Date - 21/09/19

UNIT - 3

QUICK SORTING -

```
int partition(char *A, int p, int r)
{
    int x, i, j, t;
    x = a[r];
    i = p-1;
    for(j=p; j<=r-1; j++)
    {
        if(a[j] <= x)
        {
            i = i+1;
            t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    t = a[i+1];
    a[i+1] = a[r];
    a[r] = t;
    return i+1;
}

Quicksort(char *A, int p, int r)
{
    int q;
    if(p < r)
    {
        q = partion(char *A, p, r)
        Quicksort(A, p, q-1);
        Quicksort(A, q+1, r);
    }
}
```

23/09/19

MERGE SORT - ($n \log_2 n$)

```
Merge (A, p, q, r)
{
    n1 = q - p + 1;
    n2 = r - q;
int L1[n1+1], L2[n2+1], L1[n1+1] = ∞, L2[n2+1] = ∞;
for (i = p; i <= n1; i++)
    L[i] = A[i];
for (j = 0; j <= n2; j++)
    R[j] = A[j].
i = 0;
j = 0;
for (K = p; K <= r; K++)
{
    if (L[i] <= R[j])
    {
        A[K] = L[i];
        i++;
    }
    else
        A[K] = R[j];
        j++;
} }.
```

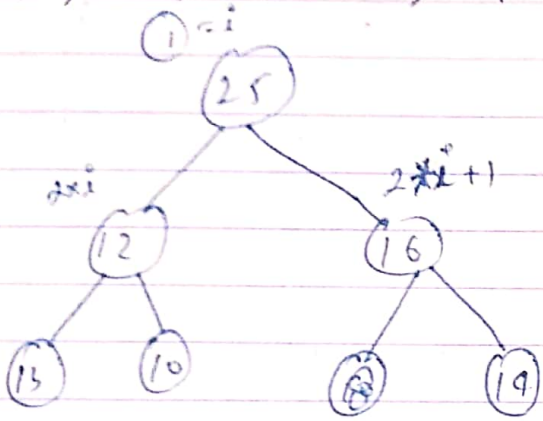
```
Merge-sort (A, P, r)
{
    if (P < r)
    {
        q = (p + r)/2;
```

$h = \log_2 n$ $\quad (2^{h+1} - 1) = n$ (nodes)

```
Merge_sort (A, p, q);
Merge_sort (A, q+1, r);
merge (A, p, q, r);
}
}
```

HEAP SORT.

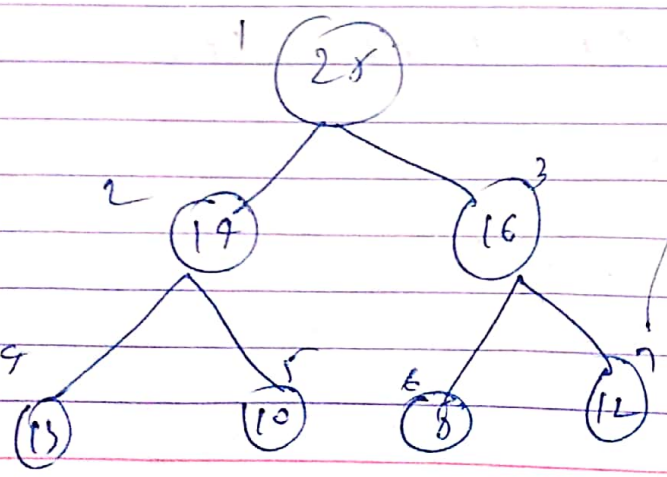
Construct almost binary tree of array
25, 12, 16, 13, 10, 8, 14



$P[i] = \left\lfloor \frac{i}{2} \right\rfloor$.

→ Heapify ← Max Heapify (for increasing order)
Min Heapify (for decreasing order)

