3/11/12 Data Structure

Set - A

1) Shell short:

Shell short sorts elements that are far apart from each other and successively reduces the interval between the elements to be sorted.

Algorithm:

```
Shell short ( )
  {
  for (gap = n/2; gap >= 1 ; gap /= 2)
   {
    for (j = gap ; j<n; j++)
     {
      for ( i = j - gap; i>=0; i-gap)
       {
        if (a[i+gap] > a[i])
         {
          break;
         }
        else
         {
          swap (a[i+gap], a[i])
         }
       }
     }
  }
```

Example:

Example

23	29	15	19	31	7	9	5	2

gap = 4

pass-1

23 7 15 19 31 29 9 5 2
23 7 9 19 31 29 15 5 2
23 7 9 5 31 29 15 19 2
23 7 9 5 2 29 15 19 31
2 7 9 5 23 29 15 19 31

gap = 2

2 5 9 7 23 29 15 19 31
2 5 9 7 15 29 23 19 31
2 5 9 7 15 19 23 29 31

gap = 1

2 5 7 9 15 19 23 29 31

Time complexity:

B-C : O(n log n)
W.C : O(n²)
A.C : O(n log n)

---

2. Binary search:

* Binary search is used for finding element position in sorted array.

In this approach the element is always searched in middle of portion of an array.

Algorithm:

```
Binary search ( int A[], int x, int low, int high)
{
    while ( low <= high)
    {
    if (a[mid] == x)
        return mid;
    if (a[mid] < x)
        low = mid + 1;
    else
        high = mid - 1;
    }
    mid = (low + high) / 2;
}
```

Example:

Array: 3 4 5 6 7 (low, high), $x = 6$

$mid = (0 + 4)/2 = 2$ $\quad mid = 5$

$x > a[mid]$

so $low = mid + 1$

$= 2 + 1$

$= 3$

Array: 3 4 5 6 7
low high

mid = (low + high)/2 = (3+4)/2

= 3

a[mid] = 6 = x

=

Hence element found at 3rd position.

ii) Rehashing:

Rehashing means hashing again. Basically when the load factor increases to more than its pre defined value (default value of load factor is 0.75), the complexity increases. So to overcome this, the size of the array is increased (doubled) and all the values are hashed again and stored in new double size array to maintain a low load factor and low complexity.

Extendible hashing:

It is a dynamic hashing method wherein directories and buckets are used to hash data. It is an aggressively flexible method in which the hash function also experience dynamic change.

Main features:

* Directories: The directories store address of buckets in pointers.

* Buckets: The buckets are used to hash the actual data.

---

3). Merge sort:

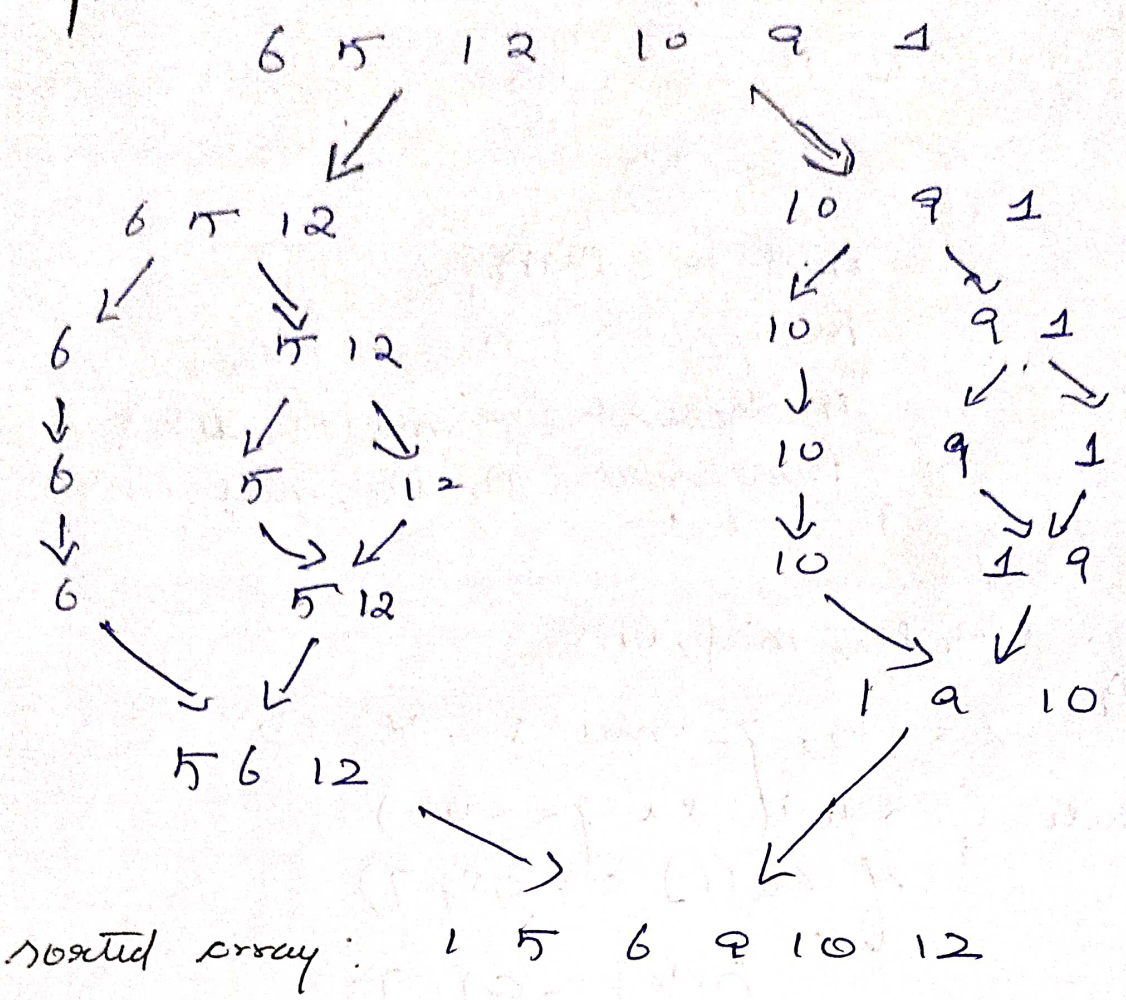
* It is based on principle of divide and conquer.

