CHAPTER 8: SEARCHING

BIBHA STHAPIT
ASST. PROFESSOR
IoE, PULCHOWK CAMPUS

Searching

- The process of retrieving some particular information from a large amount of previously stored information.
- The information can be in sorted or unsorted form.
- Normally we think of the information as divided up into records, each record having a key for use in searching.
- The goal of the search is to find all records with keys matching a given search key.
- The purpose of the search is usually to access information within the record for processing.
- It is not compulsory that the searched key values are found in the records.

Searching (Types)

- Searching generally falls in two categories:
  - Internal Search
  - External Search
- Internal Search:
  - If the data to be searched are all present in main memory, then the searching becomes internal search
  - Internal Searches are faster than External Search and hence are recommended whenever possible
  - They are mainly done among the data which occupy less space compared to the space of RAM
- External Search:
  - If most of the data to be searched are in auxiliary memory, then the search becomes External Search.
  - If the data are very large and our main memory is not large enough to hold them all during the process, then the external search is used.

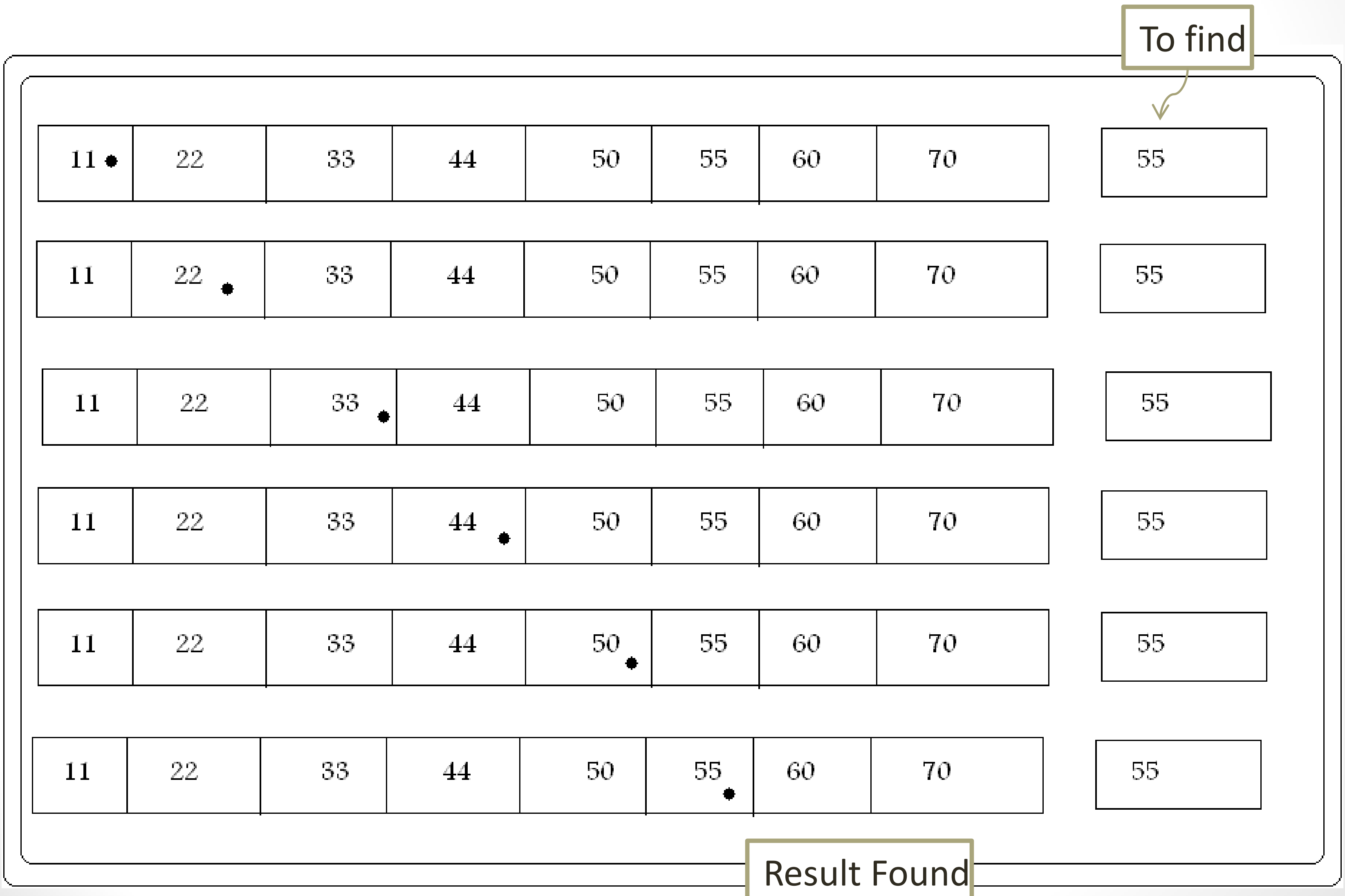
Searching Algorithms

- Different searching algorithms are used
- The choice of proper algorithm depends upon the way the data are arranged
- Any algorithm technique may be better than another according to the favorable way the data are arranged for it
- We are going to study the following 3 searching techniques:
  - Linear Search
    - For unsorted data in linear structure
  - Binary Search
    - For sorted data in linear structure
  - Tree Search
    - For data maintained in search trees