Q 25, 14, 16, 13, 10, 8, 12. find Heap size



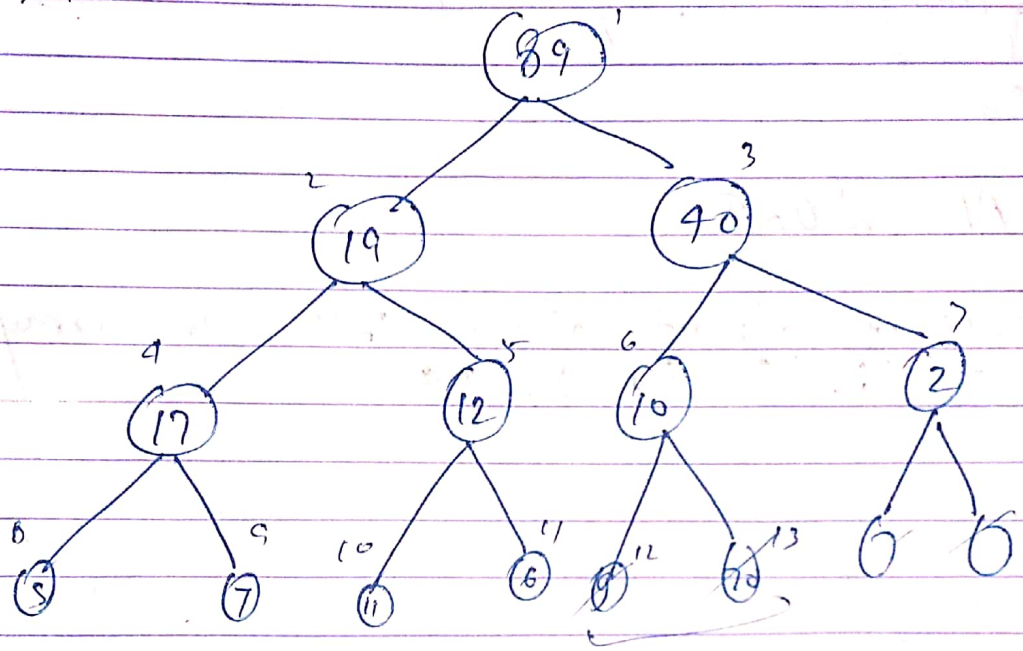
Heap size = 7.

Heap length = 7.
array

If heap size = heap length then it is either max or min heapify.

Q What is the heap size & heap length of given array - 89, 19, 40, 17, 12, 10, 2, 5, 7, 11, 6, 9, 70

⇒



Heap Size = 2. & Heap length = 13.
(Condition validates - 1)

Parent node from 1 to $\left\lceil \frac{\text{Size of array}}{2} \right\rceil$

Leaf node from $\left\lceil \frac{\text{Size of array}}{2} \right\rceil + 1$ to n.

ALGORITHM OF HEAP SORT -

```
Heap-sort (A)
{
(1)  Build_max heap (A)
   for (i = A.length  down to 2
   {
      exchange a[1] with a[i]
```

```
        A.heapsize--;
        Max_heapify(A, 1)
    }
}

build_Maxheap(A)
{
    A.heapsize = A.length
    for(i = A.heapsize)
    for(i = A.length/2 down to 1)
        Max_heapify(A, i).
}
max_heapify(A, i)
{
    l = 2*i
    r = 2*i + 1
    if(l <= A.heapsize && A[l] >= A[i])
        largest = l else largest = i
    else if(r <= A.heapsize && A[r] > A[largest])
        largest = r.
    if(largest != i)
    exchange A[i] with A[largest]
        Max_heapify(A, largest())
}
```

Date - 30/08/19

HASHING -

Q What is hashing?

Answer: * Hashing is the process of maping large amount of data item to smaller table with the help of hashing function.

* Hashing is also known as message digest function.

* It is a technique to convert a range of key values into range of indexes of an array.

* It is used to facilitate next level searching method when compared with linear & binear search.

* Hashing allow to update & retrive data entry in a constant time i.e., O(1). with the help of hashing function.

* Constant time O(1) means operation doesnt depend on size of data.

* Hashing is used with a database to enable to be retrive more quickly.

Q What is hash function ?.

Answer A fixed process that convert a key & map it to a value of certain length which is called hash value or hash.

Hash value represents the original string of character, but it is normally smaller than the original value. The parameters of good hashing function are i.e., easy to complute, even distribution & minimizk collision

Q What is hash table ?.

Answer * Hash table or hash map is a datastructure used to store key value pairs.

* It is a collection of item store to make it easy to find them later.

* It uses a hash function to compute an index into an array of buckets or slot from which the desired value can be formed.

* It is an array of list where each list is known as bucket (a node of link list

* It contains values based on the key.

* Hash table is used to implement the map interface & extends the dictionary class.

* Hash table is synchronized & contains only unique elements.

HASH FUNCTION TYPES - (Algorithm) (H.F)

(i) 1) DIVISION -

M = 1000 (0 - 999) (No. of memory slots)
Key = 12345789
H(Key) = Key % m

(ii) 2) SQUARE METHOD -

M = 1000 (0 - 999)
Key = 8492
H(Key) = $(8492)^2$ = 72114064
= 114 or 140.

(iii) Digit Extraction Method -

M = 10 (0 - 9)	M = 1000 (0 - 999)
Key = 12345789 (0 1 2 3 4 5 6 7 8)	Key = 12345678910 (0 1 2 3 4 5 6 7 8 9)
H(Key) = 5 (4th position)	H(Key) = 260 (1, 5, 7)

(iv) Folding Method -

- Fold Boundary
- Fold & Shifting method.

