COL106
Data Structures and Algorithms

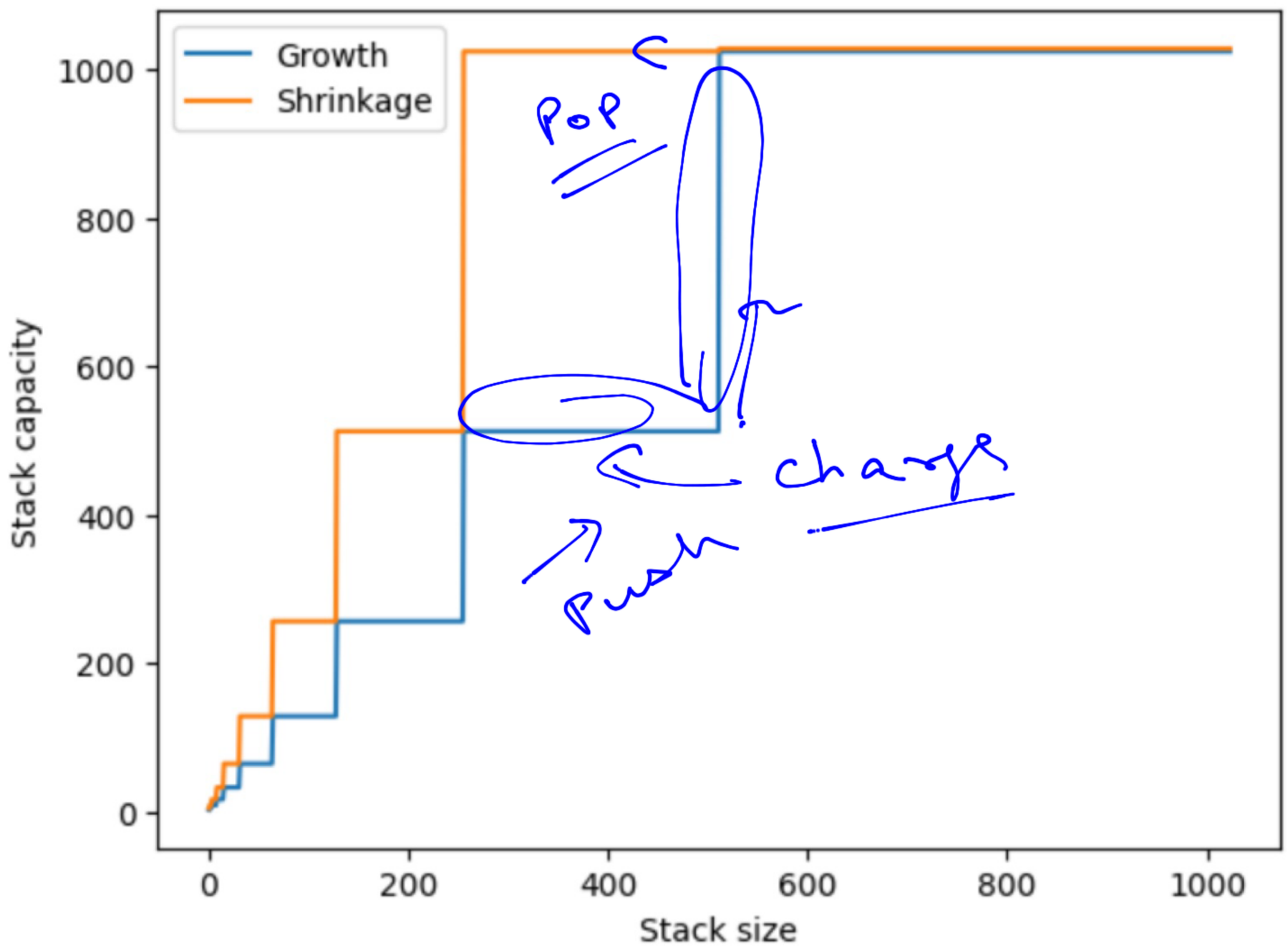
Subodh Sharma and Rahul Garg

COL106 Plagiarism Policy and Honor Code

- Copying or collaborating or use of any other unethical means in exams, quizzes is not permitted
- Every assignment will have its own honor code
  - Indicating what is permitted and what is not
  - Read the honor code carefully, ask in case of doubts
  - Violation of Honor code will lead to
    - D grade
    - Disciplinary committee (DISCO)
    - Any other penalty deemed fit by the instructors

Amortized Analysis

Dynamic Stack with Hysteresis



Amortized Analysis: Some Hints

- Consider n stack operations
- Some operations take O(1) and some O(n) time
- How to find total worst-case running-time?
- For operations that take more time
  - Pretend as if they take O(1) time
  - The balance time is “charged” to other operations done in the past or future
- For all the operations add time and charge