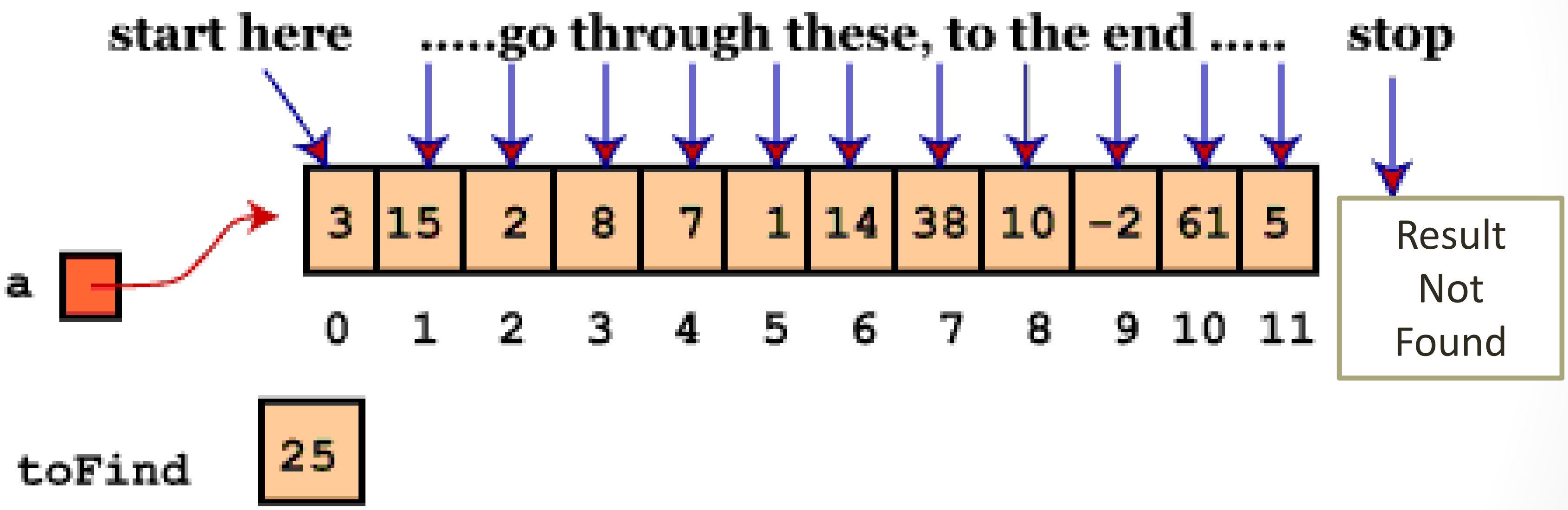
Linear Search

- Also called **Sequential Search**
- Simplest among all
- Applicable for data organized in form of array or linked list
- It is applicable for small data values
  - Each element in the array is compared with the value to be searched
  - If the values are matched, then the search is successful
  - Otherwise the comparison is kept on doing until all the values are compared
  - By the end of the comparison if the value in the array is not matched with the value to be searched, then the search is considered unsuccessful

Linear Search (Example)



Linear Search (Example)



Linear Search (Algorithm)

```
Declare and initialize necessary variables
   n; a[n]; item-to be searched from array
   flag=0  for determining the success of search
For i=0 to n-1
   if a[i]=item
          display "Search Successful"
          flag=1
          stop
   end if
End for
If flag=0
   display "Search Unsuccessful"
End if
```

Linear Search (Algorithm)

```
LINEAR_SEARCH(A, N, VAL)
Step 1: [INITIALIZE] SET POS = -1
Step 2: [INITIALIZE] SET I = 1
Step 3:     Repeat Step 4 while I<=N
Step 4:             IF A[I] = VAL
                          SET POS = I
                          PRINT POS
                          Go to Step 6
                    [END OF IF]
                     SET I = I + 1
            [END OF LOOP]
Step 5: IF POS = -1
        PRINT "VALUE IS NOT PRESENT
        IN THE ARRAY"
        [END OF IF]
Step 6: EXIT
```

Algorithm for linear search

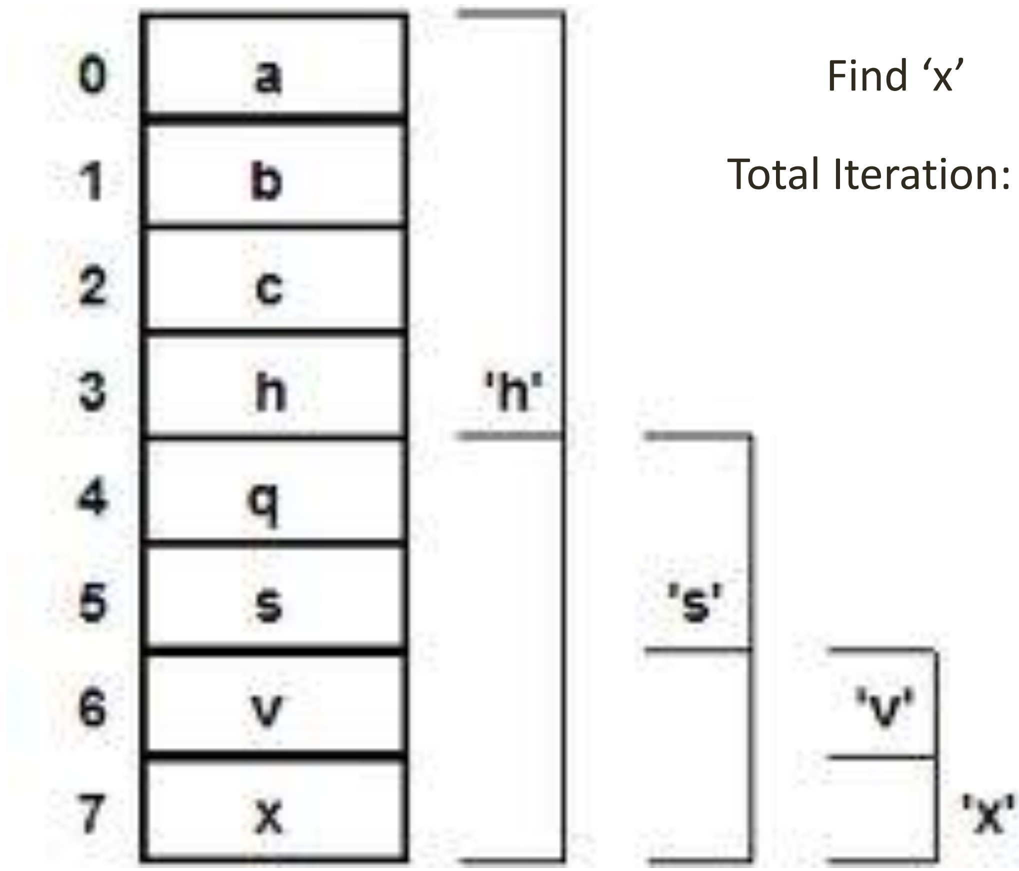
Linear Search

- It is considered simple and is very applicable when searching for small data
- It is good in searching for unsorted data
- Whereas,
- It is slower as compared to other searching algorithms
- It is applicable only for small amount of data

