Data Structures

(Hash Table)

My Brief Profile

- Nahian Salsabil (NNS)
    - Lecturer, CSE, BRACU
    - BSc: CSE, BUET, 2023
    - **Email: Email:** nahian.salsabil@bracu.ac.bd
    - **Room No**. UB80601
    - **Consultation Hour**:
        - Sunday: 11:00am - 1:30pm
        - Monday: 9:30am - 10:50am, 12:30pm - 1:30pm
        - Wednesday: 9:30am - 10:50am

Links

Resources: http://tiny.cc/CSE220_Resources

Slack: http://tiny.cc/cse220_resources_NNS

Topics Covered so far

- Linear Array
- Multi Dimensional Array
- Linked List
- Stack
- Queue
- Recursion

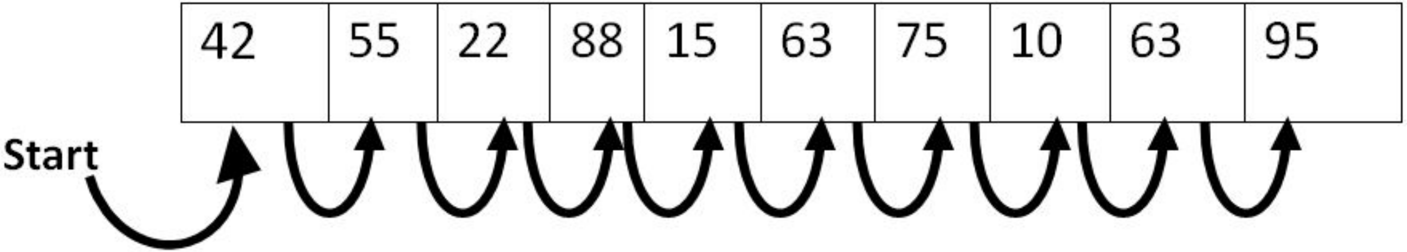
Outline

- Hash Table

Hash Table

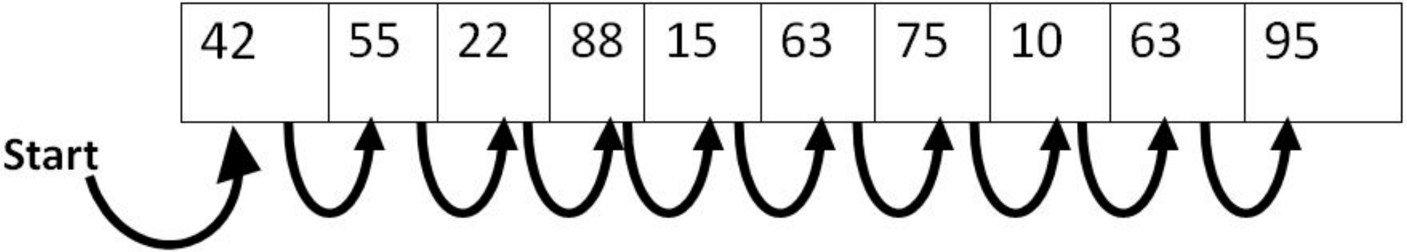
Problems of Array

- Searching O(n)



Problems of Array

- Insertion/Deletion O(n)



42	55	22	88	15	63	75	10	63	95

Problems of Linked List

- Searching O(n)
- Deletion O(n)

Task

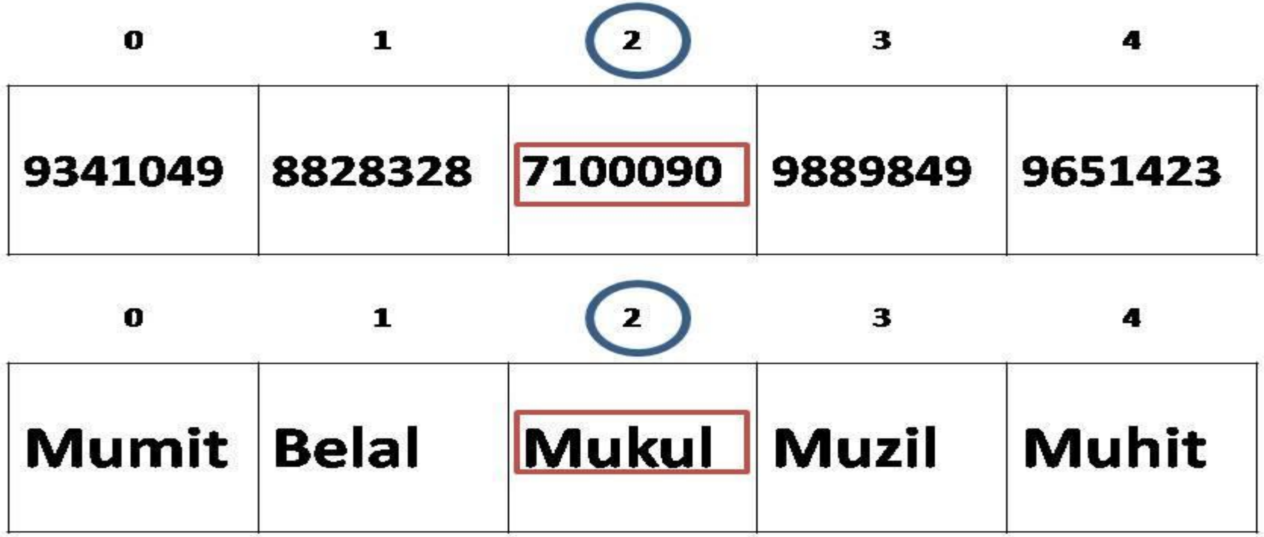
What about Sorted Array?

