SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

UNIT-III

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

Searching techniques :- There are two ways :-

1) Linear Search
2) Binary Search

Linear Search :-

⇒ linear search is a very simple search algorithm.

⇒ A sequential search is made over all items one by one.

⇒ Every item is checked & if a match is found then that particular location is returned otherwise search continues till the end of the data collection.

Algo. :-

→ Read the element to be searched.

→ compare the search element with the first element in the list.

→ if matched, display "Given element is found."

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

→ if both are not matched, compare search element with the next element in the list.

→ Repeat ~~to~~ the above steps until search element is compared with last element in the list.

```
void main() {
    int A[20], n, i, ele;
pf(" Enter the size of the list");
sf(" %d", &n);

for(i=0; i<n; i++)
    sf(" %d", &A[i]);
// linear search logic
for(i=0; i<n; i++)
 {
    if (ele == A[i]) {
   pf("Element is found at %d index", i);
      break;
   }
 }
 if (i==n)
   pf(" Not found");
 getch();
}
```

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Binary Search :-

* Binary Search algo. can be used with only sorted list of elements.
* This search process starts comparing the search element with the middle element in the list.
* If both are matched, result is declared as found. otherwise we check whether the search element is smaller or larger than the middle element in the list.
* then the same process is repeated for upper or lower half.

// C Program for binary search.

```
void main() {
  int first, last, middle, n, i, ele, A[30];
  pf(" Enter the size of the list");
  sf(" %d", &n);
  pf(" Enter %d integer values in ascending
       order ", n);
   for (i=0; i<n; i++)
      sf("%d", &A[i]);
```

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

```
pf(" Enter the value to be searched");
 sf(" %d", & ele);
  first = 0;
  last = n-1;
 middle = (first+ last)/2;
  while ( first <= last)
  {
    if (A[middle] < ele)
         first = middle +1;
    else if (A[middle]== ele)
     {
     pf(" %d found at location %d", ele, middle +1);
     break;
   }
  else
    last = middle - 1;
   middle = (first + last)/2;
 }
  if (first > last)
   pf(" Element %d not found", ele);
 return 0;
}
```

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Sorting

* Sorting refers to the operation of arranging data in some given order, such as increasing or decreasing.

Bubble Sort :-

* Bubble sort is a simple algorithm which is used to sort a given set of n elements by comparing all the elements one by one & sort them based on their values.
* If the array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if first element is greater than the second element, it will swap both the elements. then move on to compare the second & third element & so on.
* It is known as bubble sort because with every complete iteration / pass the largest element in the given array, bubbles up towards the last place. (like a water bubble rises up to the water surface.)

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Example :-

Pass / Iteration 1

14, 33, 27, 35, 10

No Swap

14, 33, 27, 35 10

Swap

14, 27, 33, 35, 10

No Swap

14, 27, 33, 10, 35

Pass 2 :-

14, 27, 33, 10, 35

No Swap

14, 27, 33, 10, 35

No Swap

14, 27, 33, 10, 35 ⇒ 14, 27, 10, 33, 35

Swap

Pass 3 :-

14, 27, 10, 33, 35

No Swap

14, 27, 10, 33, 35 ⇒ 14, 10, 27, 33, 35

Swap

Pass 4 :-

14, 10, 27, 33, 35 ⇒ 10, 14, 27, 33, 35

Swap

Sorted List

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

BubbleSort (n, list)

Step 1 :- i = 0

Step 2 :- Repeat through step (5) while (i < n-1)

Step 3 :- j = 0

Step 4 : Repeat through step (5) while (j < (n-i)-1)

Step 5 :- if list[j] > list[j+1]
temp = list[j]
list[j] = list[j+1]
list[j+1] = temp

Step 6 :- Exit

Insertion Sort

* Insertion sort works similarly as we sort cards in our hands in a card game.

Note :- we assume that first card is already sorted, we select an unsorted card. If unsorted card is greater than the card in hand, it is placed on the right otherwise to the left.

Pass 1 :- 9, 5, 4, 1, 3

Pass 2 :- 9, 5, 4, 1, 3 ⇒ 5, 9, 4, 1, 3

Pass 3 :- 5, 9, 4, 1, 3 ⇒ 4, 5, 9, 1, 3

Pass 4 :- 4, 5, 9, 1, 3 ⇒ 1, 4, 5, 9, 3

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Pass 5 :- 1, 4, 5, 9, 3 ⇒ 1, 3, 4, 5, 9
Sorted list

algorithm :-

1) for j = 2 to n
2) Key = A[j]
3) i = j - 1
4) while (i > 0 and A[i] > Key) {
5) A[i+1] = A[i]
6) i = i - 1 }
7) A[j+1] = Key

C Program for Insertion Sort

```
main() {
    int A[50], i, j, K, n, temp;
    pf ("Enter no. of elements");
    sf ("%d", &n);
    pf ("Enter the array elements");
        for (i=0; i<n; i++)
            sf("%d", &A[i]);
    for (K=1; K<n; K++)
    {
            temp = A[K];
            j = K-1
        while ( temp < A[j] && j>=0) {
                A[j+1] = A[j];
```

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

```
            j = j-1;
        }
        A[j+1] = temp;
    }
```

Q :- 25, 15, 30, 9, 99, 20, 26

Selection Sort

* Selection Sort algorithm works by finding the smallest number from the array & then placing it to the first position.
* The next array that is to be traversed will start from index next to the position where the smallest no. is placed.