Binary Search

- If the data items are presented in sorted form (i.e. ascending or descending), then this algorithm is used
- It is much more efficient algorithm than the general linear search for the sorted data
- Key value is compared with the middle element of the list
- If the values are equal then the search is successful and the process is stopped
- If the middle values is less, then, the result is in upper half of the list
- If the middle value is greater ,then ,the result is in lower half of the list
- The search is repeated for the lower or upper half of the list until we find the required value in the list or all items from the list is searched

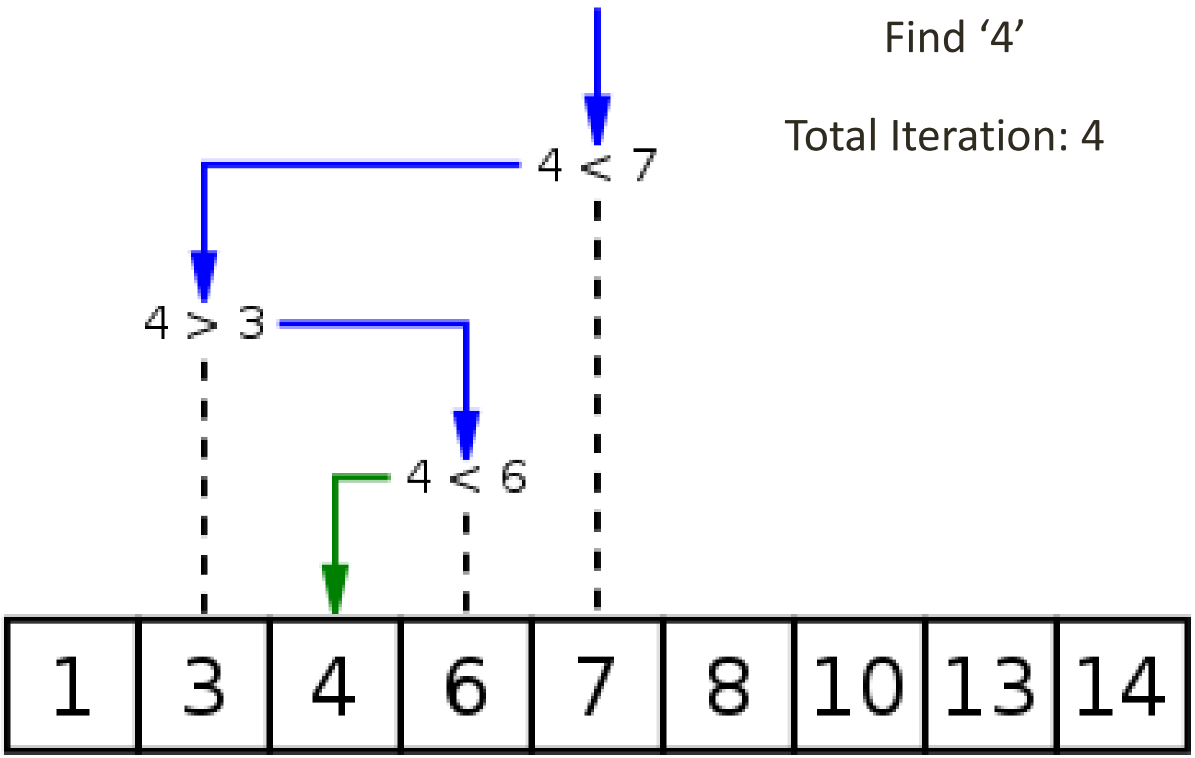
Binary Search (Example)



Find ‘x’

Total Iteration: 4

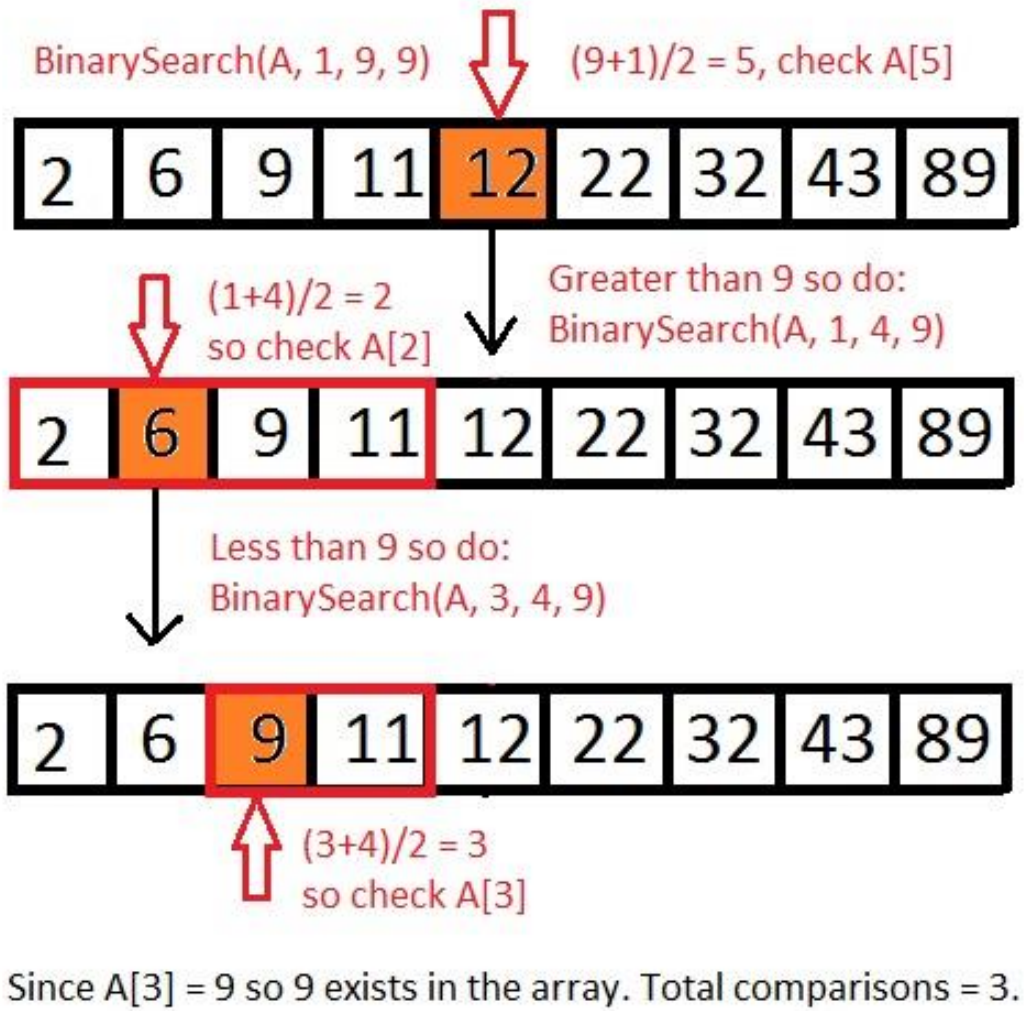
Binary Search (Example)



