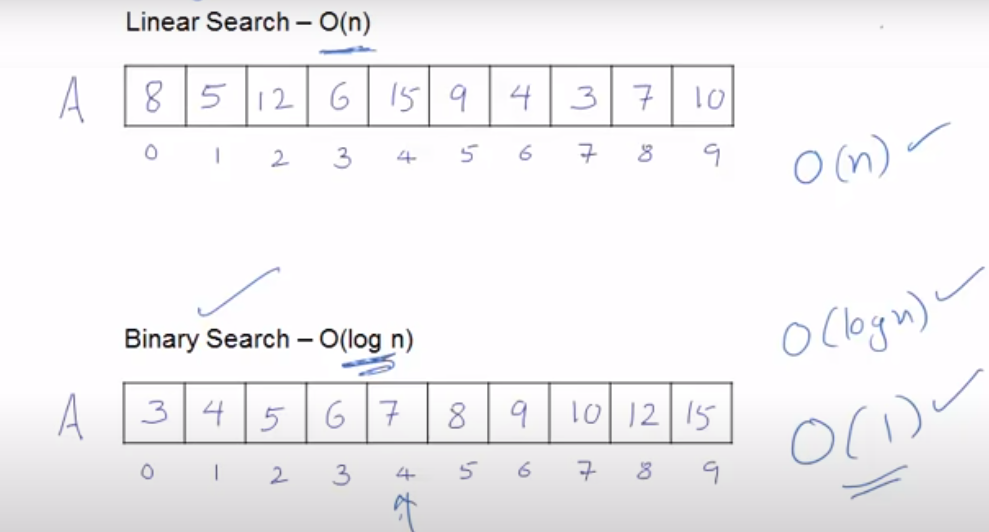
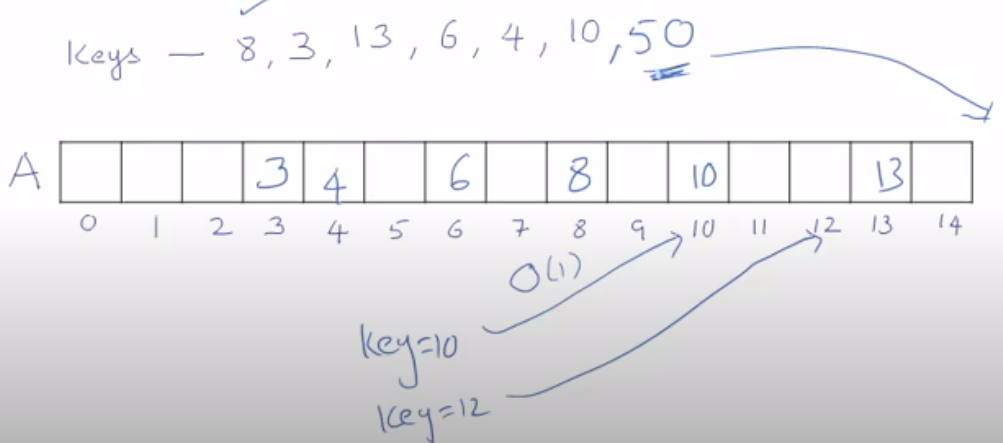
Hashing: is

