Unit - III

Searching :→ It is the process of finding a given value position in a list of values. It decides whether a search key is present or not.

- Searching Techniques
  - Sequential Search OR Linear Search
  - Interval Search OR Binary Search

Linear Search :→ It simply traverse the list completely and matth each element of the list with the item whose location is to be found. If the match is found, then the location of the item is returned, otherwise the algorithm returns Null.

Ex:-

0	1	2	3	4
30	20	11	40	70

Total element (n) = 5

Let the element to be searched is K = 40.

⊙ Now, start from the first element & Compare K with each other element of array.

0	1	2	3	4
30	20	11	40	70

↑ K ≠ 30

Not matched, then move to another element of array

⊙

0	1	2	3	4
30	20	11	40	70

↑ K ≠ 20

Not matched, move to another element.

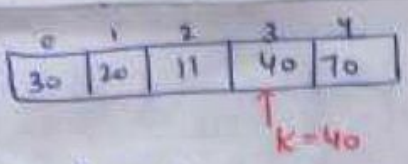
⊙

0	1	2	3	4
30	20	11	40	70

↑ K ≠ 11

Move to next element.

⊙



Element found, return the index of the element i.e. 3

[n=5, k=40]

```
*   for (i=0; i<n; i++)
    {
        if (a[i] == K)
        {
            printf("element found at %d", i);
            break;
        }
    }
    if (i == n)
    {
        printf("Element not found");
    }
```

Advantages :-

1) No need of sorting.
2) does not affected by insertions & deletions.

Disadvantages :-

1) Time Consuming.
2) less efficient.

Binary Search :- It works on sorted list. It follows divide & conquer approach in which the list is divided into two halves, and then item is compared with the middle element of the list. If the match is found, then return the location of middle element. Otherwise, we search into either of the halves depending upon the result produced through the match.

There are three cases:-

- a[mid] == data ⇒ return a[mid];
- a[mid] > data ⇒ r = mid - 1;
- a[mid] < data ⇒ l = mid + 1;

Eg:-

0	1	2	3	4	5	6
10	12	24	40	56	60	69

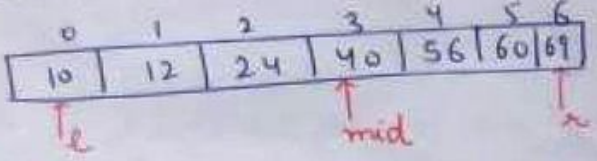
k = 56, n = 7

let the element to search is k = 56.

- Calculate

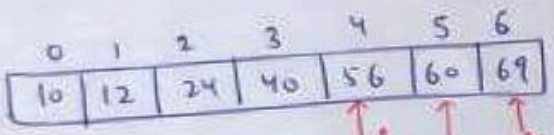
$$mid = \left\lfloor \frac{l+r}{2} \right\rfloor = \frac{0+6}{2} = 3$$

l	r	mid
0	6	3



a[mid] < k [40 < 56]

l = mid + 1 = 3 + 1 = 4

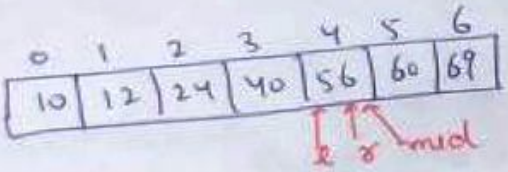
- 

$mid = \frac{4+6}{10} = 5$

l	r	mid
0	6	3
4	6	5

a[mid] > k [60 > 56]

r = mid - 1 = 5 - 1 = 4

- 

$mid = \frac{4+4}{2} = \frac{8}{2} = 4$

l	r	mid
0	6	3
4	6	5
4	4	4

a[mid] = k

return a[mid] i.e. index i.e. 4

element is found at 4.

```
int binarySearch(int a[], int l, int r, int k)
{
    int mid;
    while (l < r)
    {
        mid = (l+r)/2;
        if (a[mid] == k)
        {
            return mid;
        }
        else if (a[mid] < k)
        {
            return binarySearch(a, mid+1, r, k);
        }
        else
        {
            return binarySearch(a, l, mid-1, k);
        }
    }
    return -1;
}
```

Advantages :-

1) It reduces the search space.
2) For large data, it works efficiently.