Find ‘4’

Total Iteration: 4

Binary Search (Example)



Binary Search (Algorithm)

```
Declare and initialize necessary variables
   n; a[n]; first=0; last=n-1;item;flag=0;middle=(first + last)/2
While (first<=last)
   if (a[middle]=item)
          display "Search Successful"
          flag=1;
          stop
   else if (item<a[middle])
          last=middle-1
   else
          first=middle+1
   end if
   middle=(first + last)/2
End while
If (flag=0)
   display "Search unsuccessful"
End if
```

Binary Search (Algorithm)

```
BINARY_SEARCH(A, lower_bound, upper_bound, VAL)
Step 1: [INITIALIZE] SET BEG = lower_bound
         END = upper_bound, POS = - 1
Step 2: Repeat Steps 3 and 4 while BEG <= END
Step 3:           SET MID = (BEG + END)/2
Step 4:           IF A[MID] = VAL
                        SET POS = MID
                        PRINT POS
                        Go to Step 6
                  ELSE IF A[MID] > VAL
                        SET END = MID - 1
                  ELSE
                        SET BEG = MID + 1
                  [END OF IF]
        [END OF LOOP]
Step 5: IF POS = -1
            PRINT "VALUE IS NOT PRESENT IN THE ARRAY"
        [END OF IF]
Step 6: EXIT
```

Algorithm for binary search

Binary Search

- Binary search is best suited if
  - the data are present in sorted form and
  - are being represented in array or list
- The main drawbacks are:
  - Requires the data to be already sorted
  - Cannot be used where there are many insertions or deletions