Dictionaries

What are Dictionaries

- Abstract Data Type (ADT)
- Contains elements that can be quickly accessed using keys
- Example: Student records at IIT Delhi
  - Entry number
  - Student name
  - Hostel name
  - Room number
  - Permanent address
  - Gradesheet
  - Other relevant information

Student Records at IIT Delhi

- Entry number
- Student name
- Hostel name
- Room number
- Permanent address
- Gradesheet
- Other relevant information
- Records may be located using the entry number which may be used as a key

A Telephone Directory

- Telephone number
- Name of the person
- Address
- Occupation
- Income
- Alternate number
- Aadhar number
- Billing details
- Plan details

True Caller Directory

True Caller Directory

- Telephone number
- List of address book entries
  - Telephone number where the address book was present
  - Name according to the address book
  - Other information about the contact in the address book
- Representative name
  - Obtained after combining all the address book entries
- True caller functions
  1. Send address book entries from mobile phones to a centralized repository
  2. Synthesis all the names from multiple address books
  3. Quickly find representative name from telephone number

What is the Next Big Idea?

Operations on the Dictionaries

- Constructor/destructor
- size()
- isEmpty()
- elements()
- find(key)
- insert(e)
- remove(key)
- findAll(key)
- removeAll(key)

Dictionary ADT

```
class Element {
  public:
    type1 key;
    type2 val;
};
```

```
class Dictionary {
  public:
    Dictionary(int sz); // Create an initial dictionary of size sz
    ~Dictionary();
    int size(); // Returns the number of elements in the Dictionary
    Element * elements(); // Returns elements of the dictionary
```

Dictionary ADT

```
class Dictionary {
  public:
    Dictionary(int sz); // Create an initial dictionary of size sz
    ~Dictionary();
    int size(); // Returns the number of elements in the Dictionary
    Element * elements(); // Returns elements of the dictionary

    void insert(Element *e); // Insert into the dictionary
    Element * find(type1 key); // Finds an element in the
    // dictionary corresponding to the key
    void remove(type1 key); // Deletes an element in the
    // dictionary corresponding to the key
};
```

Implementing Dictionaries

- Arrays unsorted
- Arrays sorted
- Linked lists
- Hash tables
- Binary trees
- Red-black trees
- AVL trees
- B trees

Implementation of Dictionary

- Using arrays (unsorted)

Dictionaries using Arrays (unsorted)

```
class uArrayDictionary: public Dictionary {
  private:
    Element * e;
    int capacity, count;
  public:
    uArrayDictionary(int sz):Dictionary(sz) {
        capacity = sz;
        count = 0; // Elements are stored in e from 0 to count-1
        e = new Element[sz];
};
~uArrayDictionary() {
        capacity = 0;
        count = 0;
        delete e;
};
```

Dictionaries using Arrays (unsorted)

```
void insert(Element *ein) {
    e[count++] = *ein; // Insert the new element at the end
    // Error checking needed here when count == capacity
}
Element *find(type1 key) {
    int i;
    for (i = 0; i < count; i++)
        if (e[i].key == key) return &e[i]; // Key found
    return 0; // Key not found
}
```

Dictionaries using Arrays (unsorted)

```
void *remove(type1 key) {
    int i, j;
    for (i = 0; i < count; i++)
        if (e[i].key == key) { // Key found
            // Now delete the element e[i] and shift all the
            // elements after e[i] to one position before
            for (j = i+1; i < count; j++) {
                e[j-1] = e[j];
            }
            count--; // Decrease the count of elements
        }
    // End of function. Key not found
}
```

Dictionaries using Arrays (unsorted)

- Assume n insert/remove/find operations already performed
- Time complexity for the next operation
  - Insert: O(1)
  - Find: O(n)
  - Remove: O(n)

Implementation of Dictionary

- Using arrays (sorted)