Disadvantages :-

1) Error prone.
2) difficult
3) It requires more stack space.

Index Sequential Search :-

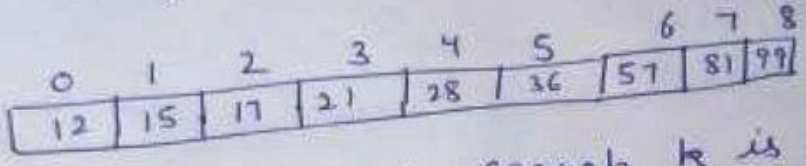
It is based on sequential & random access searching method. It search the elements according to groups.

Algo:→

1) Read the search element from the user.
2) Divide the array into groups acc. to group size.
3) Create index array that contains starting index of groups.
4) If group is present and first element of that group is less than or equal to key element go to next group

else apply linear search in previous group.

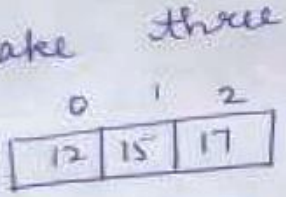
Eg:-

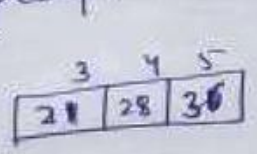


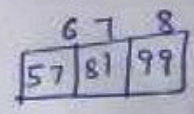
n = 9

Let the element to search k is 28 and group size = 3

⊙ Make three groups acc. to group size.







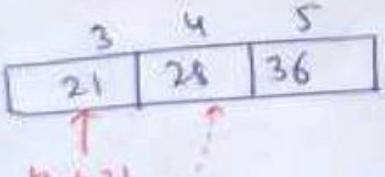
⊙ Compare first element of every group with k.

→ In Ist group, 12 < k i.e. 12 < 28, move to next group.

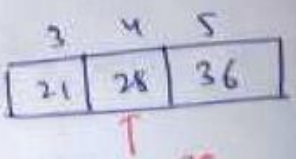
→ In IInd group, 21 < 28, move to next group.

→ In IIIrd group, 57 > 28, do linear search in previous group.

⊙



⊙



element found at 4.

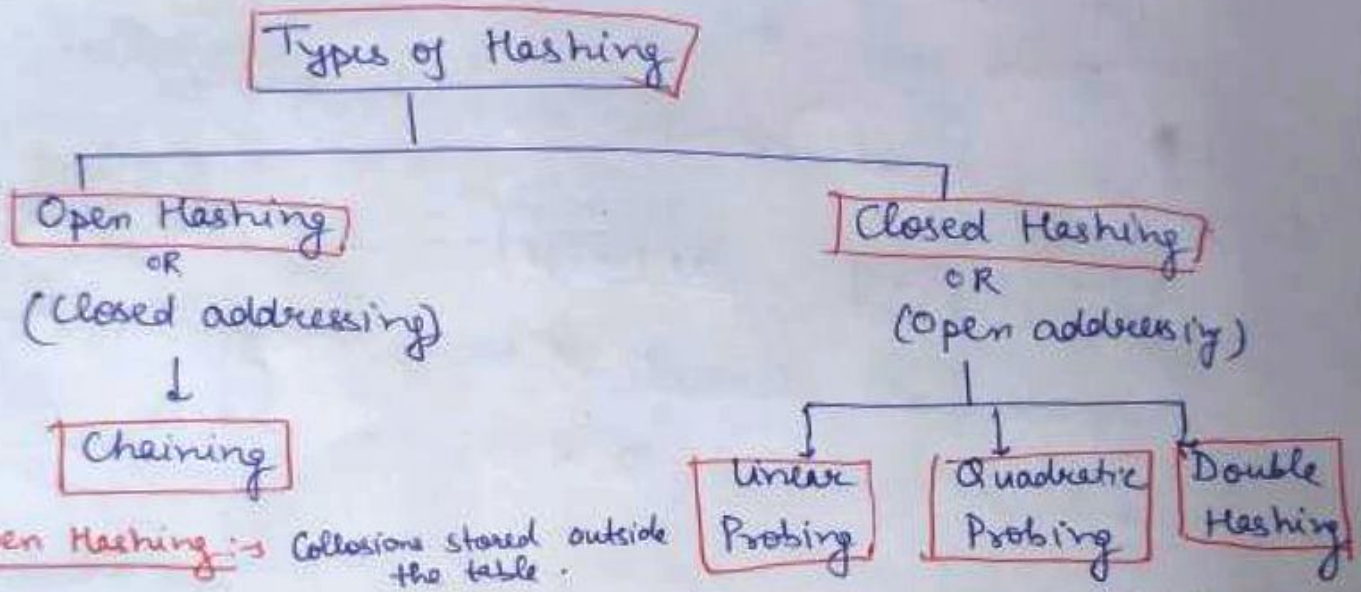
Hashing & Collision resolution techniques :-