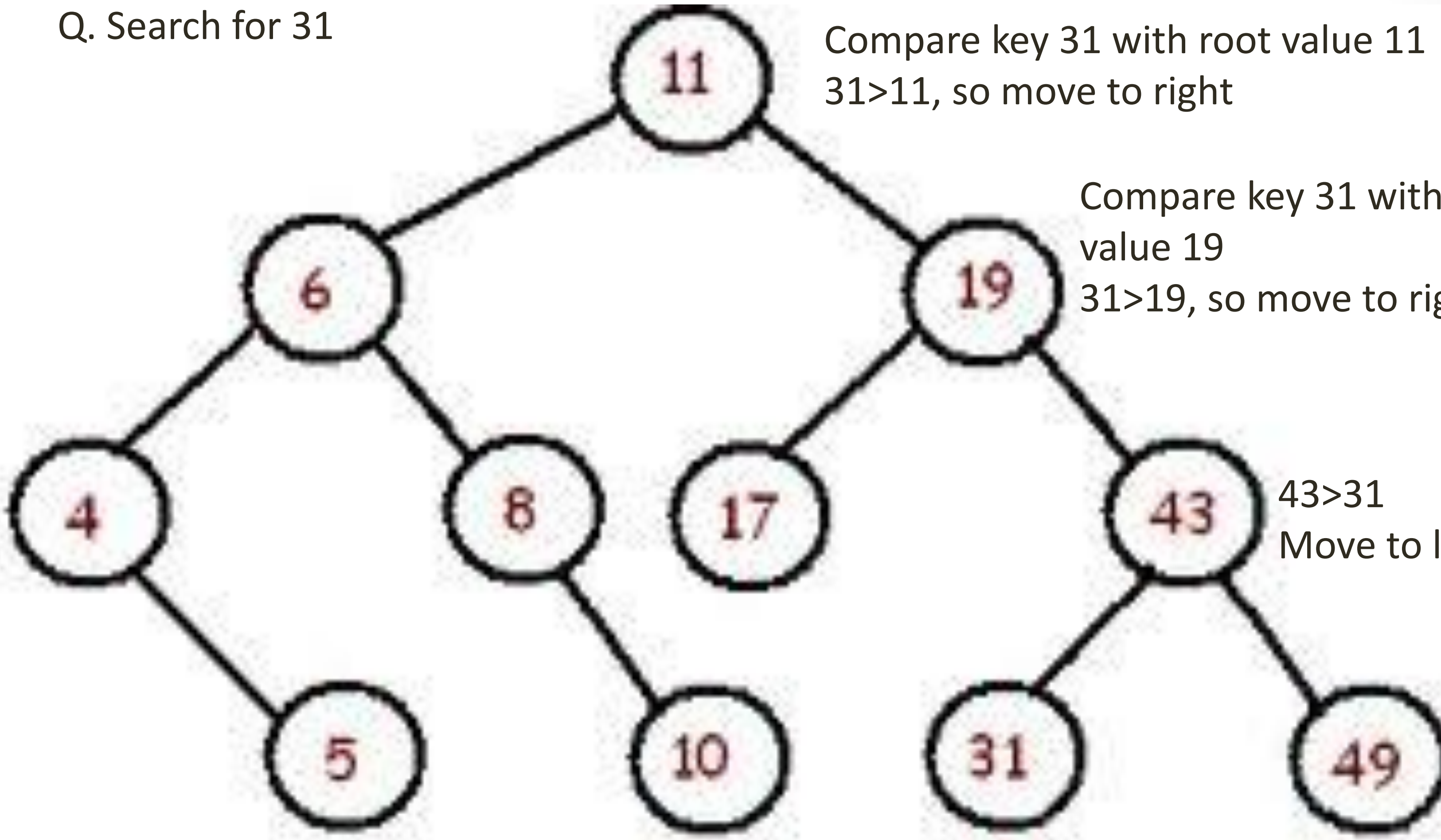
Tree Search

- If the data are arranged in the form of search tree structure, then the tree search can be applied
- Search tree generally is a Binary Search Tree
- Here the key value is compared with the root node at first.
- If it matches then the search is successful
- If it doesn’t matches then either the left or right sub-tree is searched based upon the comparison result
    - If the data item is less than the root node, then left subtree is searched
    - If the data item is greater than the root node, then right subtree is searched
- The traversal is repeated until the searched item is found or null value is reached

Tree Search (Example)

Q. Search for 31

Compare key 31 with root value 11
31>11, so move to right



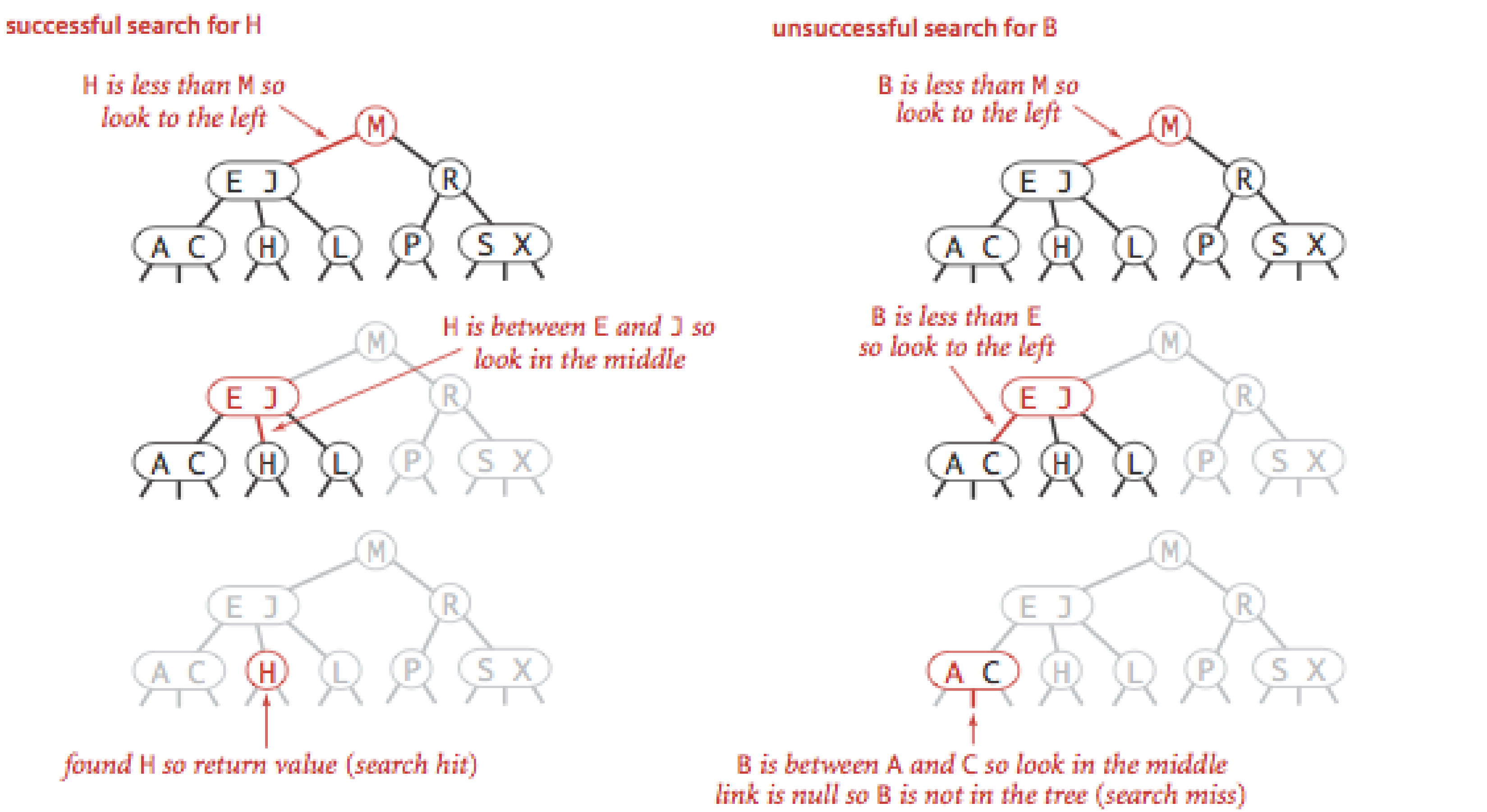
Compare key 31 with root value 19
31>19, so move to right

43>31
Move to left

Found 31
Search Successful

Tree Search (Example)

- Tree Search can be implemented in m-way tree as well



Tree Search (Algorithm)

```
Declare and initialize necessary variables
    node=root – node pointer pointing to root of the tree
    item – data to be searched; flag=0
If(node=NULL)
    display "Empty Tree"; Stop;
End if
While (node!=NULL)
    if (item=node->data)
            display "Search Successful"
            flag=1;
            stop;
    else if (item>node -> data)
            node=node -> right;
    else
            node=node ->left;
    end if
End while
If (flag=0)
    display "Search Unsuccessful"
End if
```