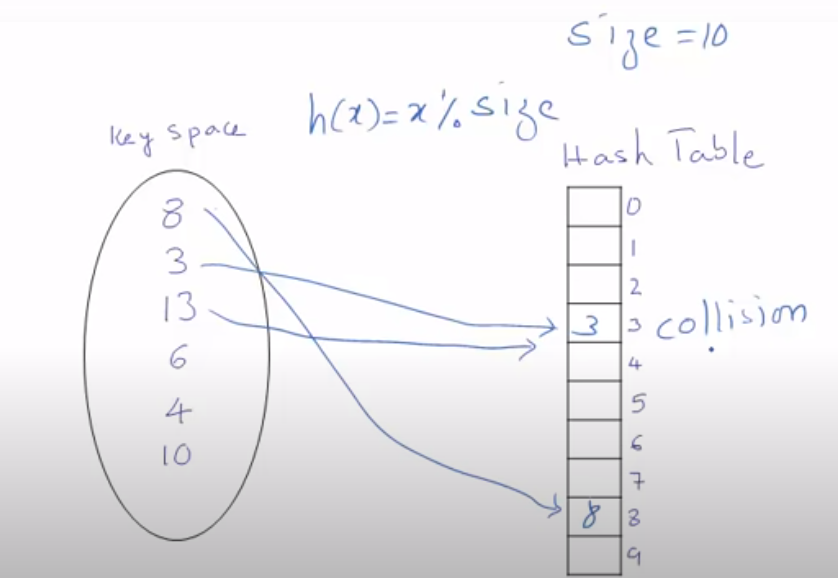
Linear search used to take o(n) and Binary search used to take o(log n) and also array must be sorted. Hashing technique takes only o(1).



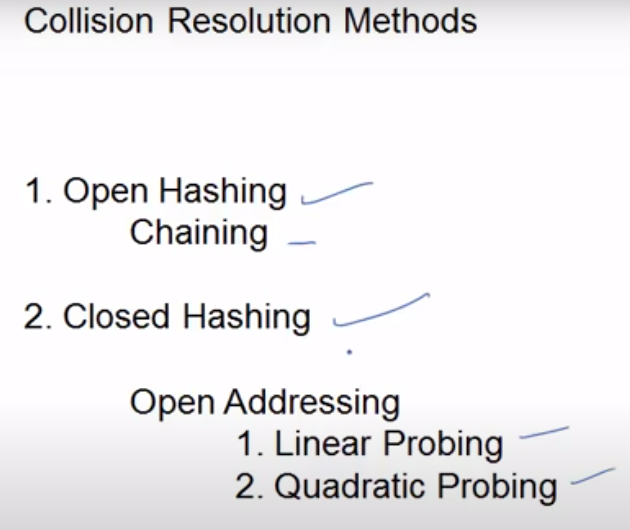
**Issue with hashing approach:** Since uses h(x) = x function. It means index no. will be same as value. Value 250 will be stored in index 250. This approach will consume lot of space and that’s not good approach. We need to modify it.



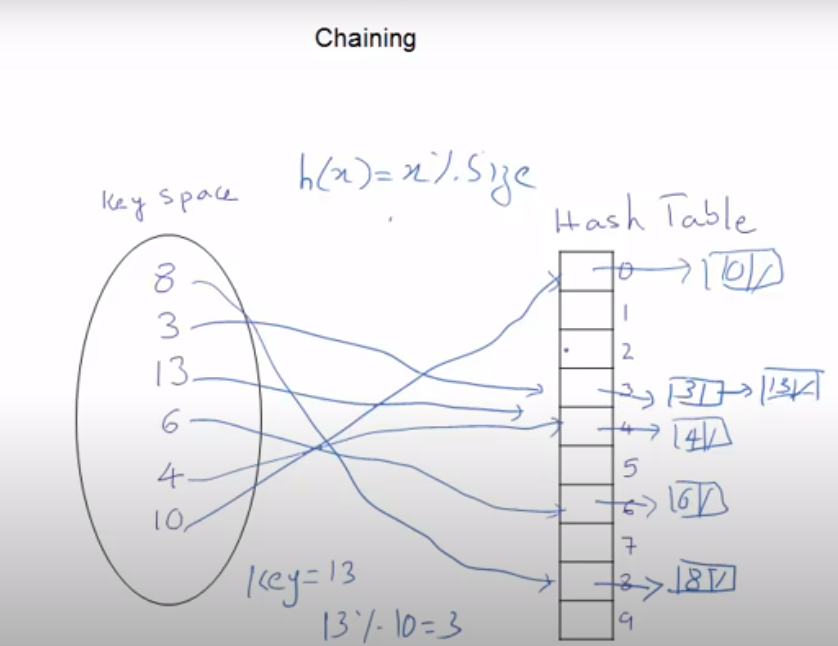
New Hash function**: h(x) = x % size of hash table (bucket size).**



Since we will get collision now with this approach so need to resolve collision.