(a) Fold Boundary Method -

M = 1000 (0-999)

Key = 123456789

H(Key) = 123 + 789 = 912 (index)

Key = 789456789.

H(Key) = 789 + 789 = 1578 → 157 + 8 = 165 (index)

(b) Fold Shifting Method -

M = (0-999)

Key = 123456789

H(Key) = 123 + 456 + 78 + 9 = 1368 → 136 + 8 = 144 (index)

123
456
789
1368

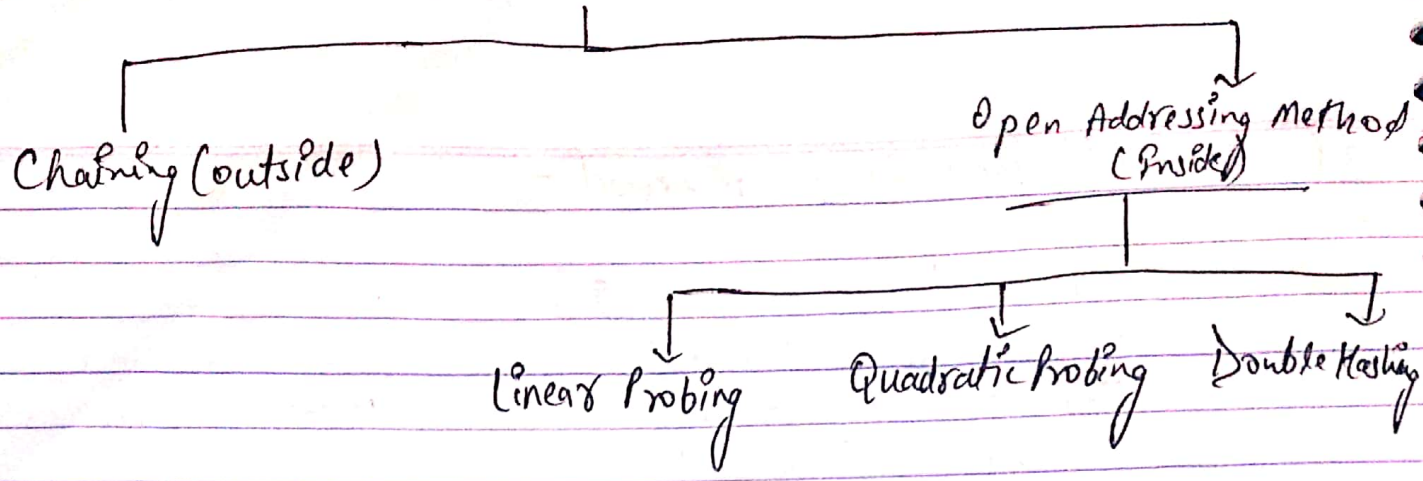
(v) Binning Method -

M = 10 (0-9)

Key Value Range (0-999)

0-99	Slot 0.
100-199	Slot 1
200-299	Slot 2
300-399	Slot 3
⋮	⋮
900 999	slot 9

COLLISION RESOLUTION TECHNIQUES -



COLLISION -

In hashing hash function used to compute hash value for a key. Has value is then used as an index to store the key in the hash table. Hash function may return the same hash value for two or more key. When the hash value of a key map to an already occupy bucket of hash table it is called collision

COLLISION RESOLUTION TECHNIQUE

Collision Resolution Technique is used for resolving or handling the collision. These techniques are classified as above given.

* Separate Chaining -

To handle the collision this technique creates a link list to the slot for which collision occurs. The new key is then inserted in the link list. These link list to the slot appears like a chain that's why this technique is known as