Think yourself!

Problems regarding Space

- Want to store username against phone numbers
- Two ways
  - Taking two arrays for numbers and names
  - Using numbers as index

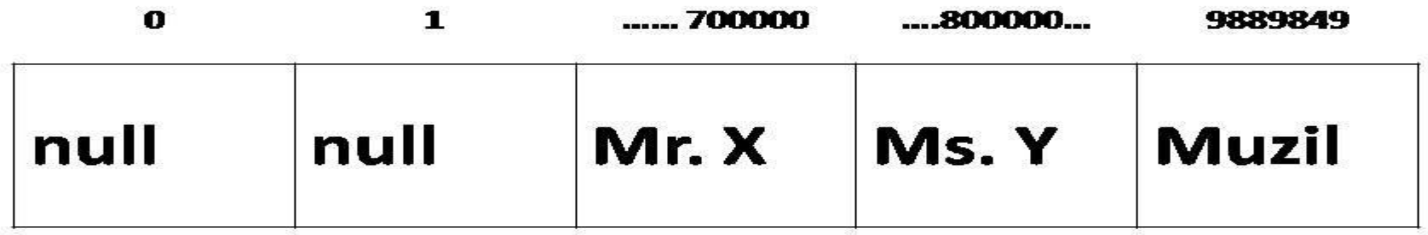
Taking Two Arrays



0	1	2	3	4
9341049	8828328	7100090	9889849	9651423

0	1	2	3	4
Mumit	Belal	Mukul	Muzil	Muhit

Using numbers as index

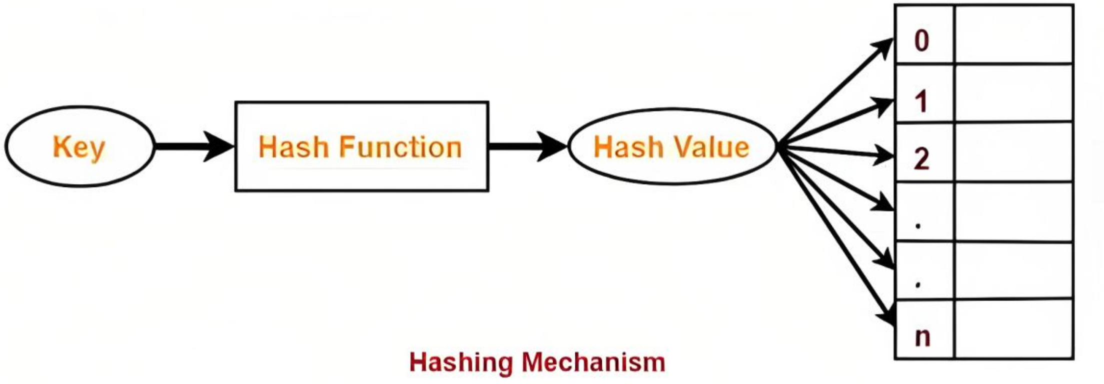


0	1	 700000	....800000...	9889849
null	null	Mr. X	Ms. Y	Muzil

How big the array should be!!!

Hash Table

- Generate hash value using **hashing/hash function**
- **Hash value** will be used as index



Hashing

- How the hash values are generated
- Several hash functions
- **Chosen wisely**

Hash Functions

- **Key % size of array**
- Example:

Key = 9889849, Value = “Muzil”

Size of the array = 10

Hash value = 9889849 % 10 = 9

So, index = 9

Hash Functions

- **Sum the digits of the key (if integer) % size of array**
- Example

Key = 9889849, Value = “Muzil”

Sum of the digits = 9+8+8+9+8+4+9 = 55

Size of the array = 10

Hash value = 55 % 10 = 5

So, index = 5

Hash Functions

- **Sum the ASCII values of the characters of the key (if string) % size of array**
- Example

Key = “ABC”, Value = 10

Sum of the ASCII values = 65+66+67 = 198

Size of the array = 10

Hash value = 198 % 10 = 8

So, index = 8

Issues

- **Example**

  Key = 9889849, Hash value = 5

  Key = 9898948, Hash Value = 5

  Index already occupied!!!

  Collision!

