Unit 8: Searching

1. Introduction
2. Sequential search
3. Binary search
4. Comparison and efficiency of searching
5. Hashing-probing (Linear and Quadratic)

What is Searching?

Searching is a process of finding an element within the list of elements stored in any order or randomly. Searching is divided into two categories **linear (sequential)** and **binary** search.

Sequential Search

In linear search, access each element of an array one by one sequentially and see whether it is desired element or not. A search will be unsuccessful if all the elements are accessed and the desired element is not found.

In brief, Simply search for the given element left to right and return the index of the element, if found. Otherwise return “Not Found”.

Algorithm:

```
LinearSearch(A, n,key)
{
        for(i=0;i<n;i++)
        {
                if(A[i] == key)
                return i;
        }
        return -1;//-1 indicates unsuccessful search
}
```

Analysis:
Time complexity = O(n)

Binary Search:

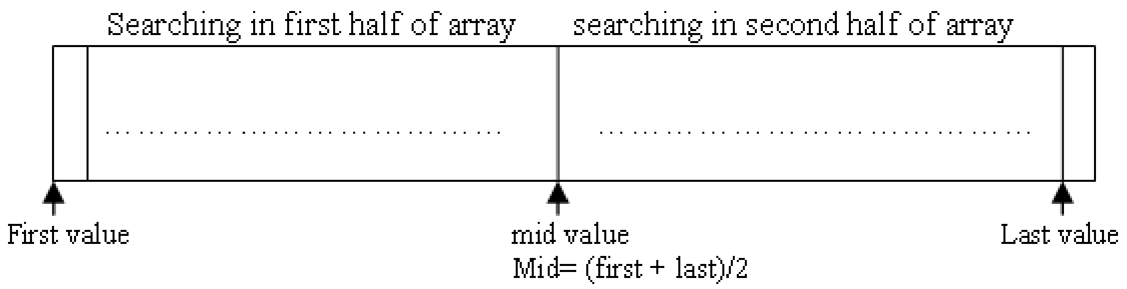
Binary search is an extremely efficient algorithm. This search technique searches the given item in minimum possible comparisons. To do this binary search, first we need to sort the array elements. The logic behind this technique is given below:

1. First find the middle element of the array
2. compare the middle element with an item.
3. There are three cases:

i) If it is a desired element then search is successful
ii) If it is less than desired item then search only the first half of the array.
iii) If it is greater than the desired element, search in the second half of the array.

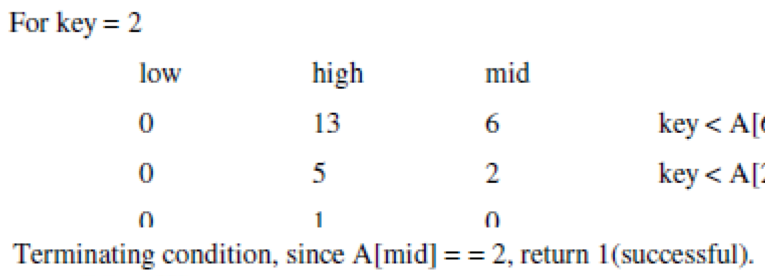
4. Repeat the same process until element is found or exhausts in the search area.

In this algorithm every time we are reducing the search area.


Searching in first half of array
searching in second half of array
First value
mid value
Mid= (first + last)/2
Last value

Running example:

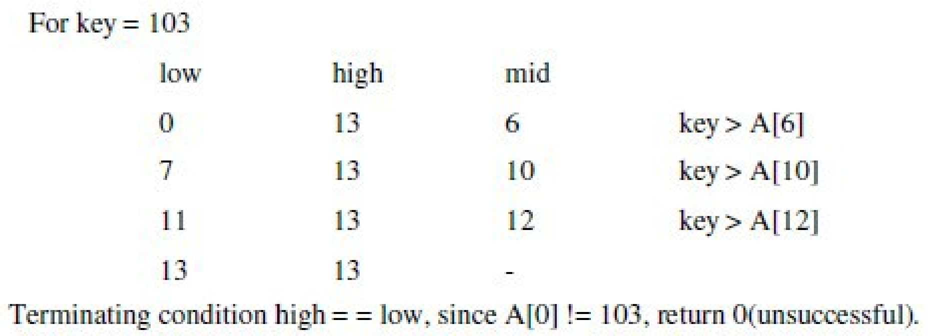
Take input array a[14] = {2 , 5 , 7, 9 ,18, 45 ,53, 59, 67, 72, 88, 95, 101, 104}



For key = 2

low	high	mid	
0	13	6	key < A[6]
0	5	2	key < A[2]
0	1	0	

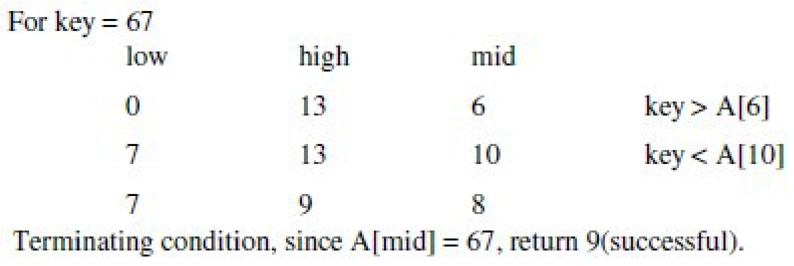
Terminating condition, since A[mid] = = 2, return 1(successful).