15, 20, 10, 30, 50, 18, 5, 40

Iteration 1 :- 15, 20, 10, 30, 50, 18, 5, 40

No Swap

15, 20, 10, 30, 50, 18, 5, 40

Swap

⇓

10, 20, 15, 30, 50, 18, 5, 40

No Swap

No swap

No Swap

Swap

⇓

5, 20, 15, 30, 50, 18, 10, 40

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

After Iteration 2 :- 5, 10, 20, 30, 50, 18, 15, 40

After Iteration 3 :- 5, 10, 15, 30, 50, 20, 18, 40

after Iteration 4 : 5, 10, 15, 18, 50, 30, 20, 40

after Iteration 5 :- 5, 10, 15, 18, 20, 50, 30, 40

after Iteration 6 :- 5, 10, 15, 18, 20, 30, 50, 40

after Iteration 7 :- 5, 10, 15, 18, 20, 30, 40, 50

algorithm :-

Step 1 :- Select 1st element of the list.

Step 2 :- compare the selected element with all the other elements in the list.

Step 3 :- In every comparison, if any element is found smaller than the selected element, both are swapped.

Step 4 :- Repeat the same procedure with element in the next position in the list till entire list is sorted.

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

C Program for Selection Sort

```
void main() {
  int  n, i, j, temp, A[50];
  pf ("Enter the size of the array");
  sf ("%d", &n);
  for (i=0 ; i<n; i++)
     sf ("%d", &A[i]);
  // selection Sort logic
   for (i=0; i<n; i++)
   {
      for (j= i+1; j<n; j++) {
           if (A[i] > A[j])
            {
               temp = A[i];
               A[i] = A[j]=
               A[j] = temp;
            }
      }
}
```

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Quick Sort

54	26	93	17	77	31	44	55	20
Pivot	i							j

Increment i from L to R until we get an element greater than pivot (54). Simultaneously decrement j until we get less than pivot.

54	26	93	17	77	31	44	55	20
		i						j

Swap A[i] & A[j]

54	26	20	17	77	31	44	55	93
				i		j		

Swap A[i] & A[j]

54	26	20	17	44	31	77	55	93
					j	i		

as $i > j$, Stop the process & interchange A[j] with Pivot

31	26	20	17	44	54	77	55	93

Sublist - 1 (31, 26, 20, 17, 44) Sublist-2 (77, 55, 93)

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

* Thus after applying the same method again & again for new sublists, we get the sorted list.

Procedure QuickSort (Q, p, r)

1. if $p < r$ then
2. q = call to Partition (Q, p, r)
3. call QuickSort (Q, p, q-1)
4. Call QuickSort (Q, q+1, r)

Partition (Q, p, r)

This algo arranges the subarray Q[p...r]

1. Set $x = x[p]$
2. $i = p$
3. $j = r+1$
4. while ($i < j$) {
5. while ($Q[i] <= x$)
6. Set $i = i+1$
7. while ($Q[j] > x$)
8. Set $j = j-1$
9. if ($i < j$) then
10. Set $Q[i] \Leftrightarrow Q[j]$ } Swap
11. Swap Q[p] & Q[j]
12. return j.

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

(5) Heap Sort

* A heap is defined as an almost complete binary tree of n nodes such that the value of each node is less than or equal to the value of its father.

* This implies that root of binary tree has the largest key in key. This type of heap is usually called descending heap or max heap.

Sorting of array using heapsort procedure consists of 3 procedures :-

(1) Build-Max-Heap procedure produces a max heap from an unordered i/p array.

(2) Max-Heapify procedure is the key to maintain the max-heap property.

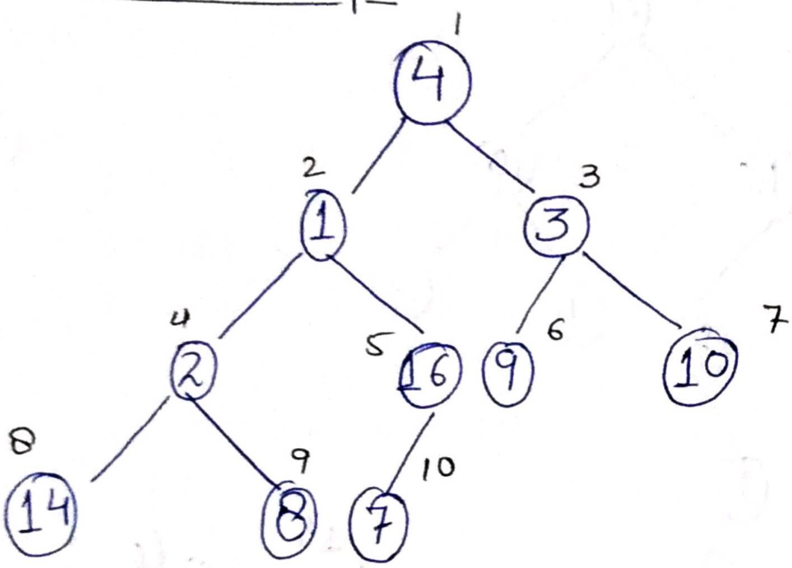
(3) Heap Sort, sorts an array.