Dictionary Using Sorted Arrays

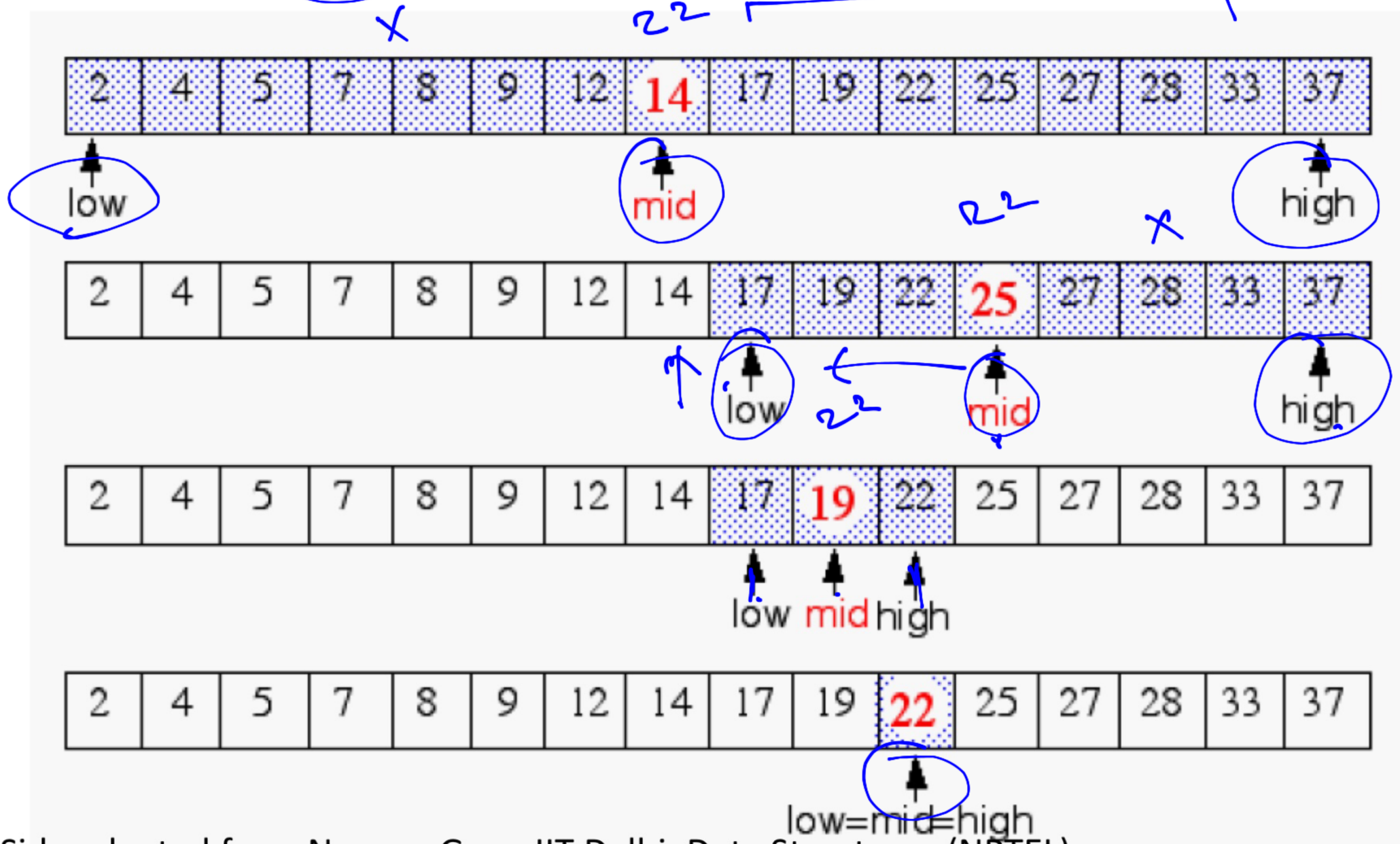
```
class sArrayDictionary: public Dictionary {
  private:
    Element * e;
    int capacity, count;
  public:
    sArrayDictionary(int sz):Dictionary(sz) {
        capacity = sz;
        count = 0;     // Elements are stored in e from 0
        e = new Element[sz];
    };
    ~sArrayDictionary() {
        capacity = 0;
        count = 0;
        delete e;
    };
```

Dictionary Using Sorted Arrays

- How to find an element?
- Use binary search
- Divide and conquer algorithm
  - Divide complex problem into smaller simpler sub-problems
  - Solve smaller subproblems
  - Combine solutions of the smaller sub-problems
  - Arrive at a solution to the complex big problem

Binary Search in a Sorted Array

Assume find(22) is to be performed



Side adapted from Naveen Garg, IIT Delhi, Data Structures (NPTEL)

Search in a Sorted Array: Iterative

```
Element *find(type1 key) { // Use binary search to find
    // Since all the elements are sorted
    int lo = 0, hi = count-1, mid = (lo + hi)/2;
    while (lo <= hi) {
        if (e[mid].key == key) // Key found
            return &e[mid];
        else if (e[mid].key < key)
            low = mid + 1;
        else // Invariant e[mid].key > key now
            hi = mid - 1;
        mid = (lo + hi) / 2;
    }
    return 0; // Key not found
}
```