Solution to Collisions

- Linear Probing
- Forward Probing

Linear Probing

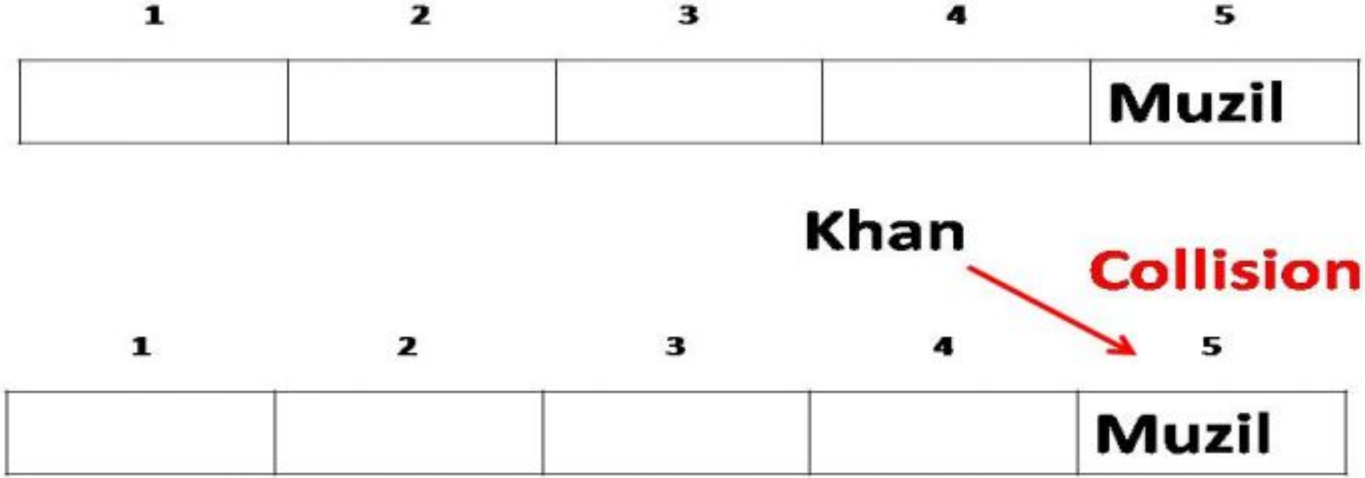
- **Example**

  Key = 9889849, Value = “Muzil”

  Key = 9898948, Value = “Khan”

  Index = 5
- Looks for the **next available space**

Linear Probing (contd)



