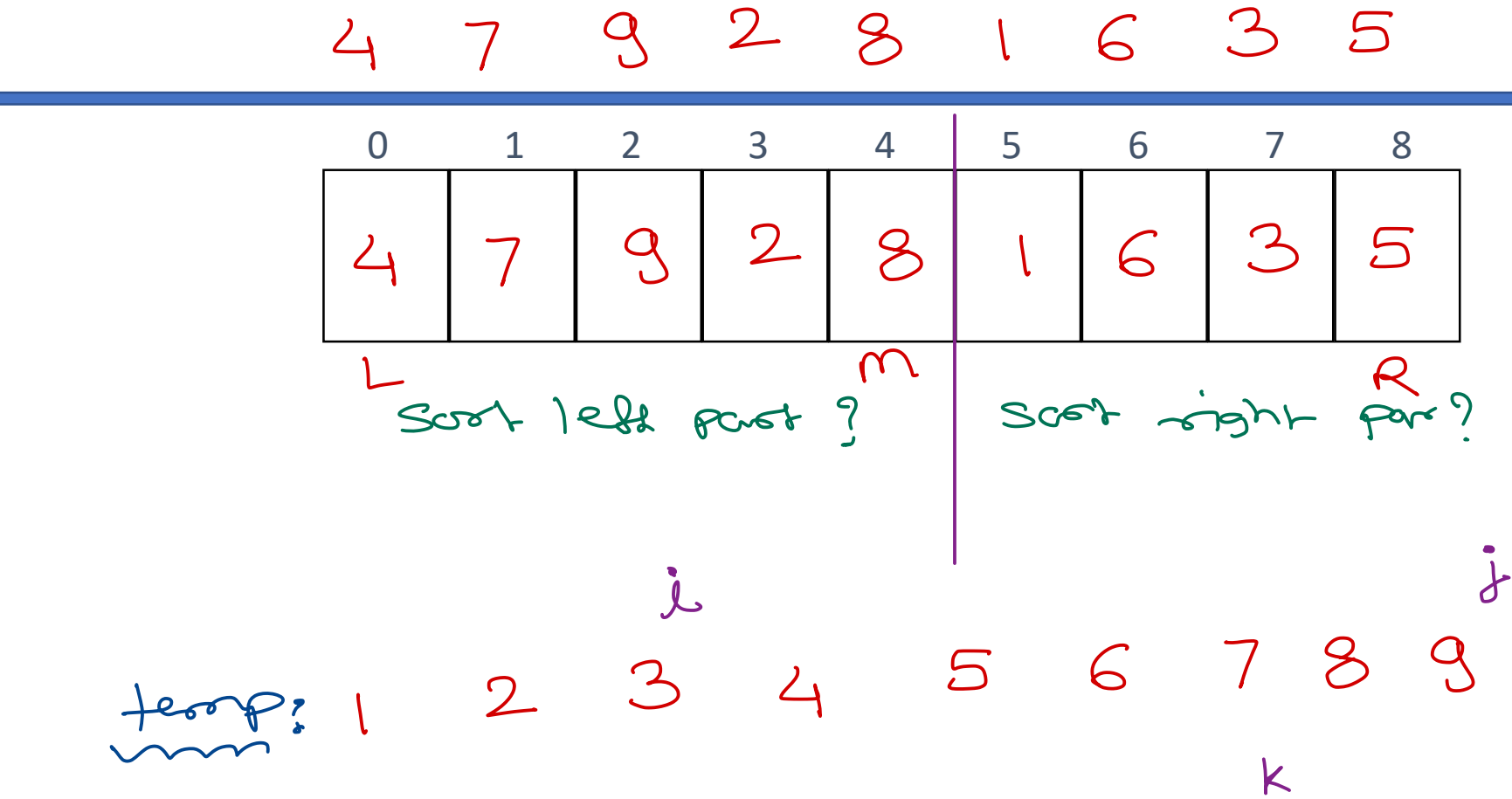
Recursion – QuickSort

- Algorithm
  1. If single element in partition, return.
  2. Last element as pivot.
  3. From left find element greater than pivot (x^{th} ele).
  4. From right find element less than pivot (y^{th} ele).
  5. Swap x^{th} ele with y^{th} ele.
  6. Repeat 2 to 4 until x < y.
  7. Swap y^{th} ele with pivot.
  8. Apply QuickSort to left partition (left to y-1).
  9. Apply QuickSort to right partition (y+1 to right).