For key = 103

low	high	mid	
0	13	6	key > A[6]
7	13	10	key > A[10]
11	13	12	key > A[12]
13	13	-	

Terminating condition high = = low, since A[0] != 103, return 0(unsuccessful).



For key = 67

low	high	mid	
0	13	6	key > A[6]
7	13	10	key < A[10]
7	9	8	

Terminating condition, since A[mid] = 67, return 9(successful).

Algorithm:

```
BinarySearch(A,low,high,key)
{
    if(low == high) //only one element
    {
        if(key = = A[low])
            return low+1; //index starts from 0
        else
            return 0;
    }
     else
    {
        m = (low + high) /2 ; //integer division
        if(key = = A[m]
            return m+1;
        else if (key < A[m])
            return BinarySearch(A, low, m-1, key) ;
        else
            return BinarySearch(A, m+1, high, key) ;
    }
}
```

Efficiency:

From the above algorithm we can say that the running time of the algorithm is
$T(n) = T(n/2) + O(1)$
$= O(\log n)$ (verify).
In the best case output is obtained at one run i.e. O(1) time if the key is at middle.
In the worst case the output is at the end of the array so running time is O(logn) time.ith
In the average case also running time is O(logn).
For unsuccessful search best, worst and average time complexity is O(logn).

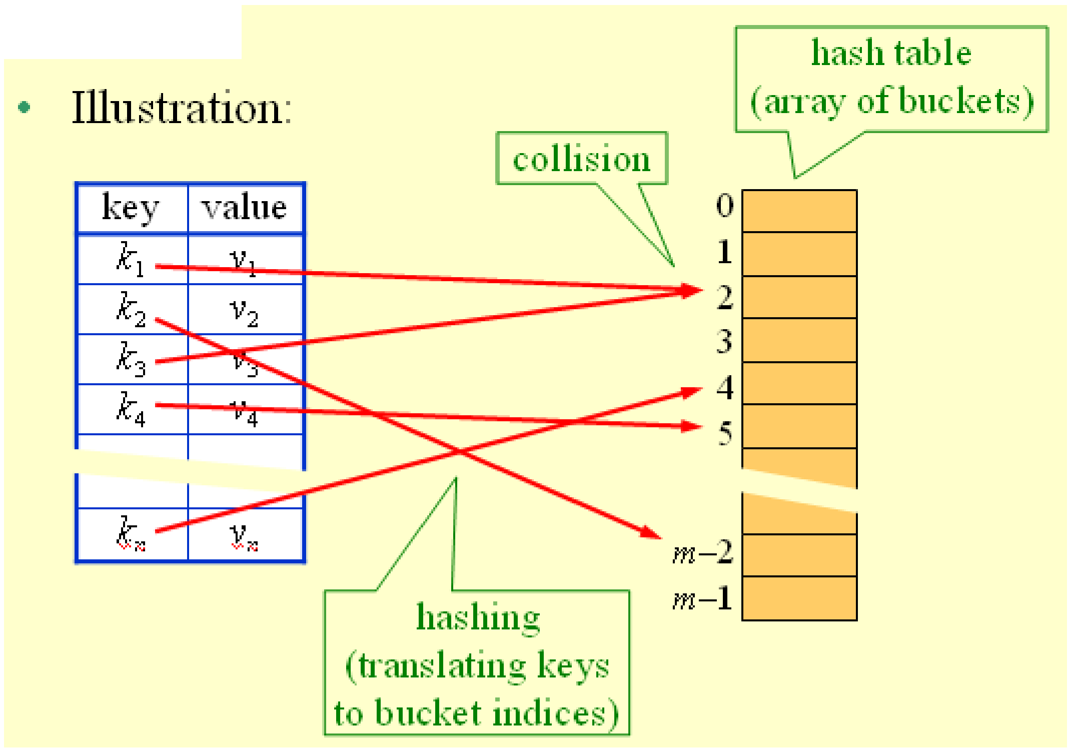
Hashing

It is an efficient searching technique in which key is placed in direct accessible address for rapid search. Hashing provides the direct access of records from the file no matter where the record is in the file. Due to which it reduces the unnecessary comparisons. This technique uses a hashing function say h which maps the key with the corresponding key address or location.