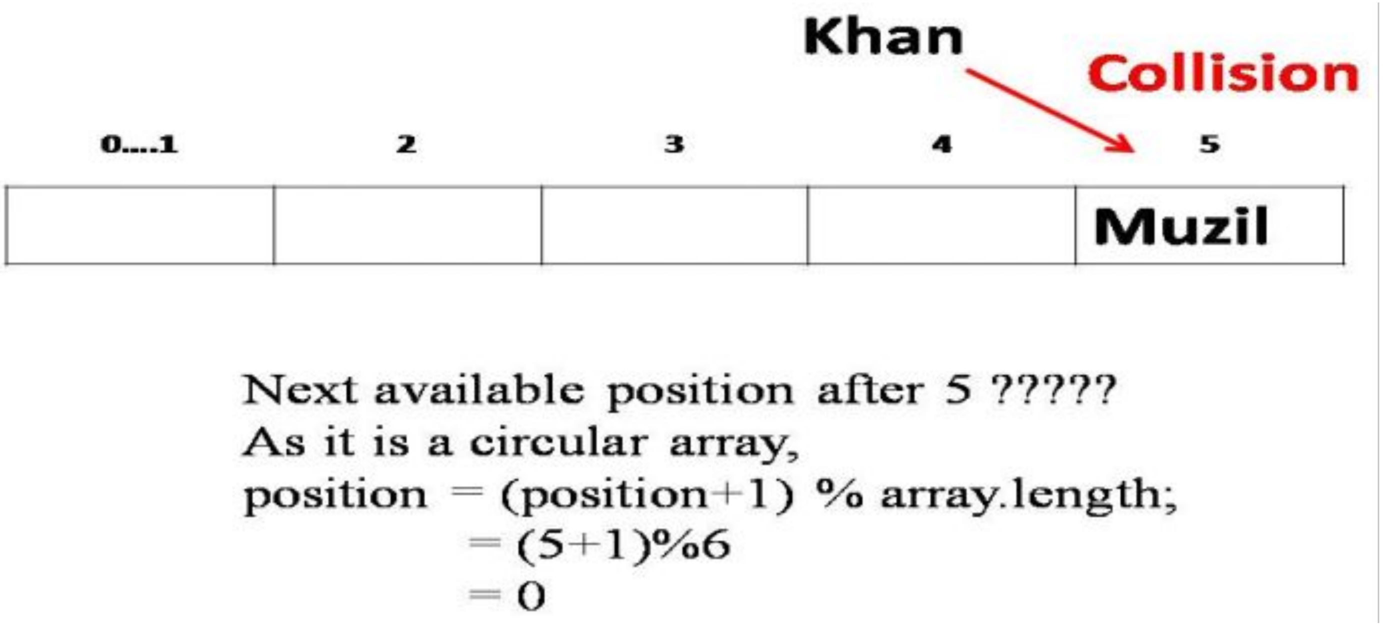
1	2	3	4	5
				Muzil

Khan

Collision

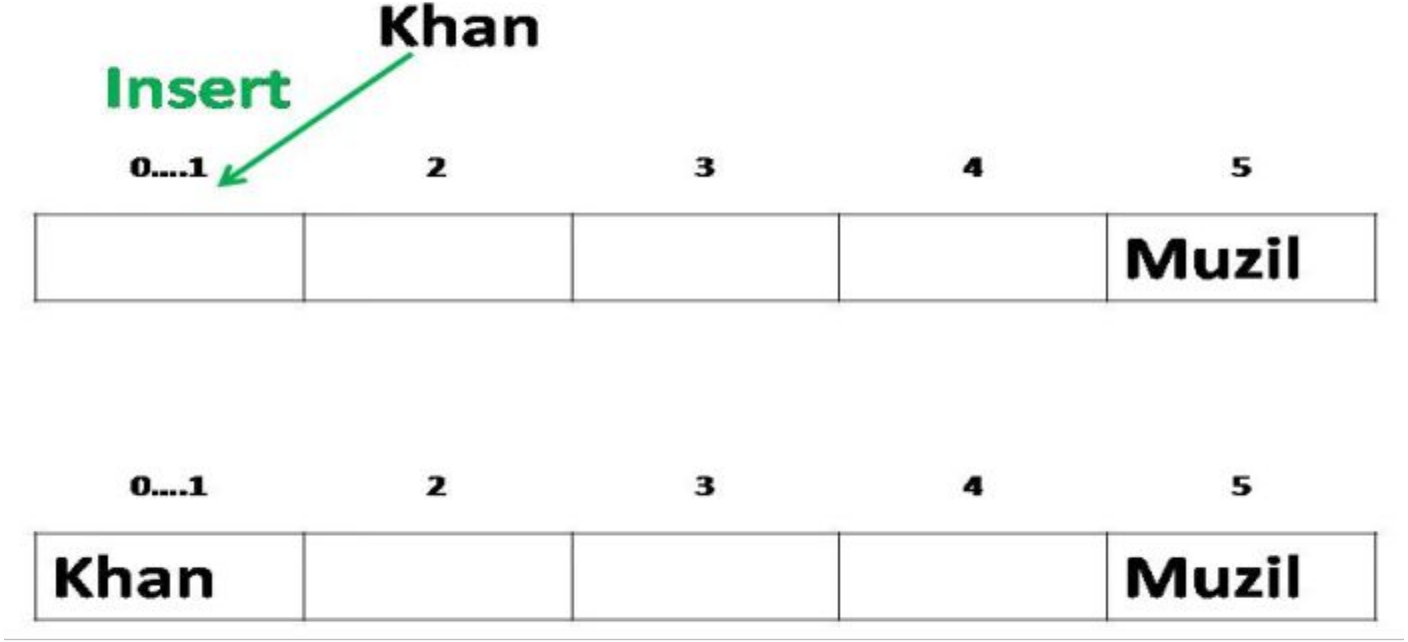
1	2	3	4	5
				Muzil

Linear Probing (contd)



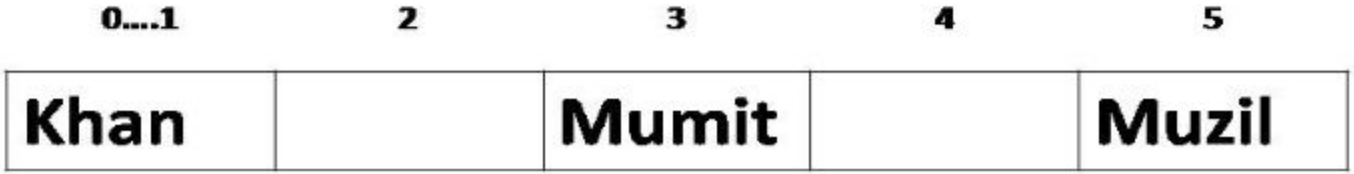
Next available position after 5 ?????
As it is a circular array,
position = (position+1) % array.length;
= (5+1)%6
= 0

Linear Probing (contd)



Linear Probing (contd)

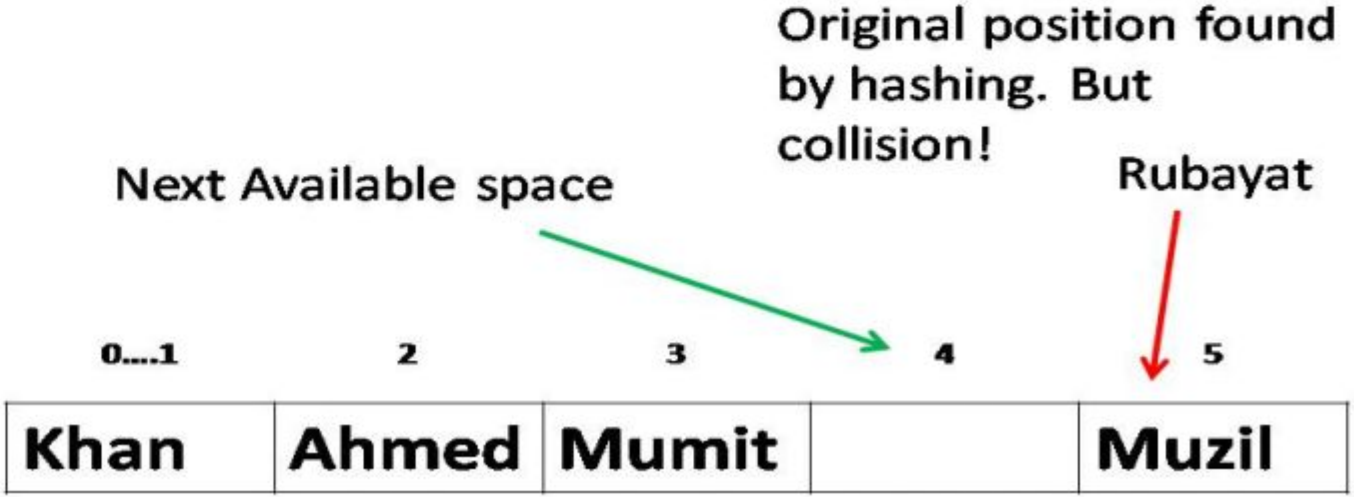
- Insert a new item



0....1	2	3	4	5
Khan		Mumit		Muzil

Linear Probing (contd)

- Insert a new item



Original position found by hashing. But collision!