Eg.)

4	1	3	2	16	9	10	14	8	7
1	2	3	4	5	6	7	8	9	10

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

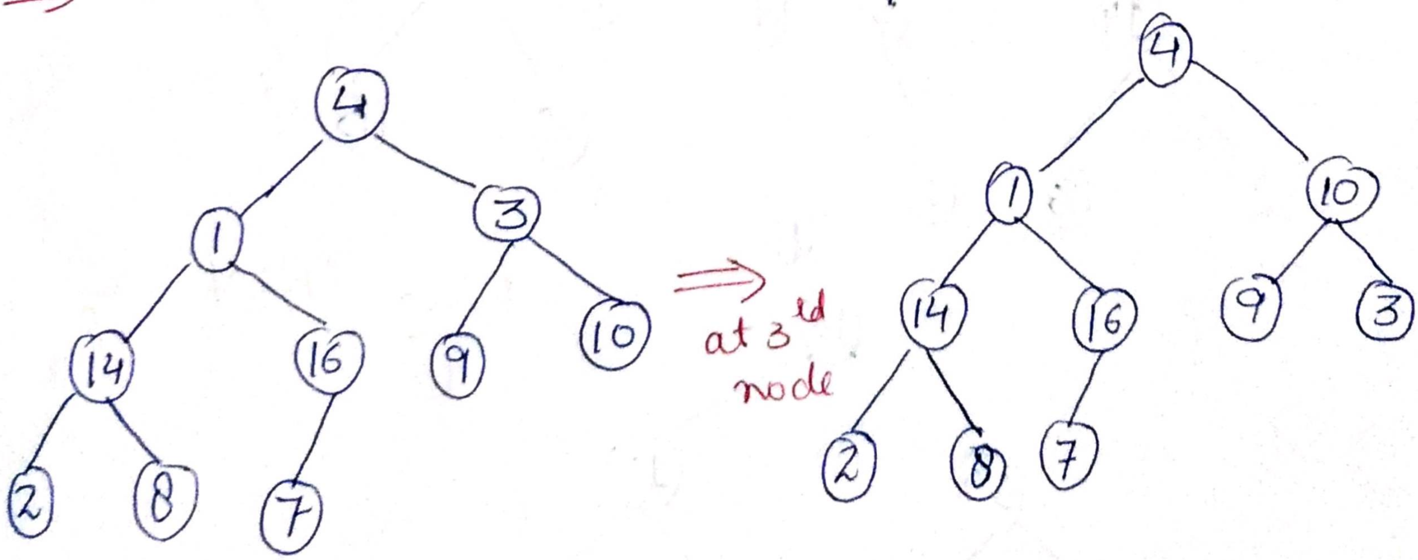
Build-Max-Heap :-



we'll start heapifying the above tree from length [A]/2 to 1 i.e., 5 to 1

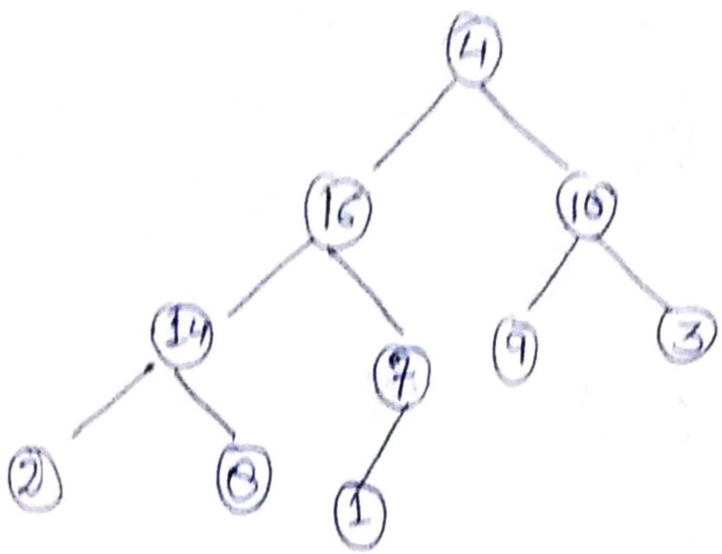
→ at 5th node Parent - child already creates a heap.

