Introduction To Searching

By Yash Gupta

Linear Search

```
Algorithm sequentialsearch(int list[] , int end , int target )
   i= 0
   while( i < end && list[i] != target )
        i++
   if( i  == end )
      return -1
   else
      return i
```

Time Complexity

Best Case	Average Case	Worst Case
$T(n) = c$ $= O(1)$	$T(n) = n/2$ $= O(n)$	$T(n) = n$ $= O(n)$

Binary Search

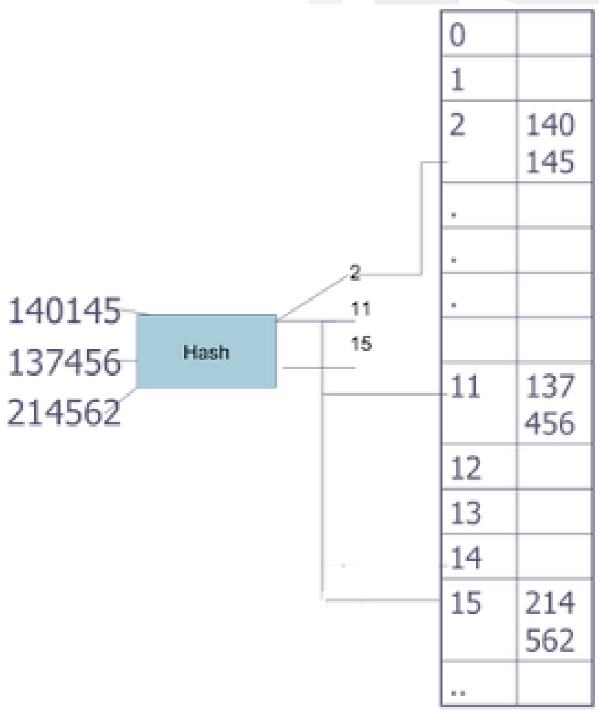
```
Algorithm binarysearch(int list[] , int left , int right , int target )
    while ( left <= right )
        mid = (left + right) / 2
        if( list[ mid ] == target )
            return mid
      else if( list[mid] < target )
             left = mid + 1
        else
            right = mid - 1
    return -1
```

Time Complexity

Best Case	Average Case	Worst Case
$T(n) = c$ $= O(1)$	$T(n) = \log n$ $= O(\log n)$	$T(n) = \log n$ $= O(n)$

Hashed Search

- Hashing is a key-to-address mapping process.



Hashing Methods

- Direct
- Subtraction
- Modulo-division: address = key % listsize (listsize must be chosen a prime no)
- Digit-Extraction
- Mid-square
- Folding : Fold Shift and Fold Boundary
- Rotation
- PseudoRandom

Collision

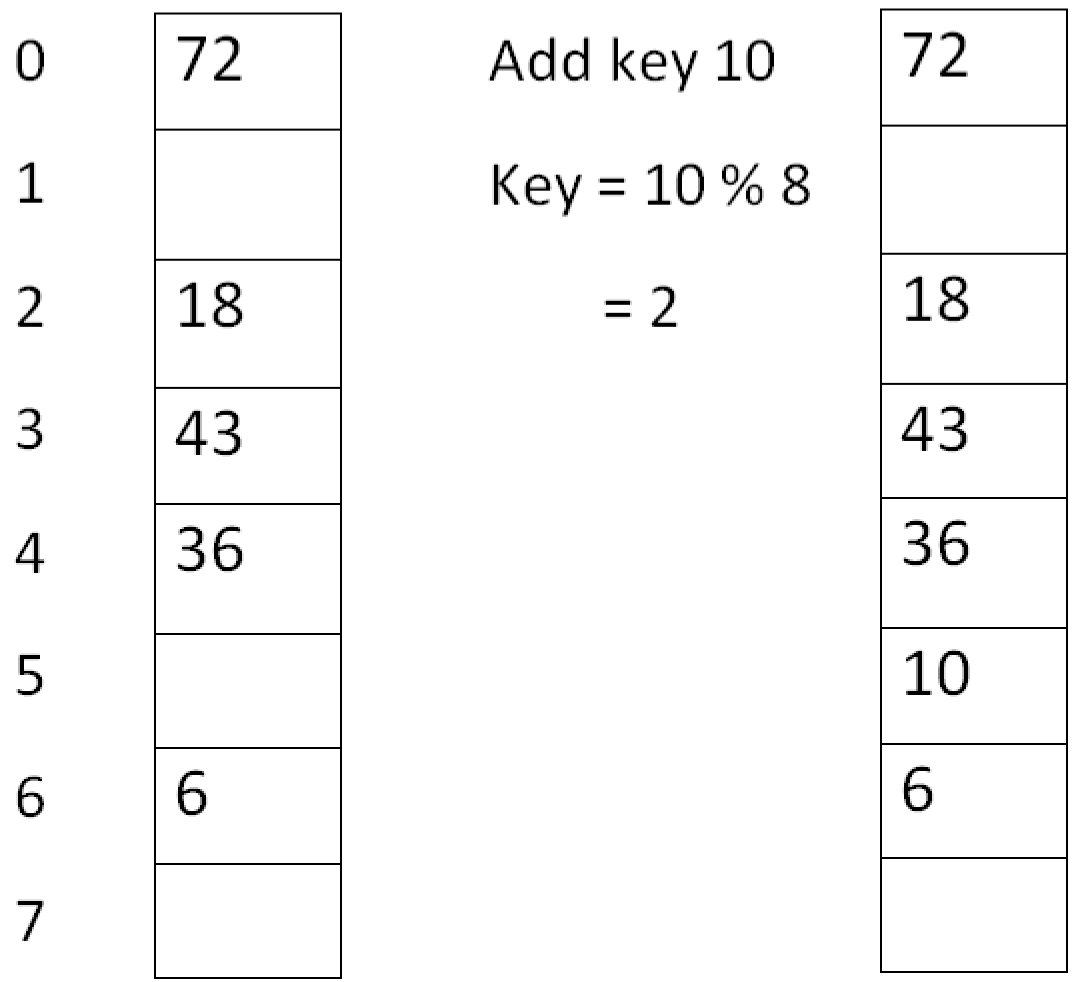
- Keys collide to same home address
- Solution
  - Allocate new address in Prime Area
  - Allocate new address in Overflow Area