A function that transforms a key into a table index is called a hash function.
A common hash function is
h(x)=x mod SIZE

if key=27 and SIZE=10 then
hash address=27%10=7

Hash-table principles



Hash collision:

If two or more than two records trying to insert in a single index of a hash table then such a situation is called hash collision.

Some popular methods for minimizing collision are:

1. Linear probing
2. Quadratic probing
3. Rehashing
4. Chaining
5. Hashing using buckets etc

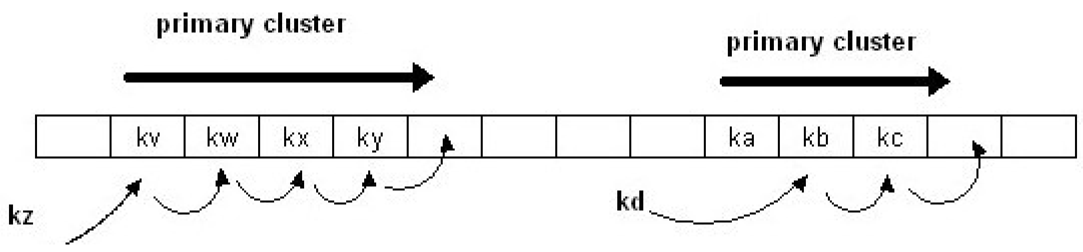
But here we need only first two methods for minimizing collision

Linear probing:

A hash-table in which a collision is resolved by putting the item in the next empty place within the occupied array space. It starts with a location where the collision occurred and does a sequential search through a hash table for the desired empty location. Hence this method searches in straight line, and it is therefore called linear probing.

Disadvantage:

Clustering problem



Example:
Insert keys {89, 18, 49, 58, 69} with the hash function h(x)=x mod 10 using linear probing.
solution:

when x=89:
h(89)=89%10=9
insert key 89 in hash-table in location 9
when x=18:
h(18)=18%10=8
insert key 18 in hash-table in location 8
when x=49:
h(49)=49%10=9 (Collision occur)
so insert key 49 in hash-table in next possible vacant location of 9 is 0
when x=58:
h(58)=58%10=8 (Collision occur)
insert key 58 in hash-table in next possible vacant location of 8 is 1
(since 9, 0 already contains values).
when x=69:
h(89)=69%10=9 (Collision occur)
insert key 69 in hash-table in next possible vacant location of 9 is 2
(since 0, 1 already contains values).

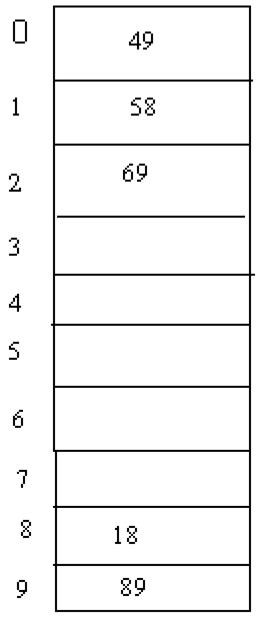


Fig Hash-table for above keys using linear probing

Quadratic Probing:

Quadratic probing is a collision resolution method that eliminates the primary clustering problem take place in a linear probing. When collision occur then the quadratic probing works as follows:

(Hash value + 1^2)% table size
if there is again collision occur then there exist rehashing.
(hash value + 2^2)%table size
if there is again collision occur then there exist rehashing.
(hash value = 3^2)% table size

in general in ith collision
$h^i(x)$=(hash value +i^2)%size

Example:
Insert keys {89, 18, 49, 58, 69} with the hash-table size 10 using quadratic probing.
solution:
when x=89:
h(89)=89%10=9
insert key 89 in hash-table in location 9
when x=18:
h(18)=18%10=8
insert key 18 in hash-table in location 8
when x=49:
h(49)=49%10=9 (Collision occur)
so use following hash function,
h1(49)=(49 + 1)%10=0
hence insert key 49 in hash-table in location 0
when x=58:
h(58)=58%10=8 (Collision occur)
so use following hash function,
h1(58)=(58 + 1)%10=9
again collision occur use again the following hash function ,
h2(58)=(58+ 2^2)%10=2
insert key 58 in hash-table in location 2
when x=69:
h(89)=69%10=9 (Collision occur)
so use following hash function,
h1(69)=(69 + 1)%10=0
again collision occur use again the following hash function ,
h2(69)=(69+ 2^2)%10=3
insert key 69 in hash-table in location 3

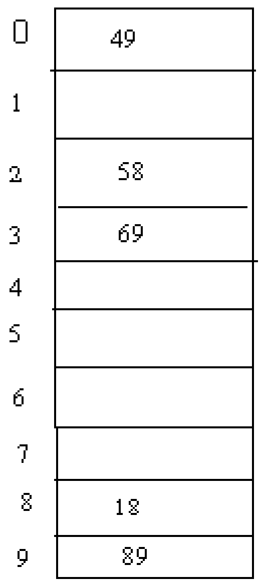


Fig:Hash table for above keys using quadratic probing