→ at 4th node we'll do following :-

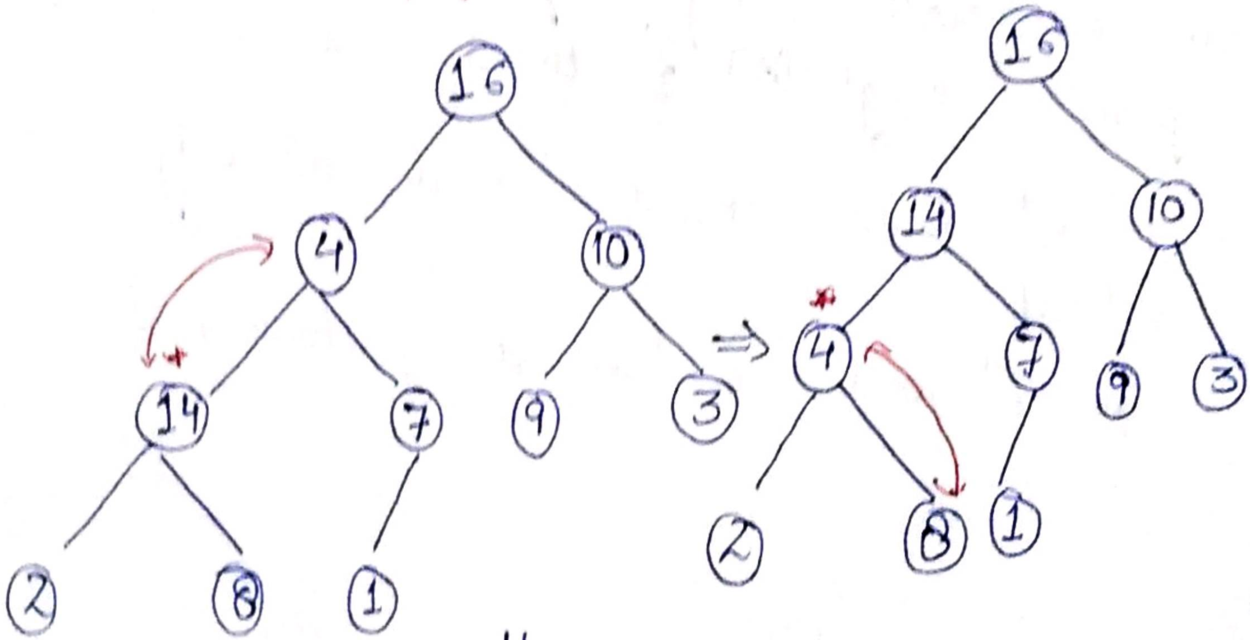


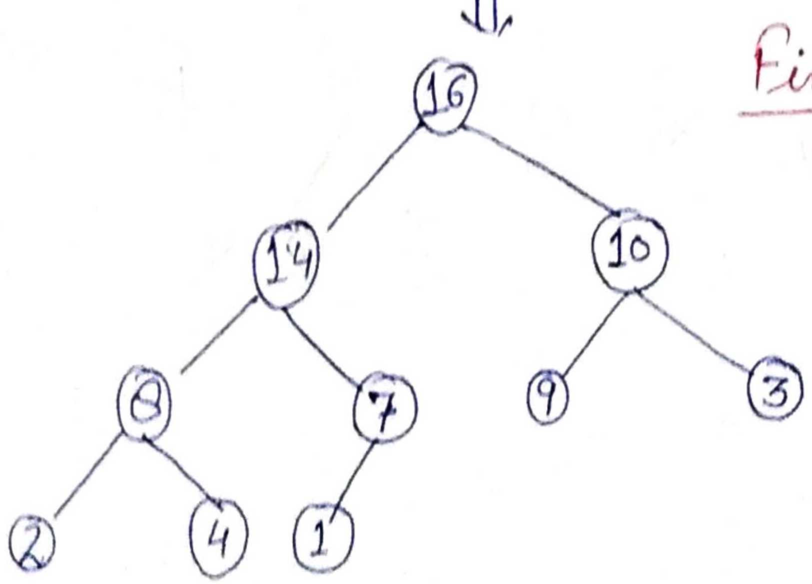
SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

after max-Heapify at 2nd node



after max-Heapify at 1st node :-





Final Heap

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Heap - Sort

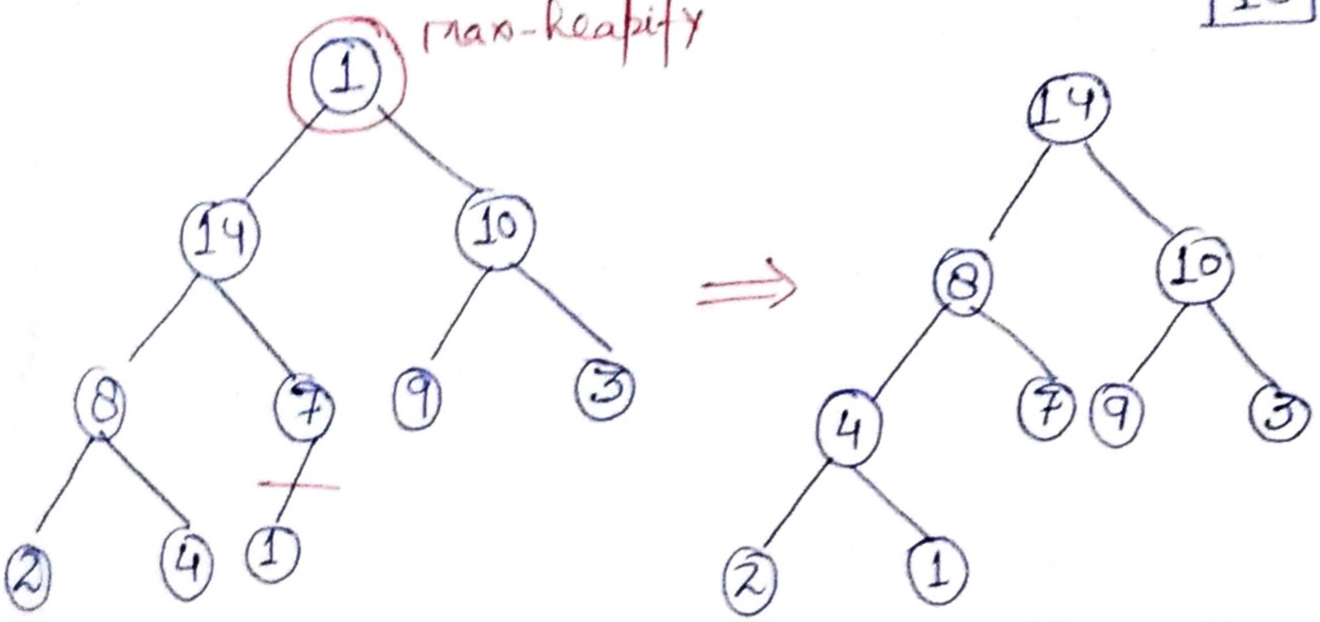
Swap A[1] with A[n]