Rubayat

Next Available space

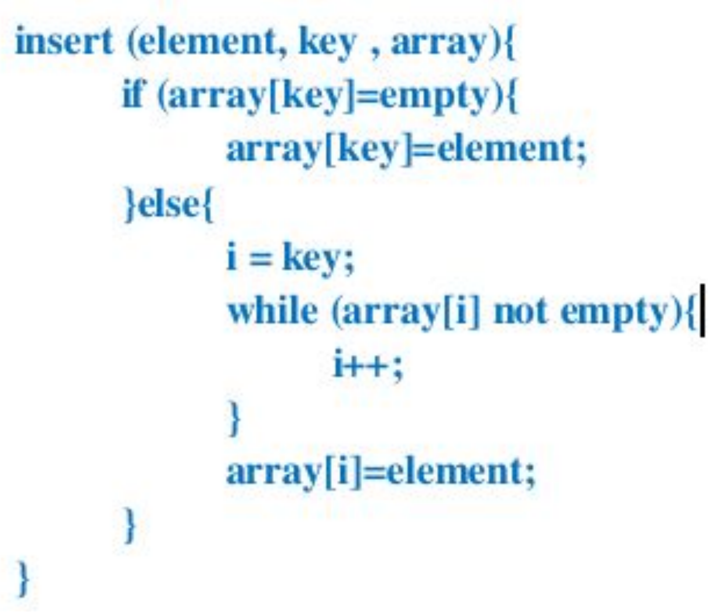
0....1	2	3	4	5
Khan	Ahmed	Mumit		Muzil

Linear Probing (contd)

- **Complexity**
  - If position empty, O(1)
  - If collision, O(n)

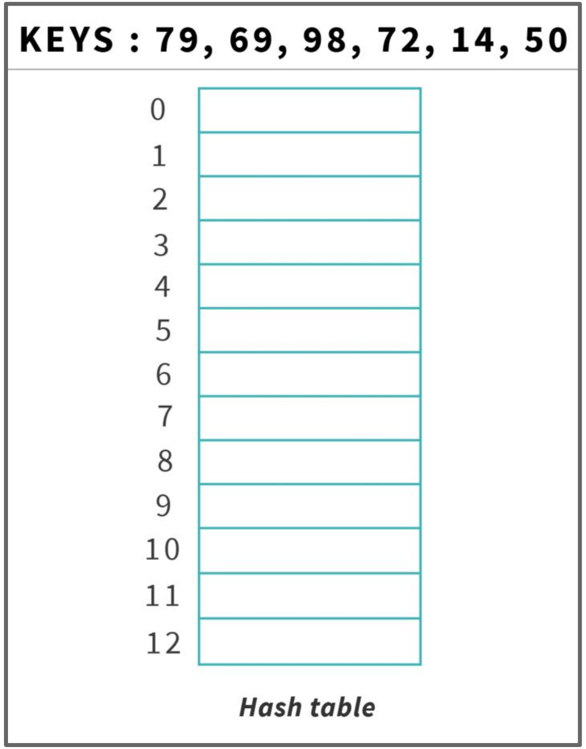
Linear Probing (contd)

- Pseudocode



```
insert (element, key , array){
        if (array[key]=empty){
                array[key]=element;
        }else{
                i = key;
                while (array[i] not empty){
                        i++;
                }
                array[i]=element;
        }
}
```