Hashing

- **Hashing** is the technique of representing longer records by shorter values called **keys**
- The keys are placed in a table called **hash table** where the keys are compared for finding the records
- **Hash table** is a dictionary in which keys are mapped to array positions by using suitable mathematical function called **hash function**.
- One of the simple search scheme where the records are indexed using certain hash function is called hashing
- Items being searched can directly be accessed by using the hash table by mapping the corresponding key values into records
- Hashing technique requires minimum (generally 1) number of comparison for searching the desired record

Hashing

- Lets consider an example:
  - Numbers from 1 to 99 can be indexed in a hash table by using the hash function of modulo 10 division
  - This function results with the last digit of the numbers which can be used as key for the hash table
  - If random numbers are chosen, all the indices may not be full
  - Some index may contain more than one values and some may not contain any
  - This imbalance in indices is called **Clustering**
  - More than one data in a single index is called **Collision**, which results with the conflict while searching

Key	Value
0	10, 20, 40
1	
2	42, 82
3	
4	64
5	95
6	
7	87, 47
8	
9	99

Hashing (terminology)

- Home Address:
  - Address produced by the hash function
- Prime area:
  - The memory location that contains all the home addresses
- Synonyms:
  - A set of keys that hash to the same locations
- Collision:
  - The location of data to be inserted is already occupied by the synonym data
- Probing:
  - Examining memory location in the hash table.

Hash Function (Types)

- Mathematical formula which when applied to a key, produces an integer which can be used as index for the key in hash table.
- Its main aim is to distribute elements uniformly to minimize the number of collision.
- Any given hashing technique can be considered ideal if,
  - There is no location collision (at least minimal)
  - The address space in memory is compact
- Different types of hash functions can be used
- Some of the popular ones are:
  - Direct hashing
  - Modulo Division
  - Multiplicative
  - Digit Extraction (truncation)
  - Mid-Square
  - Folding

Direct Hashing

- The address is the key itself
  - Hash(key)=key
- The main advantage is that there is not any collision
- The disadvantage is that the address space (storage) is as large as the key space

address	Key
0	0
1	1
---	---
---	---
50	50
51	51
---	---
---	---
1089	1089
1090	1090

Modulo Division

- Uses remainder obtained as address
- ***Hash(key) = address = key % listsize***
- Yields hash value which belongs to the set {0,1,2,3,…..,listsize}
- Fewer collisions if listsize is a prime number
- Example :
  - Numbering system to handle 1,500 students
  - If key is 12865, Address=hash(12865)=12865 % 1500=865
  - If key is 224568, Address=hash(224568)= 224568 % 1500= 1068

Multiplicative Method

- Steps in this method:
  - 1. Choose constant **c** , a real number, between 0 and 1
  - 2. Multiply key **k** by **c , [k*c]**
  - 3. Get fractional part of the product, **[k*c-floor(k*c)]**
    - This is a random number between 0 and 1
  - 4. Multiply the result by listsize and obtain the integer part, **floor(listsize *[k*c-floor(k*c)]) or floor(listsize *[k*c mod 1])**
  - 5. The final result is the required address

- ***Address=hash(k)=floor(listsize*(k*c-floor(k*c))***
  - Where 0<c<1
  - Note: floor(X) is the largest integer not greater than X
  - Knuth has suggested that best choice of c is 0.6180339887

Multiplicative Method

Example 1:

Assume;

k=12876

Listsize=100

C=0.12

Now the address is:

Address=hash(12876)

=Floor(100*(12876*0.12-floor(12876*0.12)))

=floor(100*(1545.12-floor(1545.12)))

=floor(100*(1545.12-1545))

=floor(100*0.12)

=12

Example 2:

Assume;

k=12345

Listsize=1000

C=0.618033

Now the address is:

Address=hash(12345)

=Floor(1000*(12345*0.618033 mod 1))

=floor(1000*(7629.617385 mod 1))

=floor(1000*(.617385))

=floor(617.385)

=617

Digit Extraction

- Some digits from the number in specific places are extracted
- The places from which the extraction has to be done are predefined
- The same extraction technique is used for all the keys
- Here,
  - Address=selected digit from the key

Example:
If places for digit extraction are – 10s , 100s and 100000s
345261=326
167524=152
543625=562
987709=970

Mid Square

- Few number of middle digits from the key are extracted
- Thus extracted number is squared
- The squared result is the required address value
- The number of digits chosen depends on number of digits allowed for indexing

Example:
Assume;
k=12876
Extract second and third digit
N=28
Now,
Address=hash(12876)=N*N
=28*28
=784

Mid Square

- The major disadvantage is value obtained by doing square may be too large
- The resolution can be to use only a portion of the result
  - Few number of digits from the middle of the result is used

Example:
K=39873
Address=98*98=9604
Which is long
Hence use only portion of the result
New Address=60

Folding

- Step 1 : The key is divided into number of parts having same number of digits (or maybe less for the last part)
- Step 2 : Sum all the individual divided parts
- Step 3: If there is any carry in the result, then discard it
- Step 4: Thus formed number is the address for the key

Example:
Assume;
k=12896543
Hash table size = 000 to 999 (i.e. 3 digits)
Our part division will be: 128+965+43
=1136
Truncate the carry (i.e. 1 in thousand's place)
Hence our address will be,
Address=136

Collision Resolution

- Direct hashing maps the key values with the individual addresses, hence it is a one-to-one mapping technique and no collision occurs.
- All other hashing techniques may results with some collision
- Different collision resolution techniques are used
- These techniques are independent of the hashing functions applied
- All these techniques target to minimize clustering because clustering is the main reason for collision

Collision Resolution Techniques

- Two basic techniques are used:
  - 1. Rehashing (Also called Open Addressing)
    - The types are:
      - Linear Probing
      - Quadratic Probing
      - Double Hashing
  - 2. Chaining

Collision Resolution

- Open Addressing:
  - When collision occurs, an unoccupied address is searched for placing the new element using *probe* sequence
  - Rehashing **rh** is applied to address value **h(key)** if the **h(key)** is already occupied in the hash table.
  - Again if **rh(h(key))** is already occupied we apply **rh(rh(h(key)))** until an open address is found
  - It can be done in 3 different ways:
    - Linear Probing
    - Quadratic Probing
    - Double Hashing