Reduce n by n-1 then again

Max- Heapify at A[1]

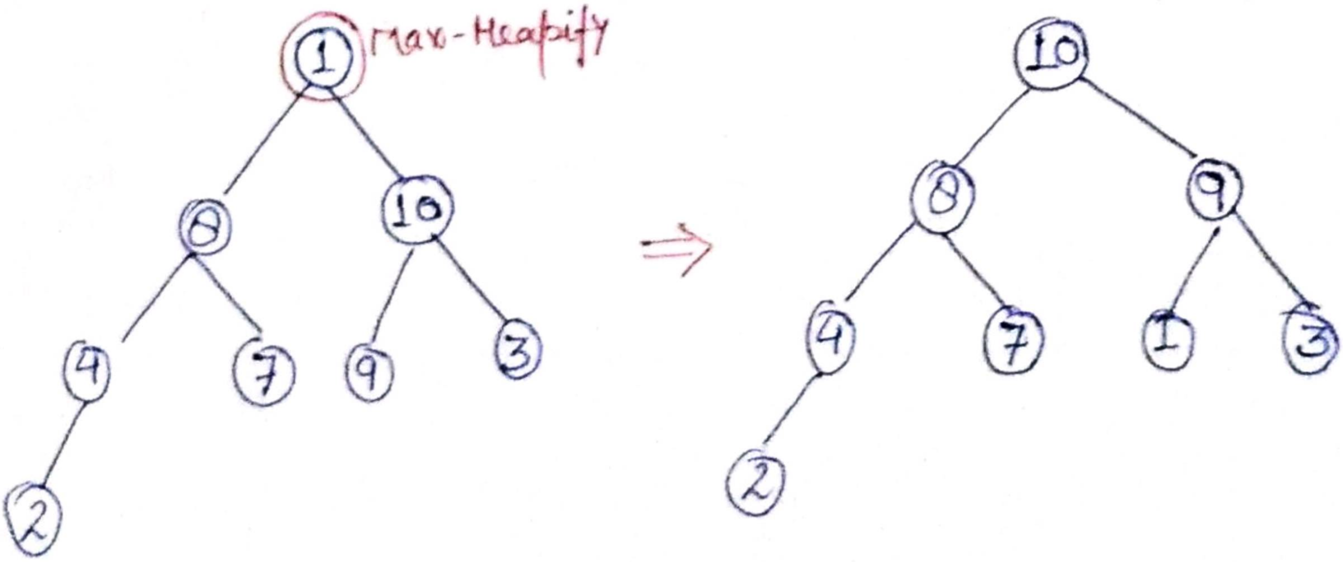
Iteration 1 :-

$\boxed{16}$



Iteration 2 :-

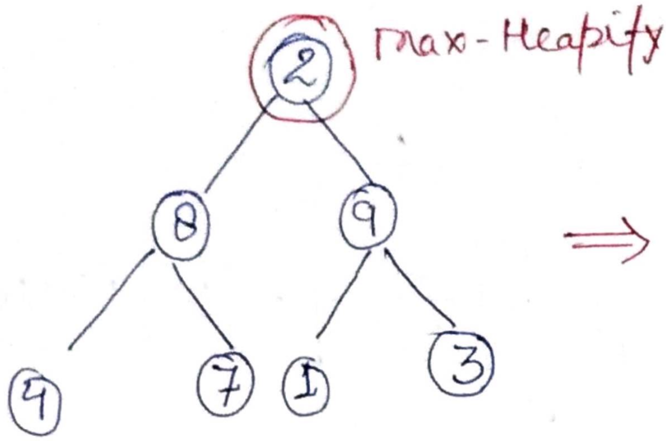
	14	16

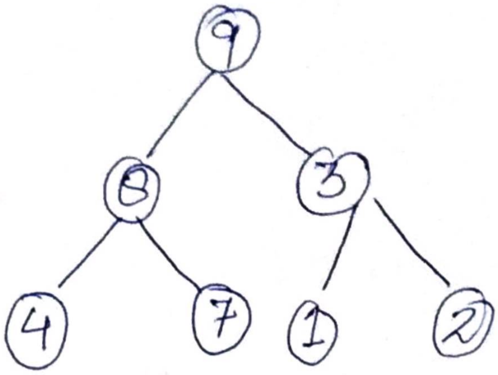


SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Iteration 3 :-

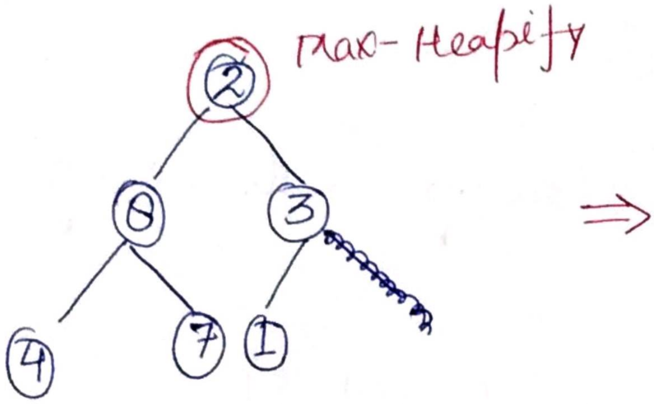
10	14	16

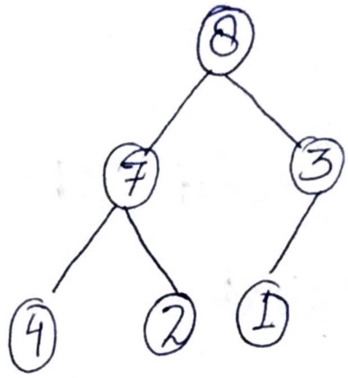




9	10	14	16

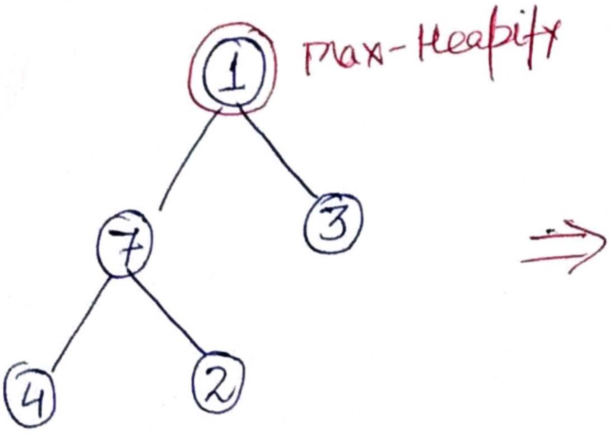
Iteration 4 :-

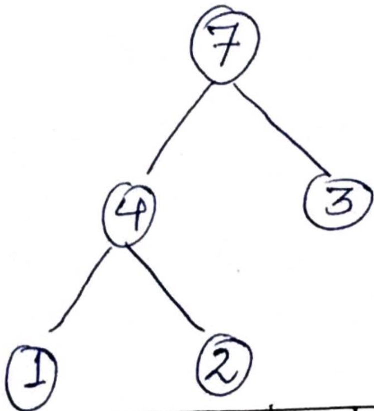




8	9	10	14	16

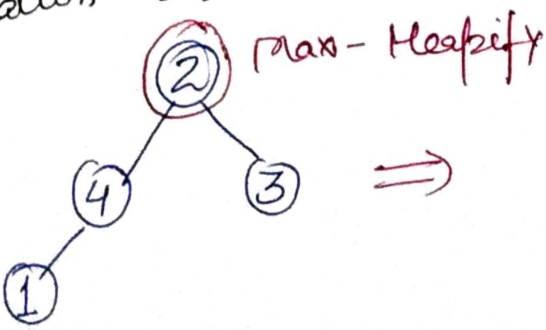
Iteration 5 :-