Double Hashing



KEYS : 79, 69, 98, 72, 14, 50

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Hash table

$h_1(k) = \text{key mod } 13$

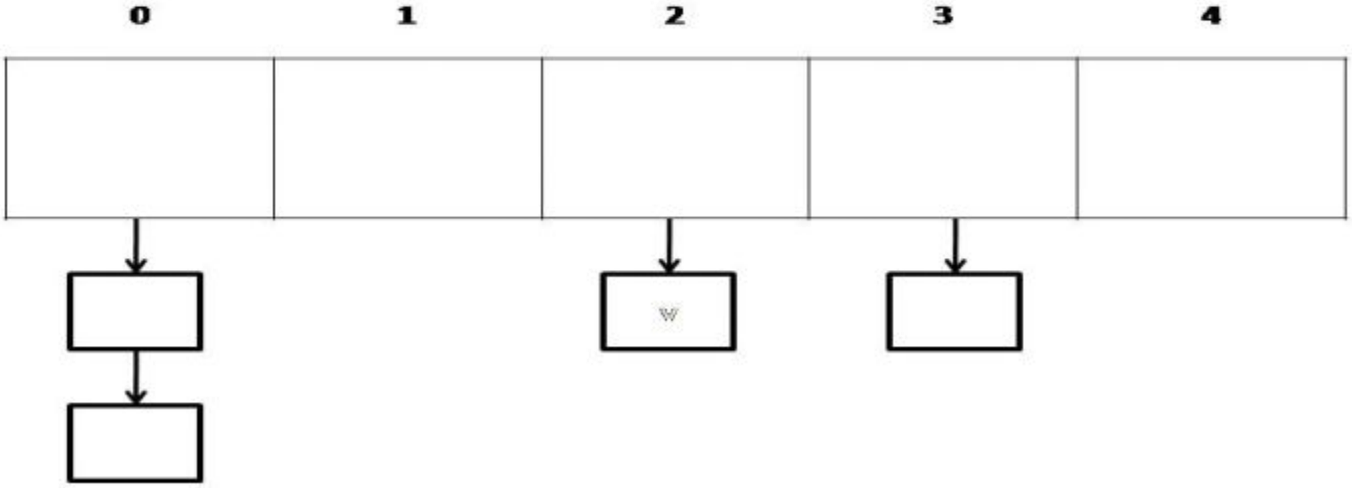
$h_2(k) = 1 + (\text{key mod } 11)$

$\text{Location} = (h_1(k) + i \text{ x } h_2(k))\%5$

i = jump times

Forward Chaining

- Each position of array is a linked list

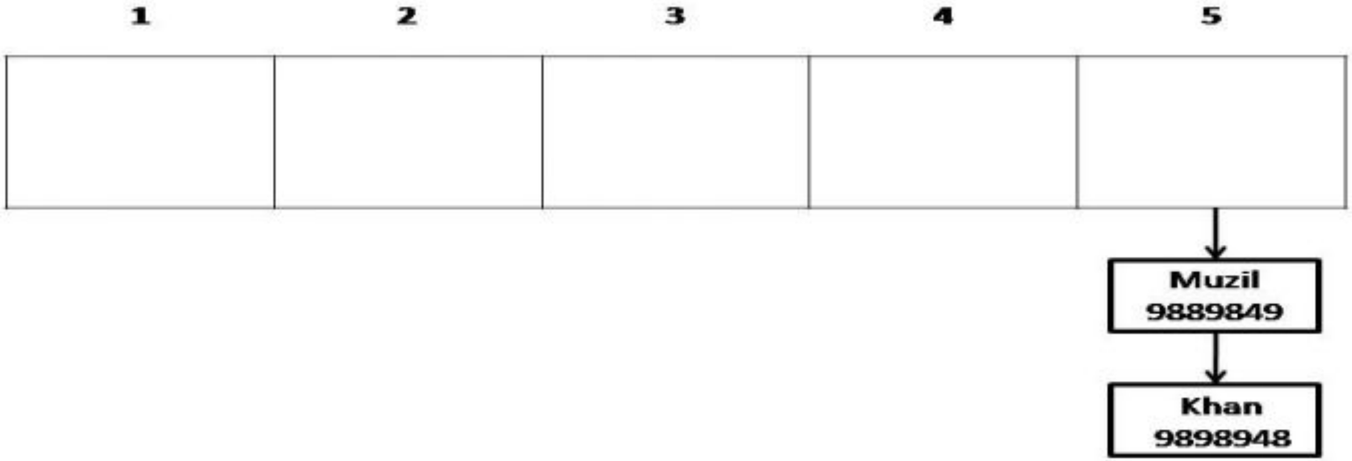


Forward Chaining (contd)