* A problem is divided into multiple sub problems. Each problem is solved individually. Finally sub problems combined to form final solution.

```
mergesort ( A, lb, ub)
{
    if (lb < ub)
    {
        mid = (lb+ub)/2
        mergesort (A, lb, mid)
        mergesort (A, mid+1, ub)
        mergesort (A, lb, mid, ub);
    }
}

merge (A, lb, mid, ub)
{
    i = lb, j = mid+1, k = lb
    while (i <= mid && j <= ub)
    {
        if (a[i] <= a[j])
        {
            b[k] = a[i];
            i++;
            k++;
        }
        else {  b[k] = a[j];
                j++;
                k++;
             }
    }
    if (i >= mid)
    {
        while (j <= ub)
        {
            b[k] = a[j];
            j++;
            k++;
        }
    else [ while (i <= mid)
           { b[k] = a[i];
             i++;
             k++;
           }
    }
```

Example:



---

4. Selection sort:

The selection sort algorithm sorts an array by repeatedly finding minimum element from unsorted part and putting it at beginning.

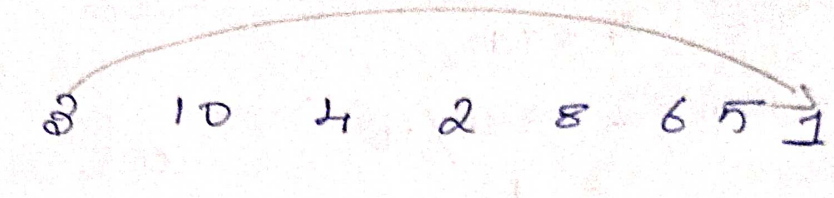
Algorithm:

```
Selection sort()
{
  for (int i=0; i<n-1; i++)
    {
      int min = i;
      for ( j = i+1; j<n; j++)
```

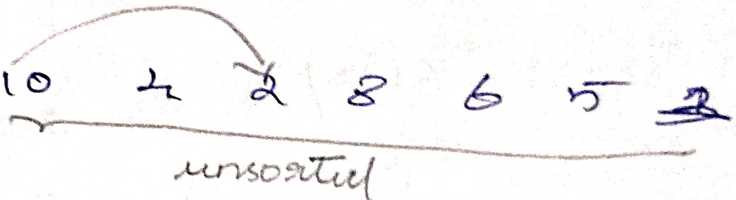
```
if (a[j] < a[min])
{
    min = j;
}
if (min != i)
    swap (a[min], a[i])
}
```

Example:

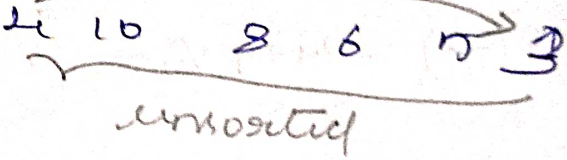
Array: 8 10 4 2 8 6 5 1



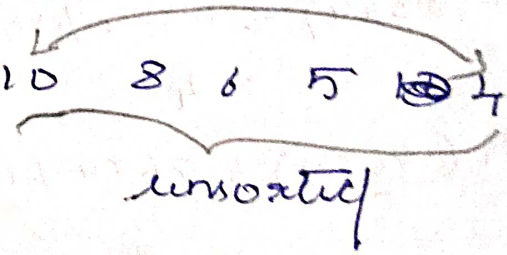
pass 1: 1 10 4 2 8 6 5 8



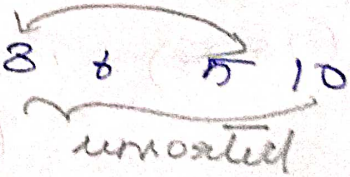
pass 2: 1 2 4 10 8 6 5 3



pass 3: 1 2 3 10 8 6 5 4



pass 4: 1 2 3 4 8 6 5 10



pass 5: 1 2 3 4 5 6 8 10

pass 6 and 7 continues since the array is sorted no element would be swapped.