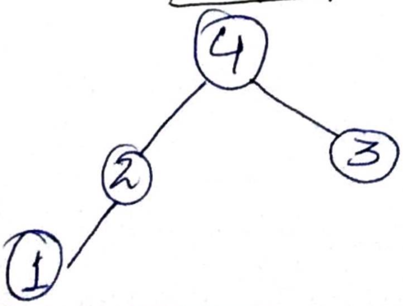




7	8	9	10	14	16

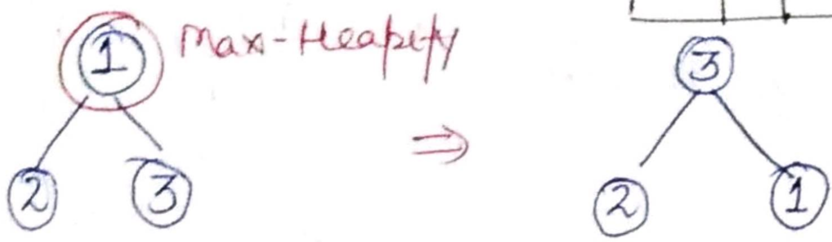
Iteration 6 :-





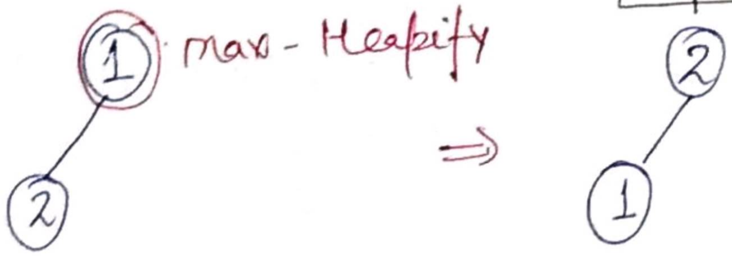
Iteration 7 :-

4	7	8	9	10	14	16



Iteration 8 :-

3	4	7	8	9	10	14	16



Question :- Sort the following list using Heap Sort

25, 55, 46, 35, 10, 90, 84, 31

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Max - Heapify :

* When Max-Heapify is called, it is assumed that the binary trees rooted at left(i) & right(i) are max heaps, but A[i] may be smaller than its children, thus violating the max-heap property.