Linear probing

- When a home address is occupied, go to the next address
- Next address = current address + 1
  - Rh(k,i) = (h(k)+i) % listsize
    - Where h(k) = k % listsize
    - and i=0, 1, 2, 3, …………, listsize-1

Linear Probing

	Insert 18, 89, 21	Insert 58, 68	Insert 11
0		*t* 58	*t* 58
1	*t* 21	*t* 21	*t* 21
2		*t* 68	*t* 68
3			*t* 11
4			
5			
6			
7			
8	*t* 18	*t* 18	*t* 18
9	*t* 89	*t* 89	*t* 89

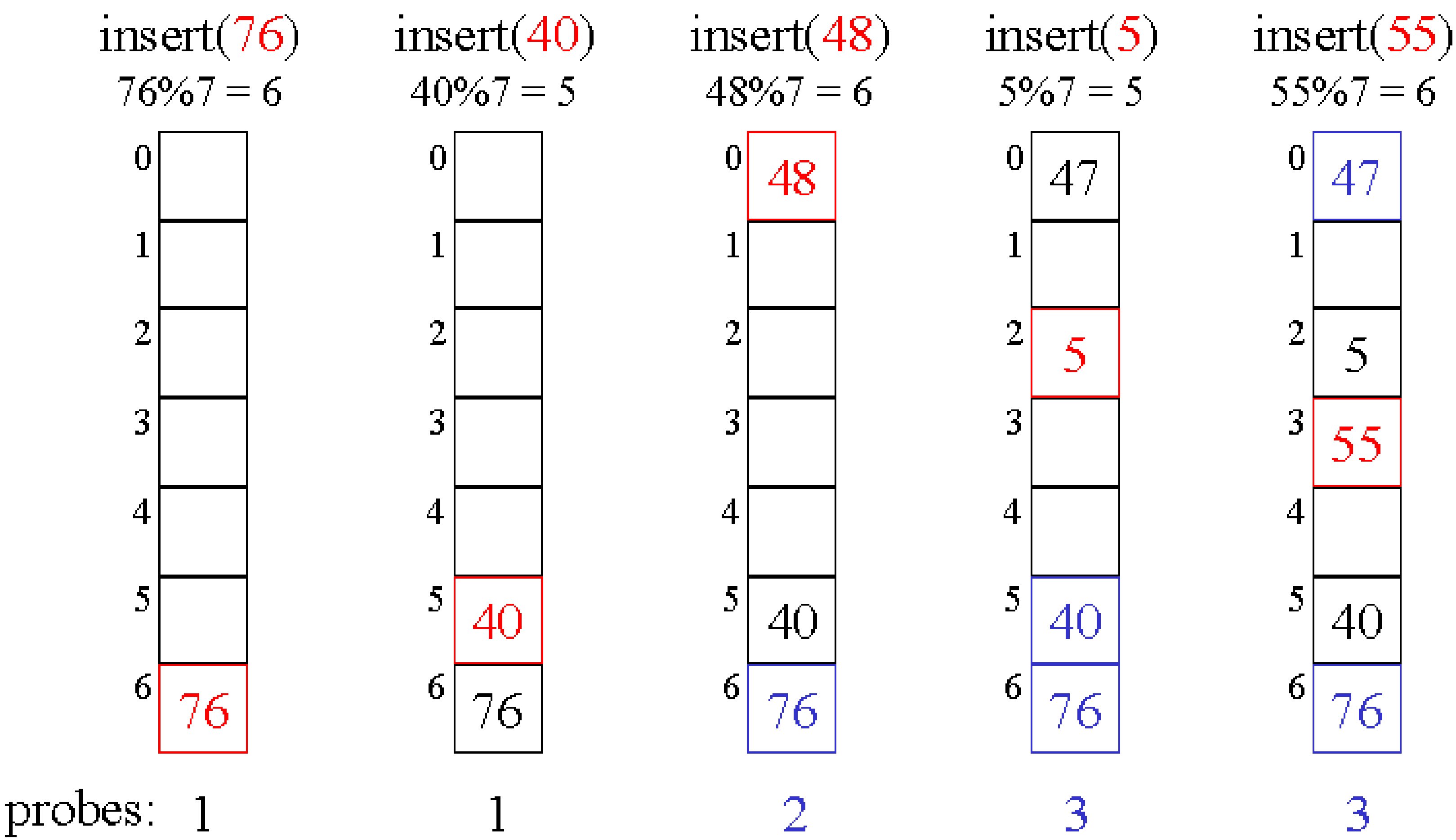
Linear Probing

- Advantages:
  - Simple to implement
  - Data tend to cluster around home address resulting to compactness of disk spaces
- Disadvantages:
  - Data tend to cluster around specific home address (Primary Clustering)
  - The linear searching is required if data is not present in the searched location, this is very slow process

Quadratic Probing

- Tends to minimize the problem of primary clustering from linear probing
- The value is moved considerable distance from the initial collision
- The address incremented is the collision probe number squared, i.e.
  - rh(k,i) = (h(k) + i^2) % listsize
    - Where h(k) = k % listsize
    - and i=0, 1, 2, 3, ………, listsize-1

Quadratic Probing



Quadratic Probing

- Advantages:
  - Works much better than linear probing
  - Removes primary clustering
- Disadvantages:
  - Time consuming than linear probing
  - Produces secondary clustering

Double Hashing

- Two different hash functions are used to generate the address if the initial hashing results with collision
- This removes the secondary collision
- The initial hash value is reused to rehash functions and new hash value is computed
  - $hp(k, i) = (h_1(k) + i*h_2(k))$ % listsize
    - Where $h_1(k) = k$ % listsize
    - and h2(k) = k % (some integer slightly less than listsize)
      - I = 0, 1, 2, 3, ………, (listsize-1)

Double Hashing

76, 93, 40, 47, 10, 55, 73, 56

$h_1(k)=k\%10$
$hp(k,i)=(h_1(k) + i * h_2(k)) \% listsize$
Where i = 0, 1, 2, 3, ………, listsize-1
$h_2(k) = k \% (listsize-1)$

	76	93	40	47	10	55	73	56
0			40	40	40	40	40	40
1					10	10	10	10
2								
3		93	93	93	93	93	93	93
4							73	73
5						55	55	55
6	76	76	76	76	76	76	76	76
7				47	47	47	47	47
8								56
9								

Open Addressing (Disadvantage)

- Major disadvantages are:
  - Each collision resolution results with the probability for future collision
  - If the number keys are more than the address size of hash table, then collision is sure to occur.
    - This is called overflow
  - To overcome these disadvantages, separate chaining is used.