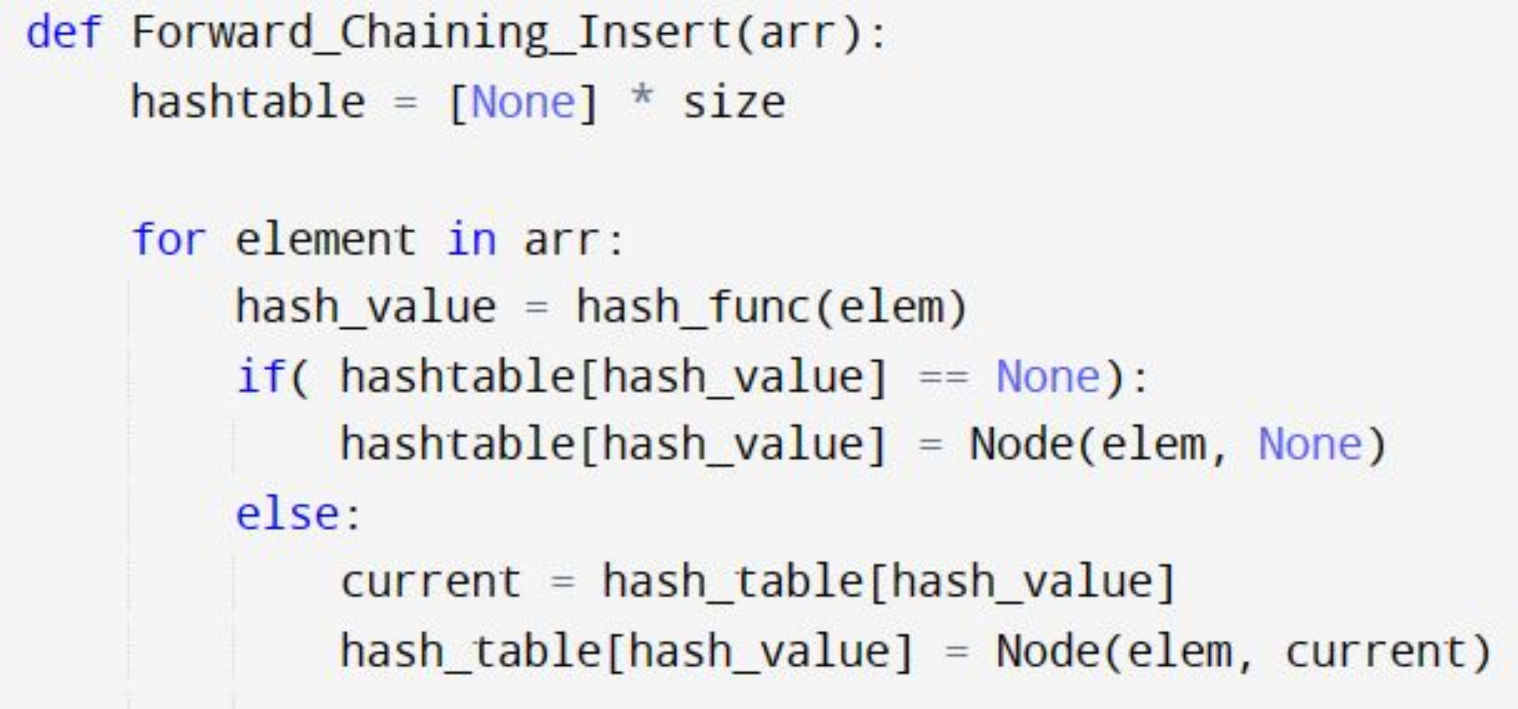
- **Example**

  Key = 9889849, Value = “Muzil”
  Key = 9898948, Value = “Khan”
  Index = 5
- Looks for the next available space

Forward Chaining (contd)



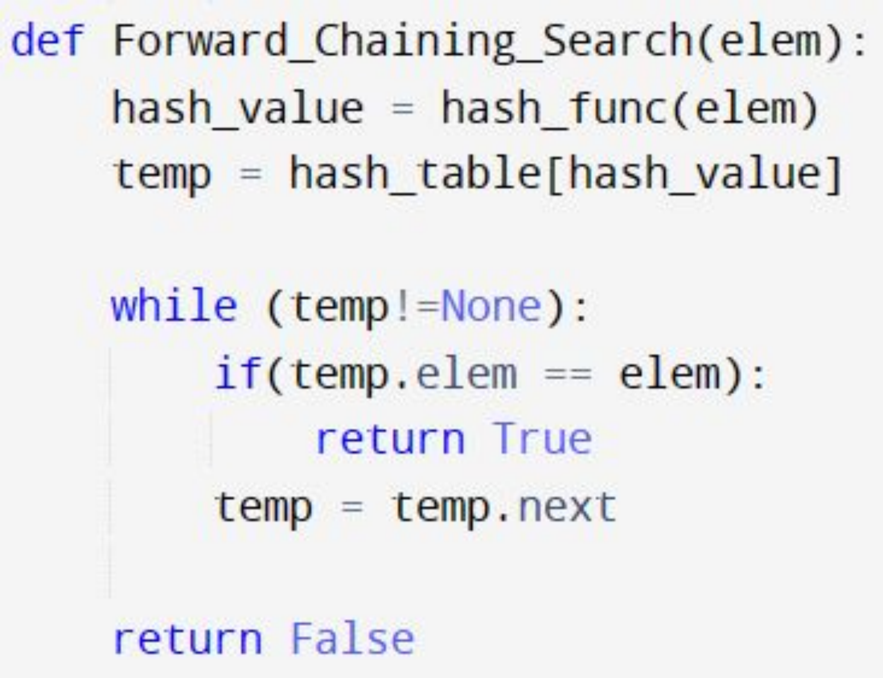
Forward Chaining Insert



```
def Forward_Chaining_Insert(arr):
    hashtable = [None] * size

    for element in arr:
        hash_value = hash_func(elem)
        if( hashtable[hash_value] == None):
            hashtable[hash_value] = Node(elem, None)
        else:
            current = hash_table[hash_value]
            hash_table[hash_value] = Node(elem, current)
```

Forward Chaining Search



```
def Forward_Chaining_Search(elem):
    hash_value = hash_func(elem)
    temp = hash_table[hash_value]

    while (temp!=None):
        if(temp.elem == elem):
            return True
        temp = temp.next

    return False
```

Key Value Pair

- **Example**

Insert (Key: 12, Value: "Apple")
Insert (Key: 5, Value: "Orange")
Insert (Key: 17, Value: "Banana")
Insert (Key: 10, Value: "Grapes")
Insert (Key: 22, Value: "Watermelon")
Insert (Key: 15, Value: "Pineapple")

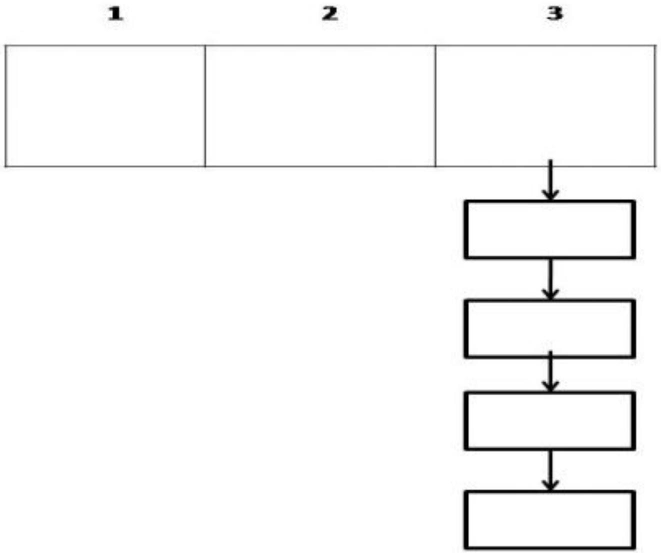
Instead of inserting only value, we will insert a tuple of key and value.
(key, value)