* Thus its function is to let the value at A(i) "float down" in the heap so that subtree rooted at i becomes a max-heap.

```
Max-Heapify(A, i)
    l = left(i)
    r = Right(i)
    if l ≤ heap-size[A] and A[l] > A[i]
        then largest = l
            else largest = i
    if r ≤ heap-size [A] and A[r] > A[largest]
        then largest = r
    if largest ≠ i
        then exchange A[i] & A[largest]
            Max-Heapify (A, largest).
```

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

(15)

7 3 12

Build - Max-Heap (A)

1. Heap-size [A] = length[A]
2. for i = length[A]/2 downto 1
3. do Max-Heapify (A,i)

Running time :- O(n ~~lgn~~), as we'll call max-heapify ~~(lgn) for n times~~.

Heap-Sort (A)

1. Build-Max-Heap (A) — n
2. for i = length[A] downto 2
3. do exchange A[1] with A[i]
4. Heap-size [A] = Heap-size[A]-1
5. Max-Heapify (A,1)

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Merge Sort (6)

b) Divide & conquer Approach :-

* Most of the algo. are recursive in nature (i.e., for solving a particular problem, they call themselves repeatedly one or more times). All these algo. follow the Divide & Conquer approach to accomplish a given task.
* In this approach, whole problem is divided into several subproblems. These subproblems are similar to the original problem but smaller in size. All these subproblems are then solved recursively. & then these solutions are combined.
* At each level of recursion the divide & conquer approach follows three steps :-

Step 1:- Divide

The whole problem is divided into several subprob.

Step 2:- Conquer

The subprob. are conquered by solving them recursively, Only if they are small enough to be solved; otherwise step 1 is executed.

Step 3:- Combine

Finally, the solutions obtained by the subproblems are combined to create solution to the original problem.

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

The Merge Sort algo. closely follow the divide & conquer Paradigm. It operates as follows:-

Divide: Divide the n-element seq. to be sorted into two subsequences of $n/2$ elements each.

Conquer:- Sort the two subsequences recursively using Merge Sort.

Combine: Merge the two sorted subsequences to produce the sorted answer.

* The key operation of the mergesort algo. is the merging of two sorted subsequences in combine step.

* To perform merging, we use an auxiliary procedure MERGE(A, p, q, r), where A is an array and p, q & r are indices numbering elements of the array s.t. $p \le q \le r$.

* The procedure assumes that the subarrays A[p ... q] & A[q+1 ... r] are in sorted order. It merges them to form a single sorted subarrays that replaces the current subarray A[p ... r].

Merge (A, p, q, r)

```
n1 <- q - p + 1                 { length of subarrays A[p .. q]
n2 <- r - q                     { length of subarray A[q+1 .. r]
create arrays L[1 ... n1 + 1] & R[1 ... n2 + 1]
for i <- 1 to n1                } copies the subarrays A[p .. q]
    do L[i] <- A[p + i - 1]     }   into L[1 .. n1]
for j <- 1 to n2                } copies the subarrays A[q+1 .. r]
    do R[j] <- A[q + j]         }   into R[1 .. n2]
```

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM


```
8.  L[n1 + 1] ← ∞        } Put Sentinel at the end of L.
9.  R[n2 + 1] ← ∞        } Put Sentinel at the end of R.
10. i ← 1                } Initialization
11. j ← 1                }
12. for K ← p to r
13.     do if L[i] ≤ R[j]
14.         then A[K] ← L[i]
15.              i ← i + 1
16.         else A[K] ← R[j]
17.              j ← j + 1
```

Running time of merge Procedure is $\Theta(n)$. As

* lines 1-3 & 8-11 takes constant time,
* lines 4-7 take $\Theta(n_1 + n_2)$ i.e., $O(n)$ time