B.C : $O(n^2)$

W.C : $O(n^2)$

Q.

i) input : { 4371, 6173, 4199, 4344, 9679, 1989 }

i Open hash table : X closed addressing :

$h(x) = x \% 10$

i) $x = 4371$

$h(4371) = 4371 \% 10$

$= 1$

$h(6173) = 6173 \% 10$

$= 3$

$h(4199) = 4199 \% 10$

$= 9$

$h(4344) = 4344 \% 10$

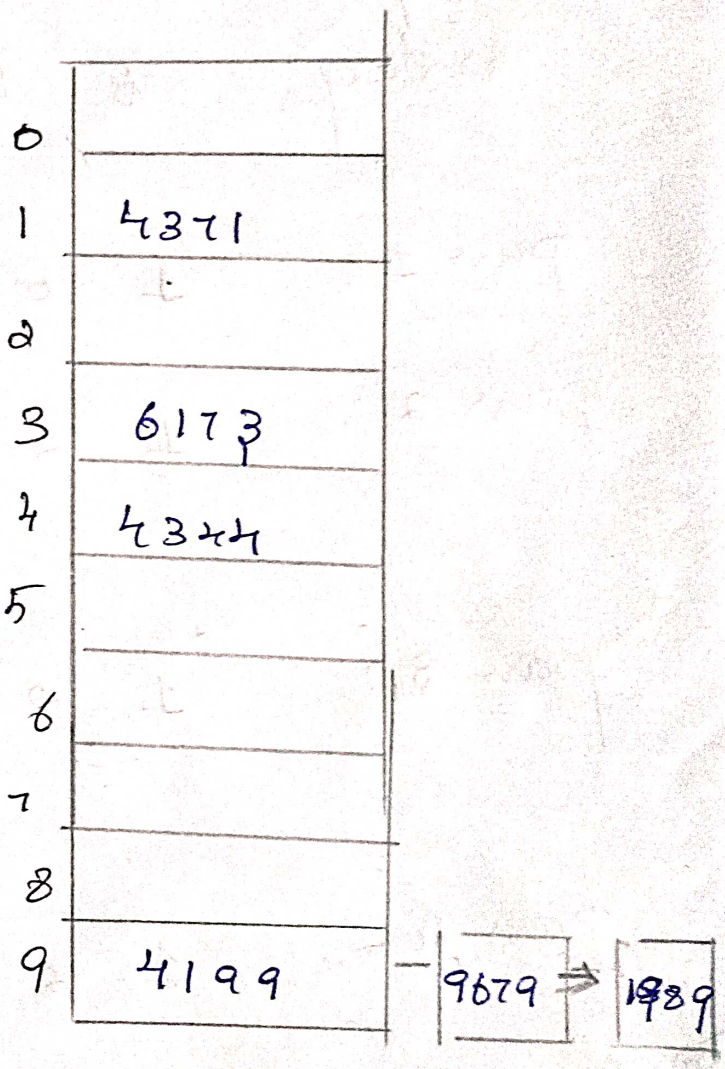
$= 4$

$h(9679) = 9679 \% 10$

$= 9$

$h(1989) = 1989 \% 10$

$= 9.$



ii) Linear probing :

$h(x) = (x+i) \% 10$

$h(4371) = 4371 \% 10$

$= 1$

$h(6173) = 6173 \% 10$

$= 3$

$h(4199) = 4199 \% 10$

$= 9$

$h(4344) = 4344 \% 10$

$= 4$

$h(9679) = 9679 \% 10$

$= 9$ (not possible)

$= (9679+1) \% 10$

$= 0$

$h(1989) \% 10$

$= 9$ (not possible)

$= (1989+1) \% 10$

$= 0$ (not possible)

$= (1989+3) \% 10$

$= 2$

0	9679
1	4371
2	1989
3	6173
4	
5	
6	
7	
8	
9	4199

iii) Quadratic Probing..

$h(x) = (x + i^2) \% 10$

$h(4371) = (4371) \% 10$

$= 1$

$h(6173) = 6173 \% 10$

$= 3$

$h(4199) = 4199 \% 10$

$= 9$

$h(4344) = 4344 \% 10$

$= 4$

$h(9679) = (9679 + 1) \% 10$

$= 0$

$h(1989) = (1989 + (3)^2) \% 10$

$= 1998 \% 10$

$= 8$

0	9679
1	4371
2	
3	6173
4	4344
5	
6	
7	
8	1998
9	4199