Search in a Sorted Array: Recursive

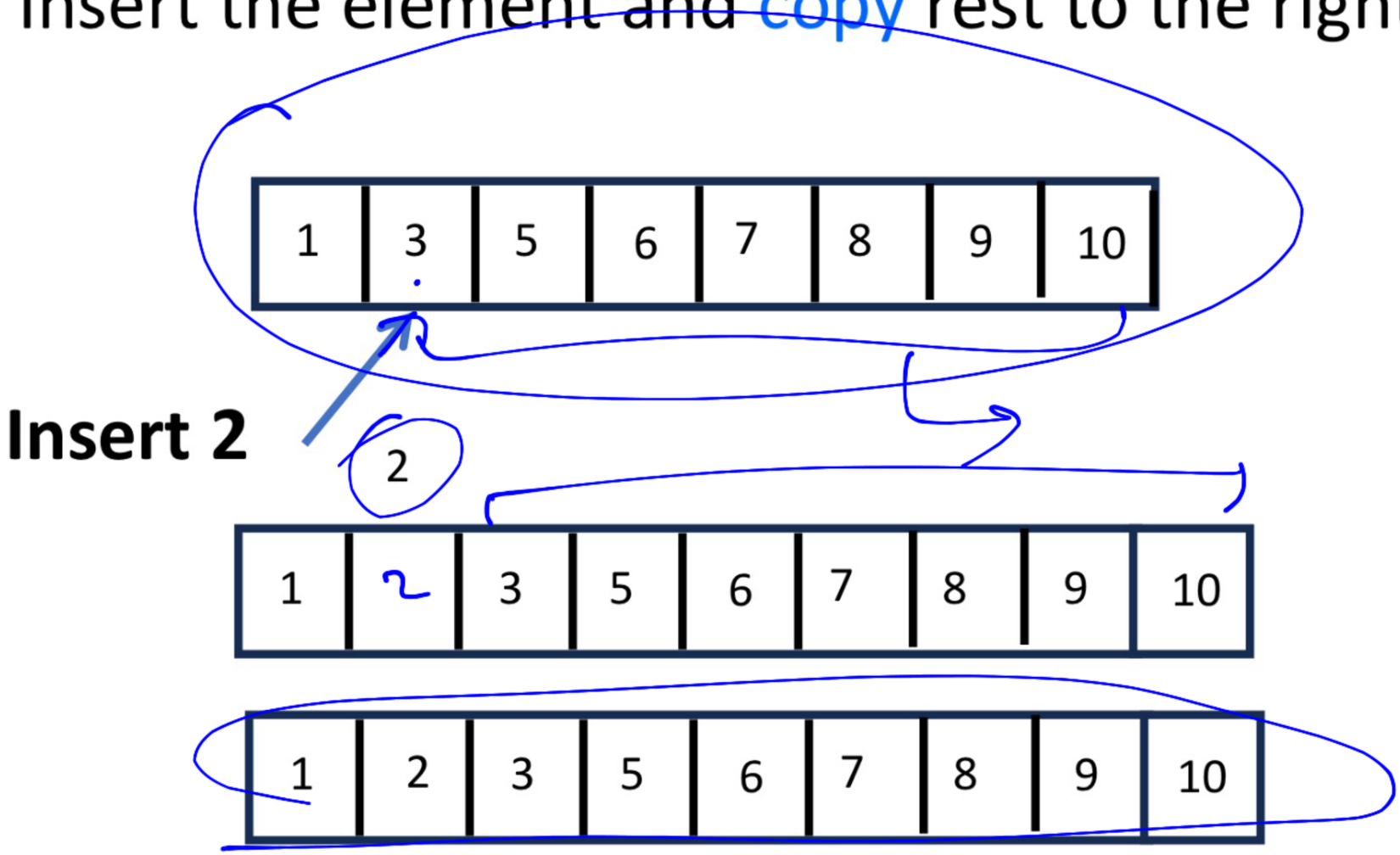
```
Element * find_recursive(element *e, type1 key, int lo, int hi) {
    int mid;
    if (lo > hi) return 0;
    mid = (low + hi) / 2;
    if (e[mid].key == key) return &e[mid];
    if (e[mid].key < key)
        return find_recursive(e, key, mid+1, hi);
    else
        return find_recursive(e, key, lo, mid-1);
}
```

Dictionary Using Sorted Arrays

- How to insert an element?