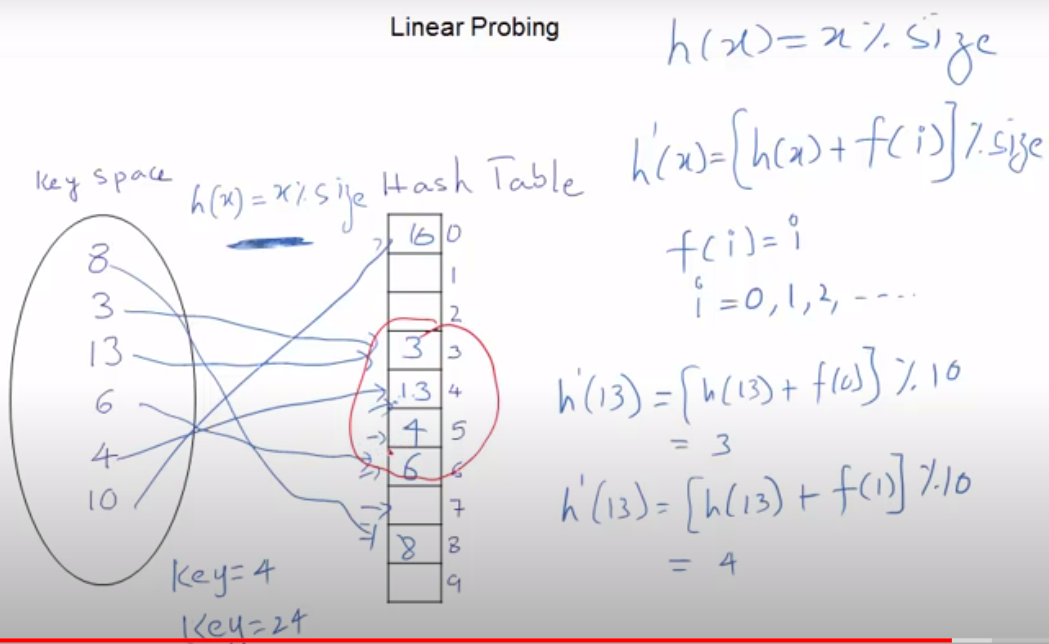


1. **Chaining method**: now we will add value in linked list for identified index value from hash function. Index value will be calculated from hashing function: h(x) = x % size