Hashing is the process or technique of mapping keys, and values into the hash table by using a hash function.

Eg:- $H(x) = x \% 10$ in an Array. The list is {11, 12, 13, 14}, it will be stored at {1, 2, 3, 4} because

① $11 \% 10 = 1$ ② $12 \% 10 = 2$ ③ $13 \% 10 = 3$ ④ $14 \% 10 = 4$

0	1	2	3	4
	11	12	13	14



@ Open Hashing :- Collisions stored outside the table.

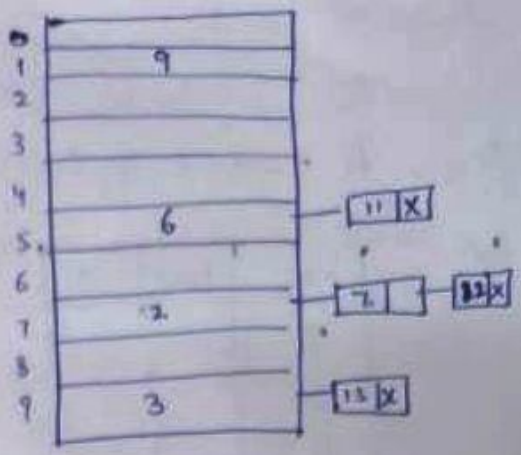
1. Chaining :→ It is implemented using linked list. Elements are hashed into the same slot index, those are added to a chain.

Q. A = 3, 2, 9, 6, 11, 13, 7, 12 where $m = 10$, $h(k) = 2k + 3$.

Sol:

* Three ways of calculating the hash function :-
  - division method
  - Folding method
  - Mid-square method

Key	location
3	$(2\times3+3)\%10 = 9$
2	7
9	1
6	5
11	5
13	9
7	7
12	7



(b) Closed Hashing :- In closed hashing, all keys are stored in the hash table itself without the use of linked list.

1. Linear Probing :→ It is used to search the closest free locations & adds new key to that cell, when collision occurs.

Q. A = 3, 2, 9, 6, 11, 13, 7, 12 where m = 10, $h(k) = 2k+3$.

Sol:

Key	Location	Probes
3	9	1
2	7	1
9	1	1
6	5	1
11	5	2
13	9	2
7	7	2
12	7	6

Sequence → 13, 9, 12, –, –, 6, 11, 2, 7, 3

* $(u+i)\%m$ where $i = 0$ to $m-1$

2. Quadratic Probing :→ It is an open addressing technique that uses quadratic polynomial for searching until a empty slot is found.

* It can also be defined as that it allows the insertion

K : at first free location from $(u+i^2)\%m$ where $i=0$ to $m-1$.