Sorted Arrays: Insertion

- Use binary search to find location to be inserted
- Insert the element and copy rest to the right



Dictionaries using Sorted Arrays: Insertion

```
void insert(Element *ein) { // First perform binary search
     int lo = 0, hi = count-1, mid;
     int j;
     Element *tmp;
     while (lo <= hi) {
          mid = (lo + hi) / 2;
          if (e[mid].key == key) // Key found
               break;
          else if (e[mid].key < key)
               low = mid + 1;
          else // Invariant e[mid].key > key now
               hi = mid - 1;
}
// Possible cases: (i) lo <= hi ==> e[mid].key == key.
// So insert after e[mid]
// (ii) lo > hi ==> e[hi].key < key. Insert after e[hi]
```

Dictionaries using Sorted Arrays: Insertion

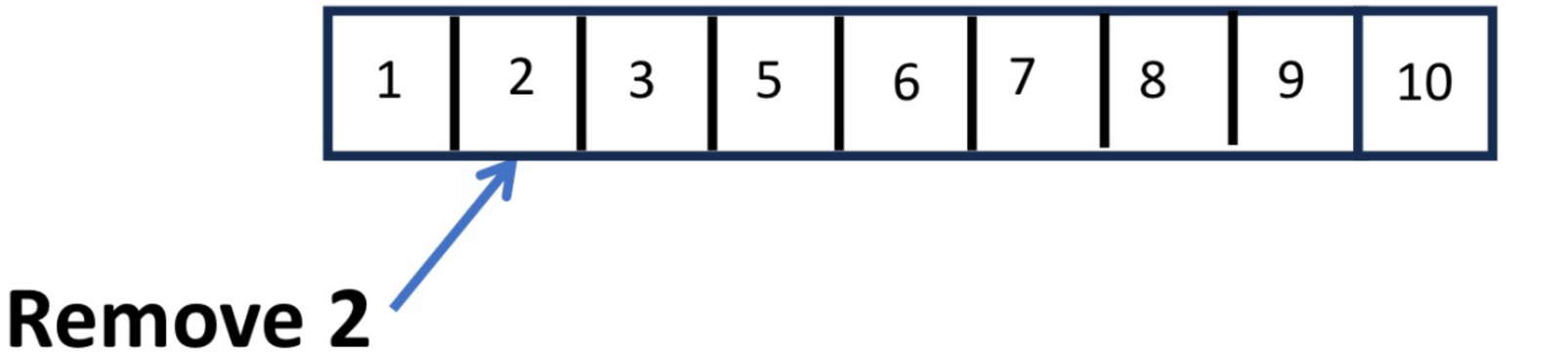
```
// Possible cases: (i) lo <= hi ==> e[mid].key == key.
// So insert after e[mid]
// (ii) lo > hi ==> e[hi].key < key. Insert after e[hi]
if (lo <= hi) j = mid + 1;
else j = hi + 1;
while (j <= count) {
     tmp = e[j];
     e[j] = ein;
     ein = tmp;
     j++;
}
count++;
```

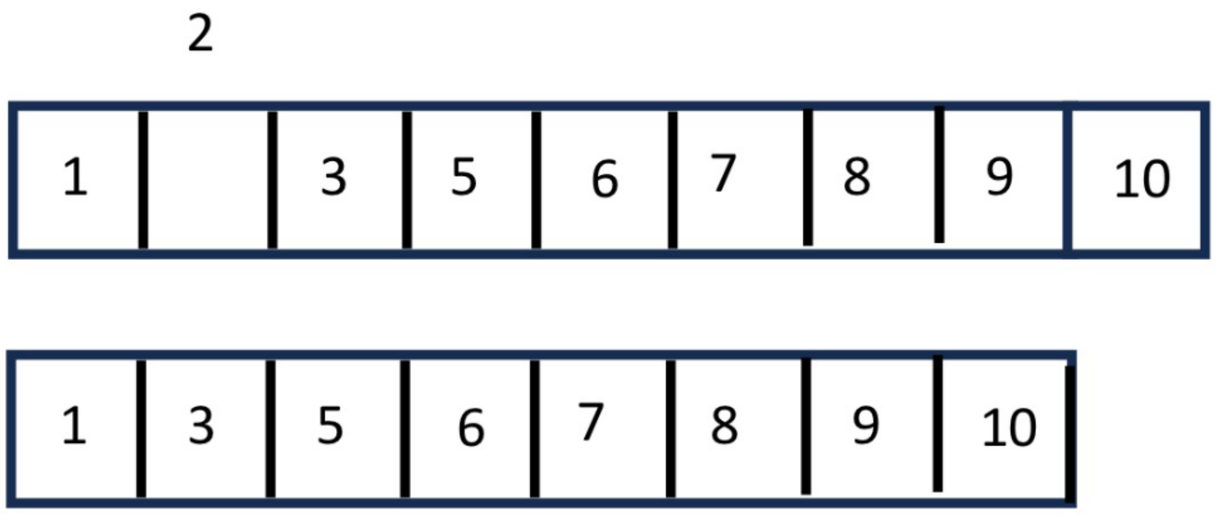
Dictionary Using Sorted Arrays

- How to remove an element?