- QS(arr, 0, 8)
  - QS(arr, 0, 3)
    - QS(arr, 0, 1)
      - QS(arr, 0, 0)
    - QS(arr, 3, 3)
  - QS(arr, 5, 8)
    - QS(arr, 5, 5)
    - QS(arr, 7, 8)
      - QS(arr, 9, 9)



Merge Sort


4 7 9 2 8 1 6 3 5
0 1 2 3 4 5 6 7 8
4 7 9 2 8 1 6 3 5
L
m
R
Sort left part ?
Sort right par ?
i
j
temp:
1 2 3 4 5 6 7 8 9
k

Merge Sort

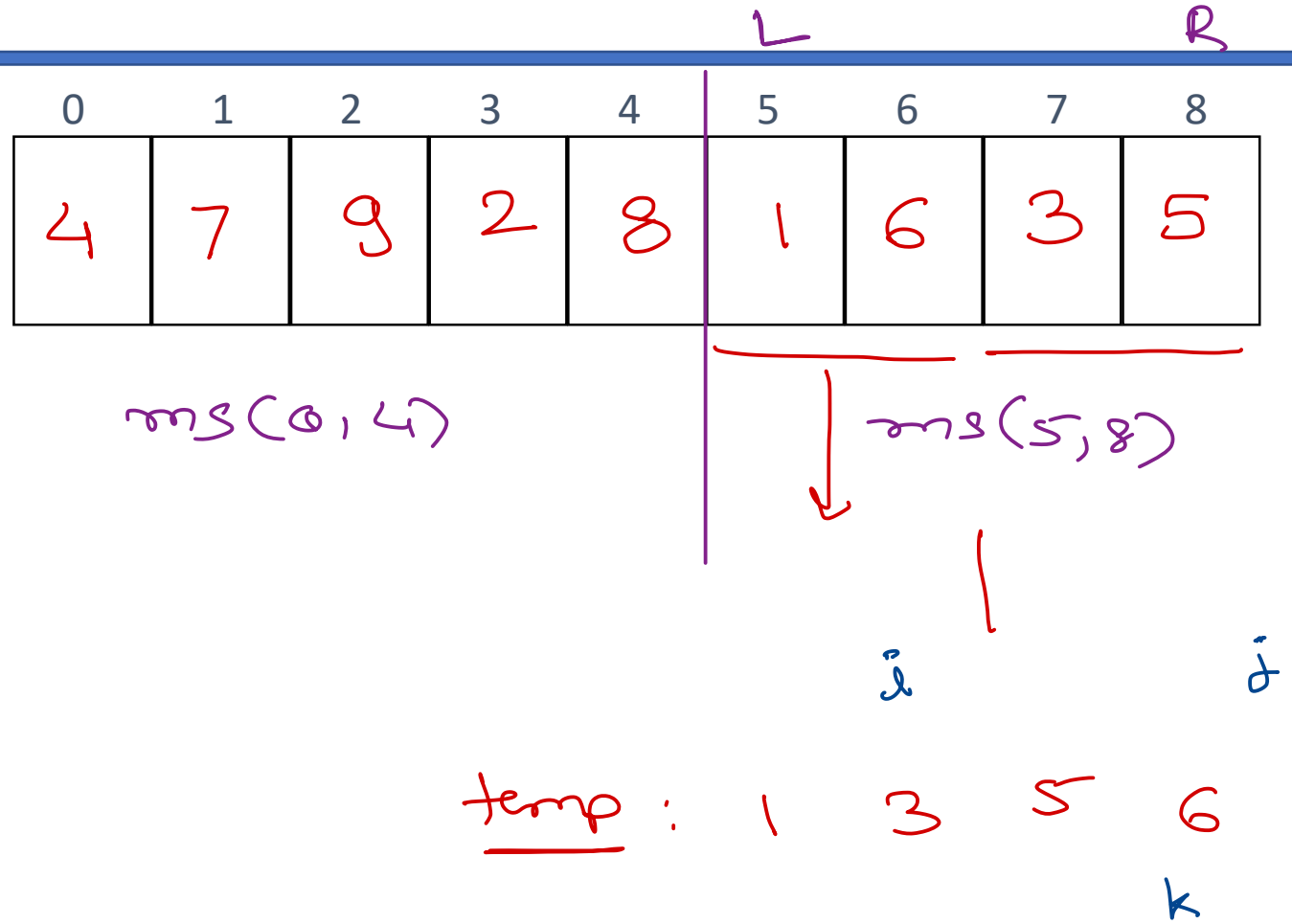
num of recursion

↳ O(log n)

each recur call

↳ O(n)

T.C. = O(n log n)



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>