Collision Resolution

- Open Addressing
  - Linear Probe
  - Quadratic Probe
- Link List
- Bucket

Linear Probe

- Address = (Key + Probe) % Size


0
1
2
3
4
5
6
7
72
18
43
36
6
Add key 10
Key = 10 % 8
= 2
72
18
43
36
10
6

0	72		72
1			
2	18		18
3	43		43
4	36		36
5	10		10
6	6		6
7			5

Add key 5

Key = 5 % 8

= 5

0	72	72
1		15
2	18	18
3	43	43
4	36	36
5	10	10
6	6	6
7	5	5

Add key 15

Key = 15 % 8

= 7

Quadratic Probe

- Address = (Key + Probe^2) % Size

Add key 18

Probe 0 :

= (18+0) % 10

= 8

Add Key 89

= (89 + 0) % 10

= 9

Add key 21

= (21 + 0) % 10

= 1

0		
1		21
2		
3		
4		
5		
6		
7		
8		18
9		89

0	
1	21
2	
3	
4	
5	
6	
7	
8	18
9	89

Add key 58

Probe 0 :

= (58+0) % 10

= 8

Probe 1 :

= (58 + 1) % 10

= 9

Probe 2 :

= (58 + 4) % 10

= 2

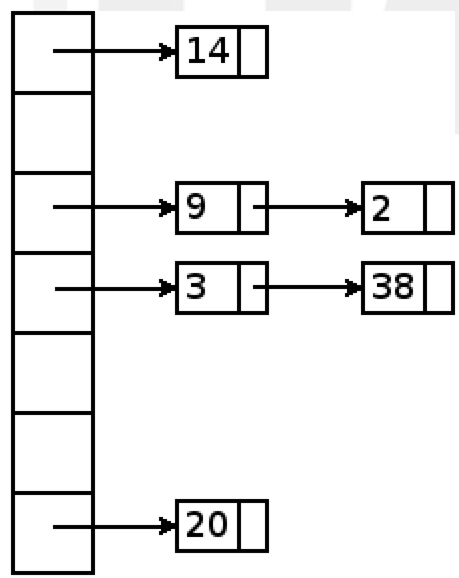
21
58
18
89

Time Complexity

Best Case	Average Case	Worst Case
$T(n) = c$ $= O(1)$	$T(n) = c$ $= O(1)$	$T(n) = n$ $= O(n)$

Link List

- Chain the keys that collide at same location
- The collided Keys occupy overflow area instead of prime area



Time Complexity

Best Case	Average Case	Worst Case
$T(n) = c$ $= O(1)$	$T(n) = c$ $= O(1)$	$T(n) = c$ $= O(1)$

Contact Info

- trainers@finaldesk.com
- rishabh@finaldesk.com
- nilesh@finaldesk.com
- jignesh@finaldesk.com
- yash@finaldesk.com
- anand@finaldesk.com