Sorted Arrays: Removal

- Use binary search to find location to be removed
- Remove the element and copy rest to the left





Dictionaries using Sorted Arrays: Removal

```
void *remove(type1 key) {
    Element *efound;
    int i, j;
    efound = find(key);
    if (efound == 0) // Key not found. Nothing to do.
        return;      // TBD error reporting. Raise exception
    i = efound - e;  // i is now index into e where key was found
    for (j = i+1; i < count; j++) {
        e[j-1] = e[j];
    }
    count--; // Decrease the count of elements
}
```

Dictionary Using Sorted Arrays

- Assume n insert/remove/find operations
- Time complexity for the next operation
- Insert: O(log(n)) + O(n) = O(n)
- Find: O(log(n))
- Remove: O(log(n)) + O(n) = O(n)

Implementation of Dictionary

- Using doubly linked lists (unsorted)
- Insert: InsertAfter(header) or InsertBefore(trailer)
- Find(key): Scan through the DLL to locate a note matching the key
- Remove
  - node=Find(key)
  - DLLDelete(node)

Dictionary using Unsorted DLL

- Assume n operations (insert, remove, find)
- Time complexity for the next operation
- Insert: O(1)
- Find: O(n)
- Remove: O(n)

Dictionary using Sorted DLL

- Time complexity of the following operations?
- Insert(element): O(n)
- Find(key): O(n)
- Remove(key): O(n)

Dictionary using Sorted DLL

- Insert(element): Scan through the elements till you find the right position and then insert
- Find(key): Scan through the elements till you find the key
- Remove(key): Scan through the elements till you find the key
- Where to scan from? Does it really matter?
  - Header
  - Trailer
- T(N) is the worst-case time for any input of size N

Dictionary using Sorted DLL

- Insert() best and worst cases
    - Scan from header worst case
    - Scan from header best case
    - Scan from trailer worst case
    - Scan from trailer best case

Dictionaries using Skip Lists

- Homework: What will be the expected worst case time complexity for the following operations if you were to use skip lists to implement dictionaries?
- Insert
- Find
- Remove

Dictionary using a Hash Table

- What will be the expected worst case time complexity for the following operations if you were to use hash table to implement dictionaries?
  - Insert
  - Find
  - Remove

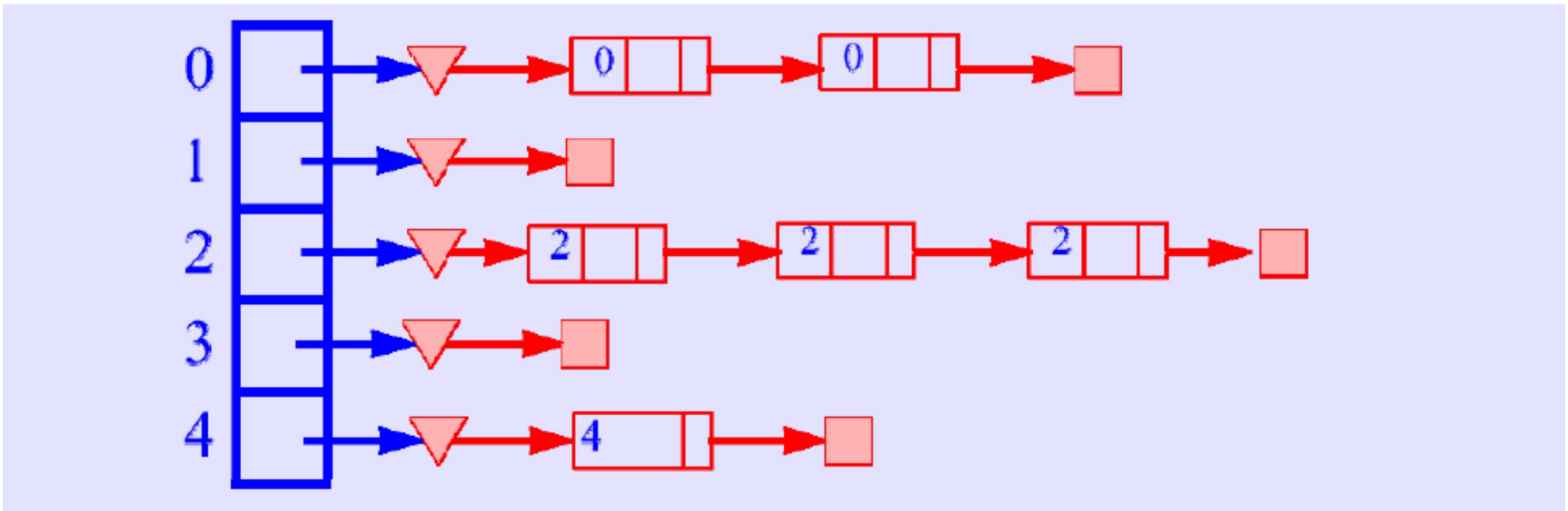
h(key) : 0 -- m-1

Hash function

Hash table of size m

Analysis of Hash Table using Chaining

- Number of elements in the Dictionary: n
- Size of Hash table: m
- Store the elements using a linked list in case of collision