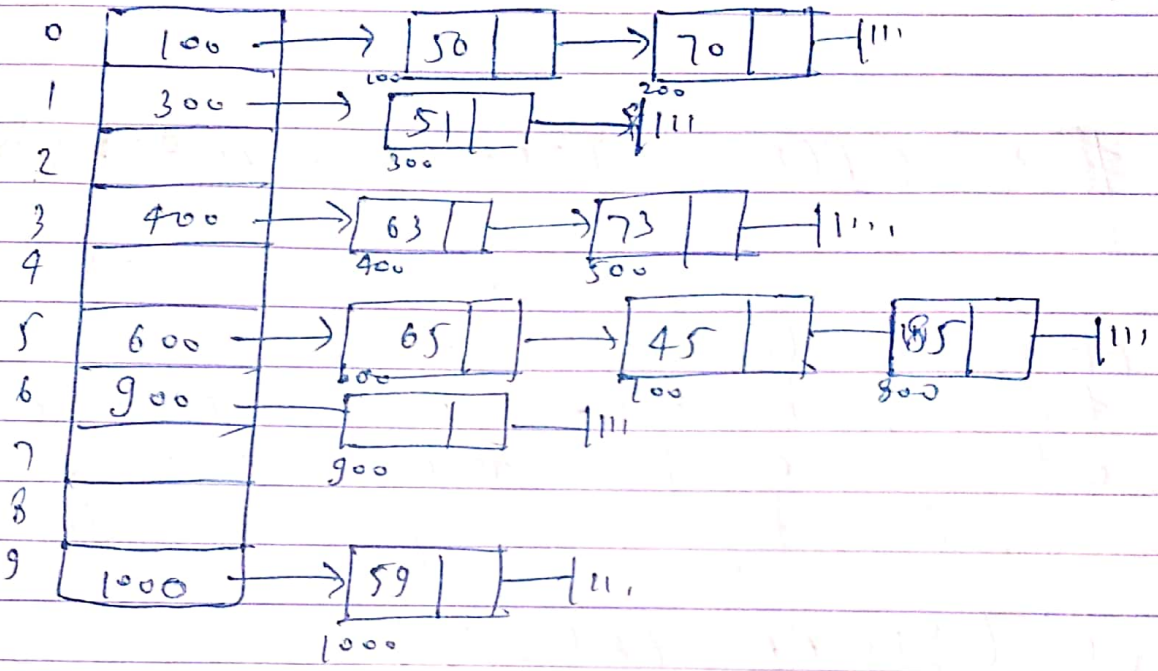
No. of Size.

Seperate chaining.

NOTE - The hash table contains array of pointers (addresses) which points to the first node of list of corresponding hash key index.

ex- m = 10. (0-9)

elements = 50, 70, 51, 59, 65, 45, 85, 63, 73, 86



- TIME COMPLEXITY FOR SEARCHING -

(a) In worst case all the key might mapped to the same bucket of hash table.

(b) In such case all key will present in single link list. So, time taken for searching in worst case is O(n)

- LOAD FACTOR (α)

$$\alpha = \frac{\text{No. of element present in hash table}}{\text{Total size of hash table}}$$

Time complexity = $O(\alpha)$

ex- If elements lies (0-99) & M = 10.

$\therefore \alpha = \frac{100}{10} = 10.$

* OPEN ADDRESING METHOD -

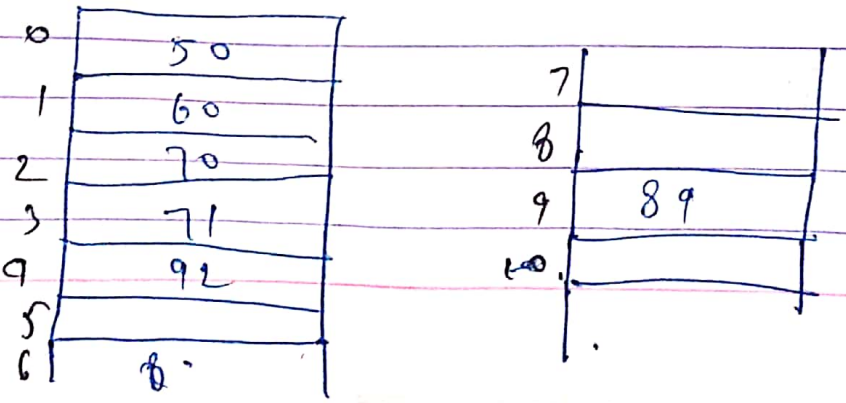
In open addressing unlike separate chaining all the Keys are stored inside the hash table i.e., no Key is stored outside the hash table memory.

Various type of open adressing method are -

(i) LINEAR PROBING -

When a collision occurs we linearly probe for the next bucket. We Keep probing untill an empty bucket is found.

ex- M=10, elements = 50, 60, 70, 71, 89, 92,



(ii) QUADRATIC PROBING.

In Quadratic Probing when collision occurs, we probe for i^2 th bucket in ith operation iteration. We keep probing untill an empty bucket is found.

Ex - $m = 10$, elements - 50, 60, 70, 71, 92, 95, 89.

50 mod 10 = 0.
60 mod 10 = 0
$(60 + 1^2)$ mod 10 = 1.
70 mod 10 = 0 coll.
$(70 + 1^2)$ mod 10 = 1 coll.
$(70 + 2^2)$ mod 10 = 4 coll.

0	50
1	60
2	71
3	92
4	70
5	95
6	
7	
8	
9	89.

(iii) DOUBLE HASHING -

In Double hashing we use another hash function say $h_2(x)$ & look for i th $h_2(x)$ bucket in ith iteration. It requires more computation time as two hash function needed to be computed.