Chaining

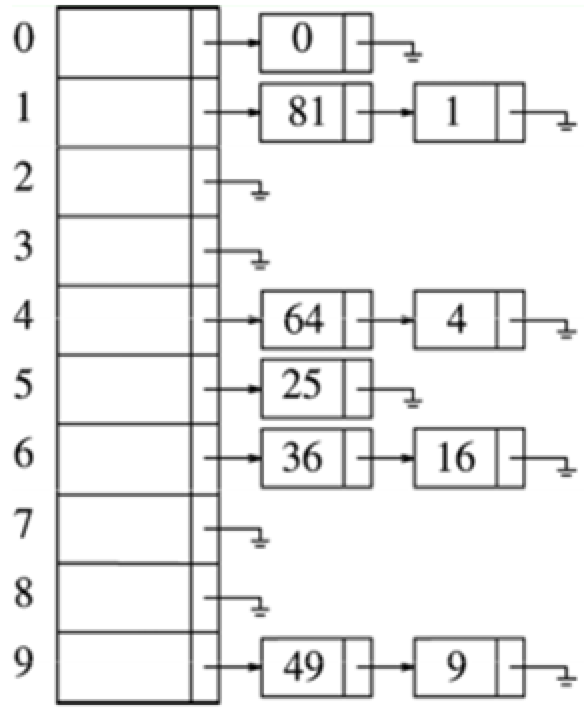
- Also called separate chaining
- Use fixed size hash table
- Link lists are used to store the synonyms
- Each slot in hash table points to the head of the linked list
- All the elements for that address is placed in linked list

- Chaining strategy: maintains a linked list at every hash index for collided items
- Hash table T is a vector of linked list
  - Insert element at the head or at the tail
- Key k is stored in list at T[h(k)]

Chaining

- E.g. Tablesize=10
  - H(k) = k mod 10
  - Insert first 10 perfect squares
  - Insertion sequence:

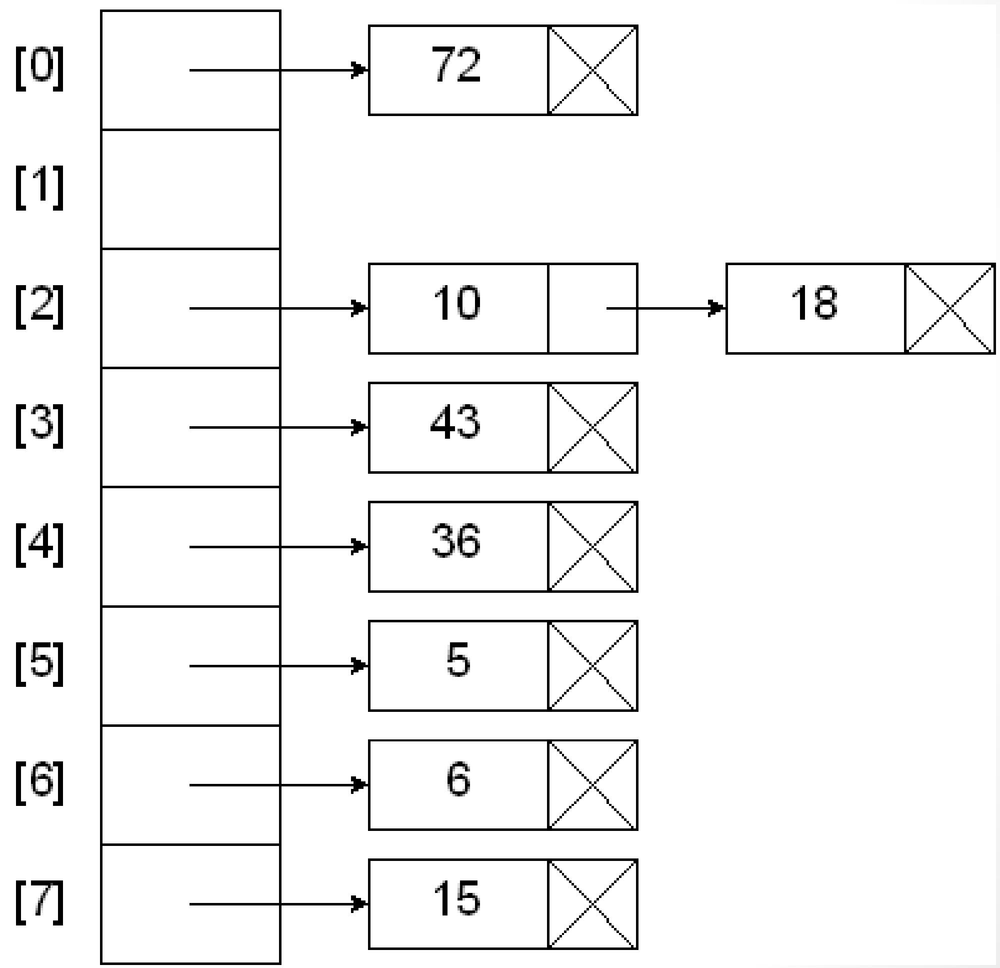
{0,1,4,9,16,25,36,49,64,81}



Chaining

Hash key = key % table size

4	=	36	%	8	
2	=	18	%	8	
0	=	72	%	8	
3	=	43	%	8	
6	=	6	%	8	
2	=	10	%	8	
5	=	5	%	8	
7	=	15	%	8	



THANK YOU!