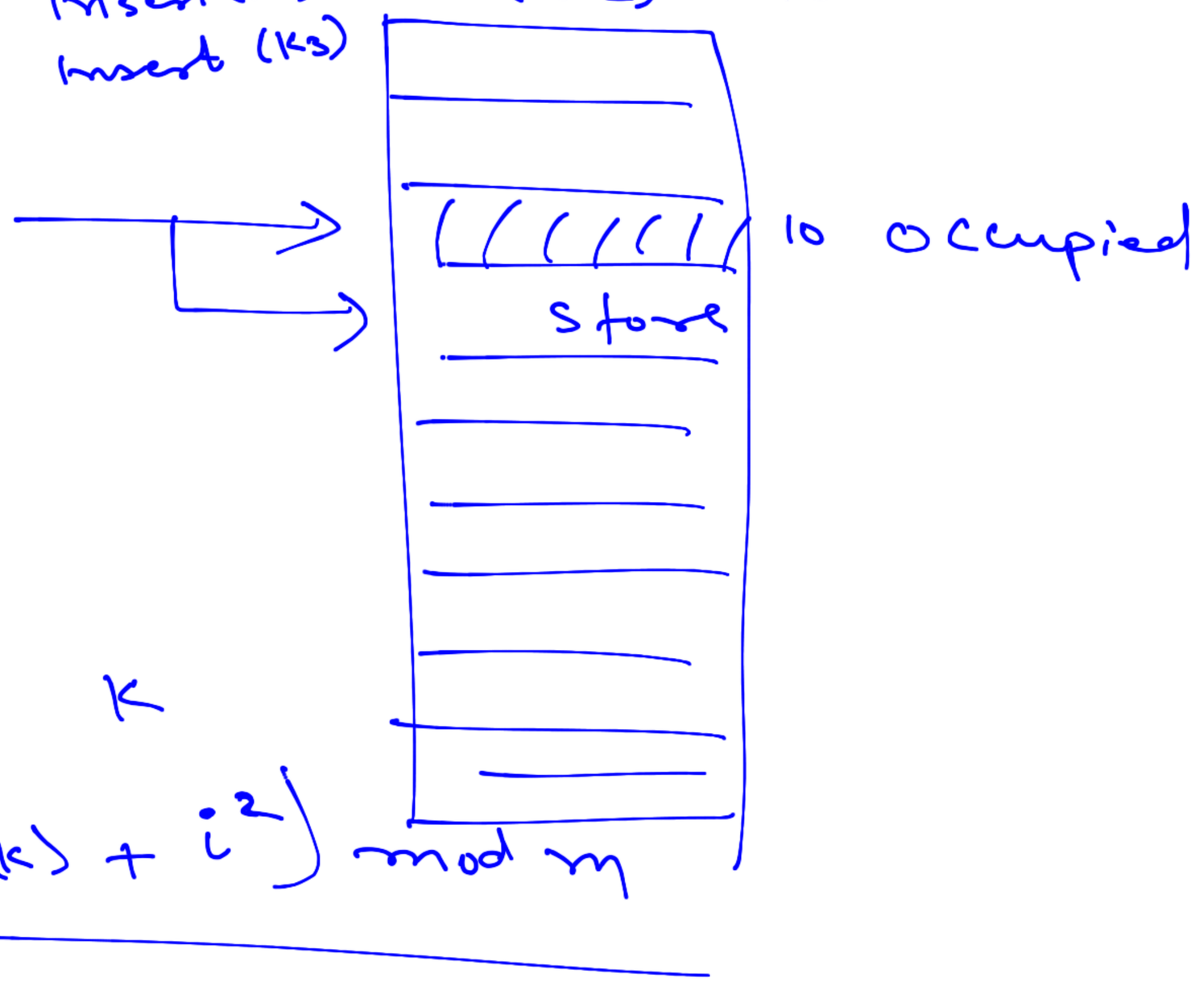


Linear Probing

insert(K1) - h(K1) = 10 h(K3) = 10
insert(K2) h(K2) = 10
insert(K3)

h(key)

Insert (key)