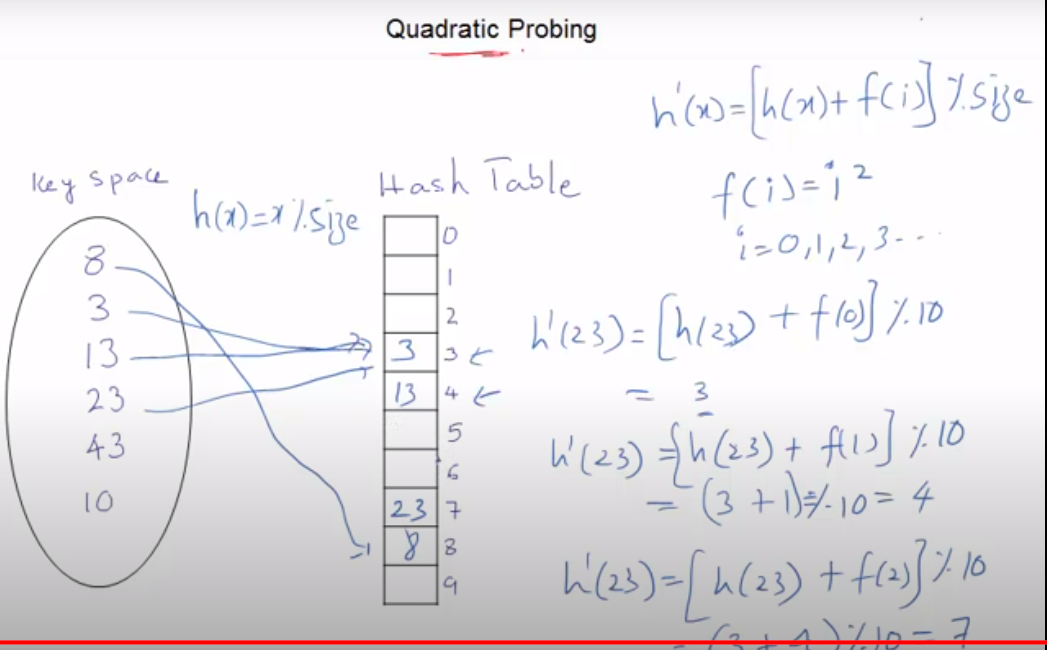


1. **Linear probing: function will be:**

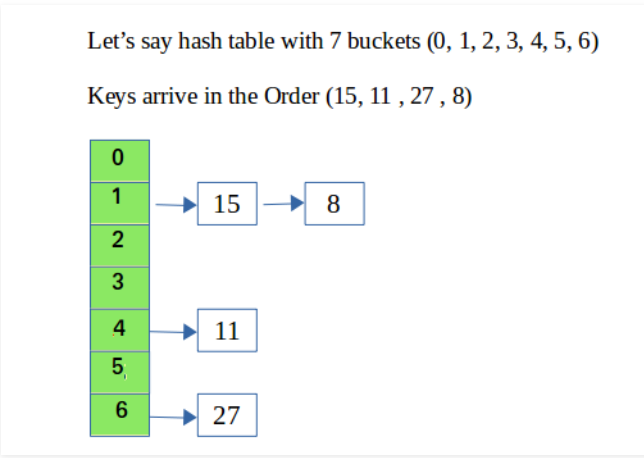
**h’(x) = [ h(x) + f(i)] % size, where f(i) = I and I = 0,1,2,3…..**



1. **Quadratic probing:**
2. **h’(x) = [ h(x) + f(i)] % size, where f(i) = I2(I square) and I = 0,1,2,3…..**



**Implementation for chaining:**



// CPP program to implement hashing with chaining

#include<bits/stdc++.h>

using namespace std;

class Hash

{

int BUCKET; // No. of buckets

// Pointer to an array containing buckets

list<int> \*table;

public:

Hash(int V); // Constructor

// inserts a key into hash table

void insertItem(int x);

// deletes a key from hash table

void deleteItem(int key);

// hash function to map values to key

int hashFunction(int x) {

return (x % BUCKET);

}

void displayHash();

};

Hash::Hash(int b)

{

this->BUCKET = b;

table = new list<int>[BUCKET];

}

void Hash::insertItem(int key)

{

int index = hashFunction(key);

table[index].push\_back(key);

}

void Hash::deleteItem(int key)

{

// get the hash index of key

int index = hashFunction(key);

// find the key in (inex)th list

list <int> :: iterator i;

for (i = table[index].begin();

i != table[index].end(); i++) {

if (\*i == key)

break;

}

// if key is found in hash table, remove it

if (i != table[index].end())

table[index].erase(i);

}

// function to display hash table

void Hash::displayHash() {

for (int i = 0; i < BUCKET; i++) {

cout << i;

for (auto x : table[i])

cout << " --> " << x;

cout << endl;

}

}

// Driver program

int main()

{

// array that contains keys to be mapped

int a[] = {15, 11, 27, 8, 12};

int n = sizeof(a)/sizeof(a[0]);

// insert the keys into the hash table

Hash h(7); // 7 is count of buckets in

// hash table

for (int i = 0; i < n; i++)

h.insertItem(a[i]);

// delete 12 from hash table

h.deleteItem(12);

// display the Hash table

h.displayHash();

return 0;

}