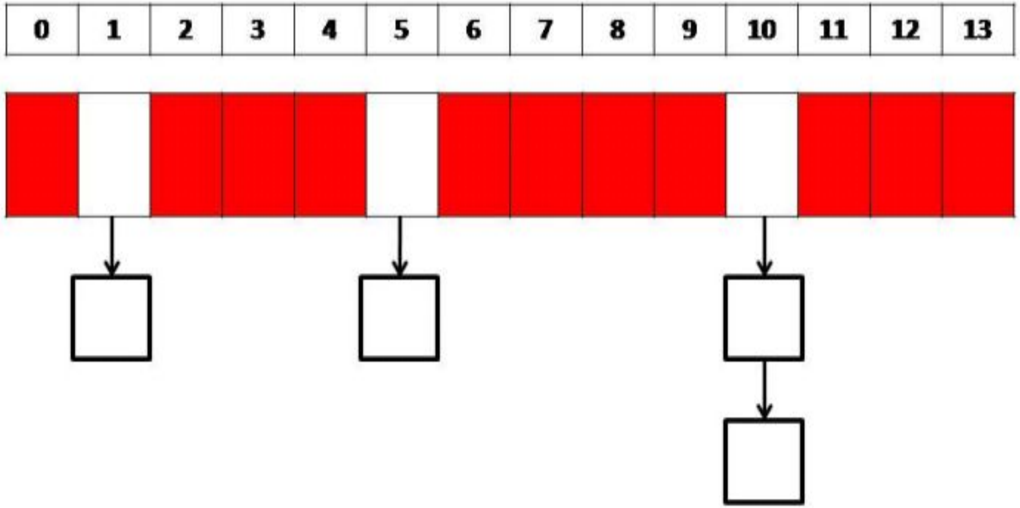
Task

- **Question:**

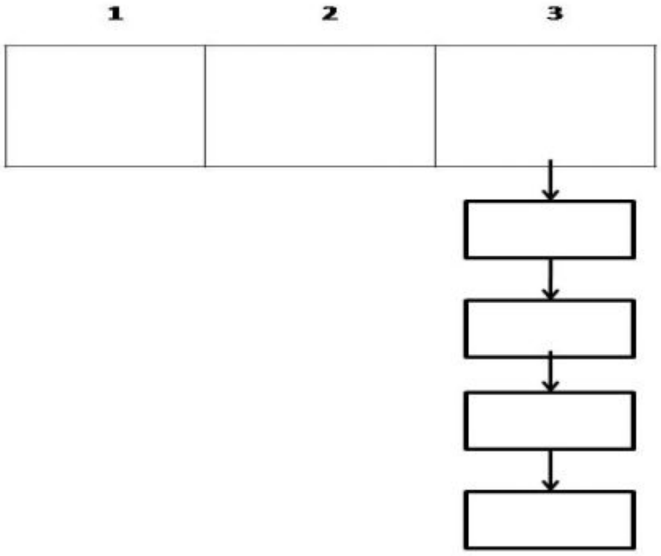
  Let’s say we have two hashtables of size 3 and 14 respectively. If we you forward chaining, which will be more efficient?

Task (contd)





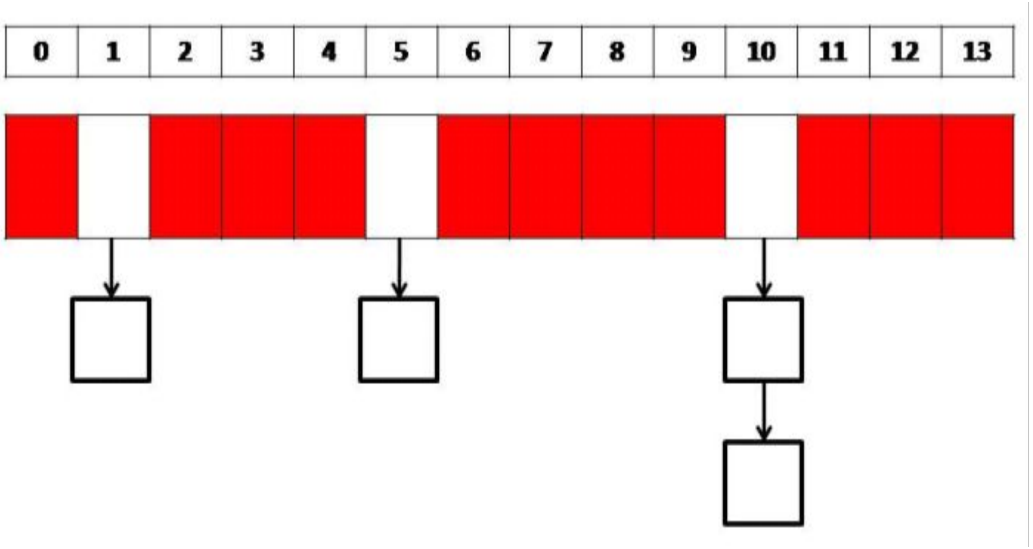
Task (contd)



Unused space = 2

Used space = 4

Task (contd)



Unused space = 11

Used space = 4

Space Wasted!!