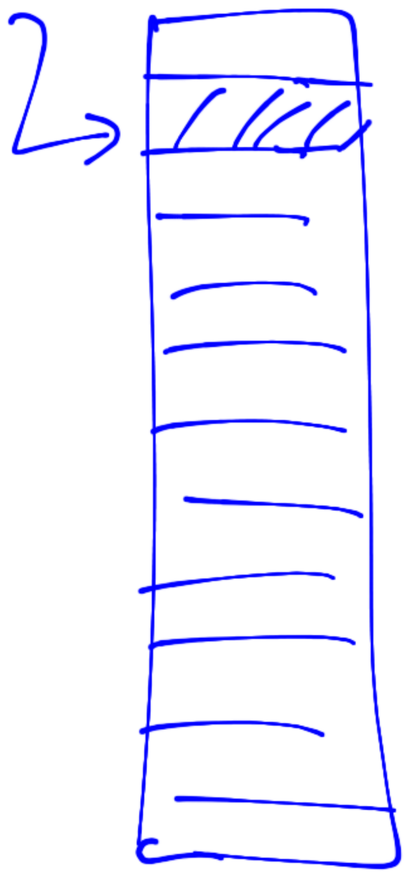


h(k)

ith probe for k

Look at $(h(k) + i^2) \bmod m$

$h(K) = 2$



$h(k) + i$: 3

$2+4$: 6

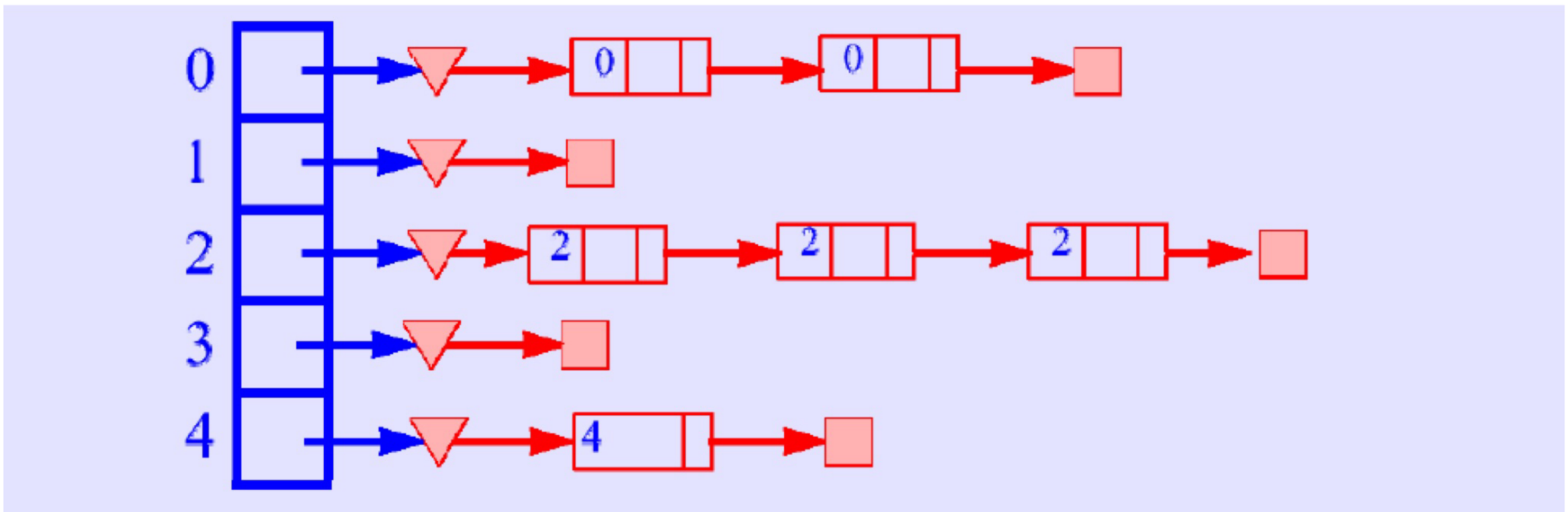
$2+9$: 11

$h(k) + ai + b$

choose a, b coprime to m

Analysis of Hash Table using Chaining

- Number of elements in the Dictionary: n
- Size of Hash table: m
- Average length of list: $\alpha = n/m$



Hash Table using Chaining

- Successful search time
  - O(1) for computing hash
  - $\alpha/2$ for searching in the list (on average)
  - Total time is $O(1+\alpha/2)$
- Unsuccessful search time
  - O(1) for computing hash
  - α for searching in the list (on average)
  - Total time: $O(1+\alpha)$
- Insert time
  - O(1) for computing hash
  - O(1) for inserting in the list
- Time for all operations is O(1) if m = c n

Dictionary using a Hash Table

- Homework: What will be the expected worst case time complexity for the following operations if you were to use hash table (chaining, linear probing, quadratic probing, double hashing) to implement dictionaries?
  - Insert
  - Find
  - Remove

Universal Hashing

A set S of hash functions is universal if for all inputs k_1 k_2.

$$\Pr_{h_i \in S}\left[h_i(k_1) = h_i(k_2)\right] \leq \frac{1}{m}$$

m: Size of hash table