Q. A = 3, 2, 9, 6, 11, 13, 7, 12 where $m=10$, $h(k)=2k+3$

Sol:

Key	Location	Probe
3	9	1
2	7	1
9	1	1
6	5	1
11	5	2
13	9	2
7	7	2
12	7	5

0	13
1	9
2	
3	12
4	
5	6
6	11
7	2
8	7
9	3

$(u+i^2)\%m$

For k=11,

⊙ $u=5$, $i=0$

$= (5+0)\%10 = 5$ → already full

⊙ $i=1$ $= (5+1^2)\%10 = 6$ → put 11 at 6.

For k=13,

⊙ $u=9$, $i=0$

$= (9+0^2)\%10 = 9$ → already full

⊙ $i=1$ $= (9+1^2)\%10 = 0$ → put 13 at 0.

For k=7,

⊙ $u=7$, $i=0$

$= (7+0^2)\%10 = 7$ → already full

⊙ $i=1 \Rightarrow (7+1^2)\%10 = 8$ → put 7 at 8.

For k=12

⊙ $u=7$, $i=0$ ⇒ 7 → full

⊙ $i=1$ ⇒ 8 → full

⊙ $i=2$ ⇒ 1 → full

⊙ $i=3$ ⇒ 6 → full

⊙ $i=4$ ⇒ 3 → Put 12 at 3.

insert 13.

3. Double Hashing :→ In this, two hash functions are used. One is used for calculating the locations, where as another can be defined as hash function. It can also be defined "insert k_i at first place from $(u+v*i)\%m$ where $i = 0$ to $m-1$.

$u \rightarrow$ location computed

$v \rightarrow (h_2(k)\%m)$.

Q. A = 3, 2, 9, 6, 11, 13, 7, 12 , $m = 10$, $h_1(k) = 2k+3$, $h_2(k) = 3k+1$.

Sol:

Key	Location(u)	v	Probes
3	9	-	1
2	7	-	1
9	1	-	1
6	5	-	1
11	5	4	3
13	9	0	-
7	7	2	-
12	7	7	2

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

For k = 11,

- $u = 5$, $v = (3k+1)\%10 = 34\%10 = 4$
- $i = 0$, $(u + v*i)\%10 = 5+4\times0 = 5$ → already full
- $i = 1$, $(5+4\times1)\%10 = 9$ → already full
- $i = 2$, $(5+4\times2)\%10 = 3$ → put 11 at 3.

For k = 13,

- $u = 9$, $v = (3\times13+1)\%10 = 0$
- $i = 0$, $(9+0\times0)\%10 = 9$ → already full.
- $i = 1$, $(9+0\times1)\%10 = 9$ → already full.

Always, value will be 9, but 9 is not free so we cannot insert 13.

For k = 7, $v = (3 \times 7 + 1) \% 10 = 22 \% 10 = 2$

① u = 7, v = 2

① i = 0, $(7 + 2 \times 0) \% 10 = 7$ → already full.

② i = 1, $(7 + 2 \times 1) \% 10 = 9$ → already full.

③ i = 2, $(7 + 2 \times 2) \% 10 = 1$ → already full.

④ i = 3, $(7 + 2 \times 3) \% 10 = 3$ → already full.

⑤ i = 4, $(7 + 2 \times 4) \% 10 = 5$ → already full.

⑥ i = 5, $(7 + 2 \times 5) \% 10 = 7$ → already full

⑦ i = 6, $(7 + 2 \times 6) \% 10 = 9$ → already full

⑧ i = 7, $(7 + 2 \times 7) \% 10 = 1$ → already full

⑨ i = 8, $(7 + 2 \times 8) \% 10 = 3$ → already full.

⑩ i = 9, $(7 + 2 \times 9) \% 10 = 5$ → already full

Can't insert it in the table because range of i is 0 to 9 only.

For k = 12, $v = (3 \times 12 + 1) \% 10 = 7$

① u = 7, v = 7

① i = 0, $(7 + 7 \times 0) \% 10 = 7$ → already full

② i = 1, $(7 + 7 \times 1) \% 10 = 4$ → put 12 at 4.