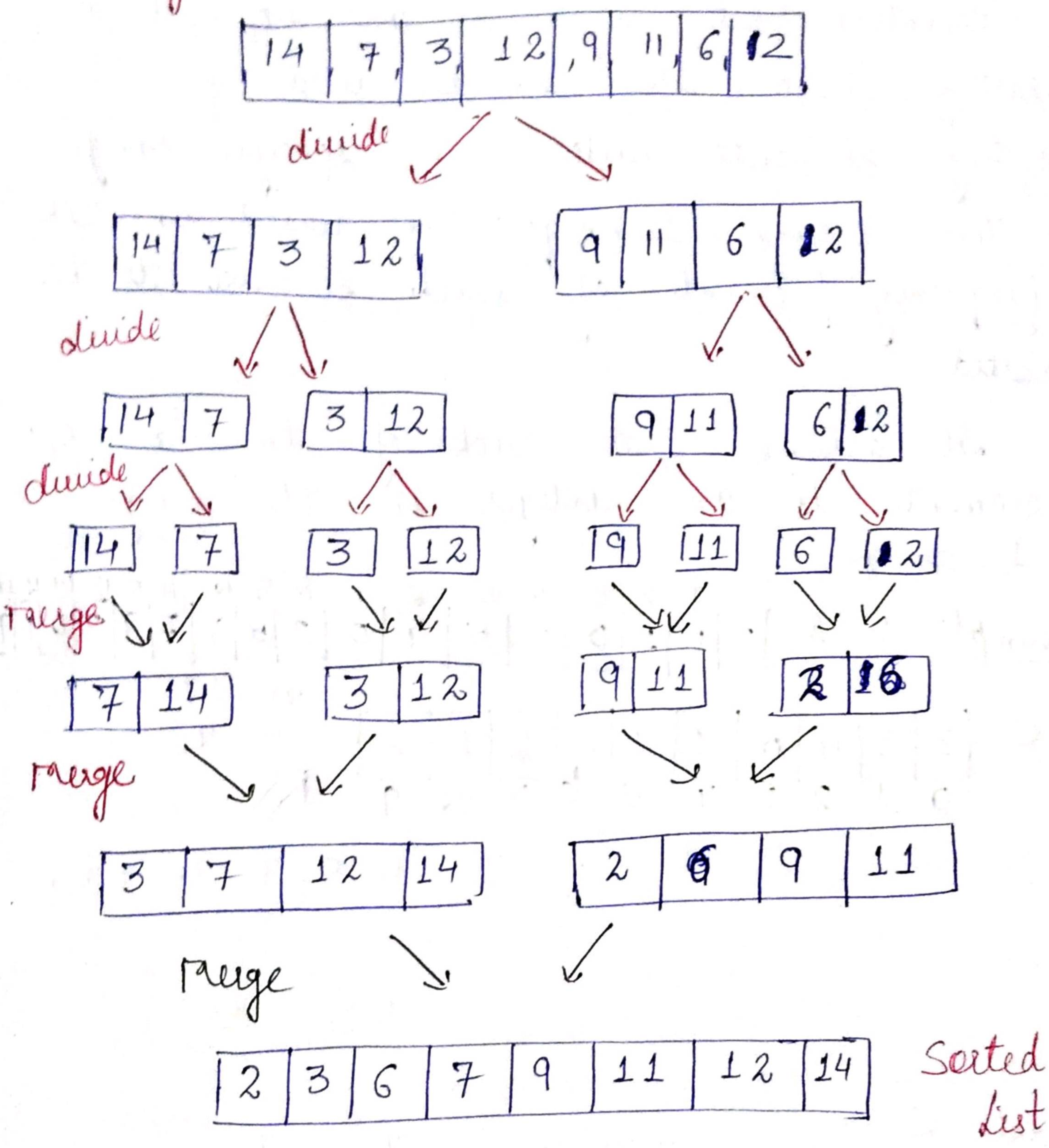
* We can now use the merge Procedure as a subroutine in the mergesort algo. The procedure MergeSort(A, p, r) sorts the elements in the subarray $A[p \ldots r]$.
* If $p \geq r$, the subarray has at most one element & is therefore already sorted. Otherwise, the divide step computes an index q that partition $A[p..r]$ into two subarrays: $A[p..q]$ containing $\lceil n/2 \rceil$ elements & $A[q+1..r]$ containing $\lfloor n/2 \rfloor$ elements.

MergeSort (A, p, r)

```
if p < r
    then q ← ⌊(p + r)/2⌋
        MergeSort (A, p, q)
        MergeSort (A, q+1, r)
```

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Merge Sort Example :-



SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

(8) Counting Sort

* Counting sort algo. is an efficient sorting algo. that can be used for sorting elements within a specific range.
* This sorting technique is based on the frequency / count of each element to be sorted.

* It assumes that each of the 'n' i/p element is an integer in the range 1 to K.

Example :- A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	1	1	0	2	5	4	0	2	8	7	7	9	2	0	1	9

$n = 17$
$K = 9$

C

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

```
for(i=0; i<n; i++)
{
    ++ C[A[i]];
}
```

Update C :-

0	1	2	3	4	5	6	7	8	9
3	6	10	10	11	12	12	14	15	17

```
for (i=1; i<=K; i++)
{
    c[i] = c[i] + c[i-1];
}
```

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Output Array B[ ] :-

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		0 (3)			1 (2)				2 (4)							9 (1)

(1) Start with $i = 16$ the element is 9 (In A[ ])
Now check with updated C, value is 17.

0	1	2	3	4	5	6	7	8	9
3	6	10	10	11	12	12	14	15	17

decrement by 1 i.e; $17-1 = 16$,
store 9 at index 16 in B

(2) decrement i by 1 i.e; $i = 15$, $A[15] = 1$
$C[1] = 6$, decrement by 1 i.e; $6-1 = 5$
store 1 at B[5].

(3) Now $i = 14$, $A[14] = 0$, $C[0] = 3$,
decrement by 1, i.e; $3-1 = 2$
store 0 at B[2]

(4) Now $i = 13$, $A[13] = 2$, $C[2] = 10$
decrement by 1 i.e; $10-1 = 9$
store 2 at B[9]

Till Now C is

0	1	2	3	4	5	6	7	8	9
2	5	9	10	11	12	12	14	15	16

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

we'll repeat the process till we reach $i = 0$ & we'll get the ~~updated~~ sorted array.

```
for (i = n-1; i >= 0; i--)
{
    B[--C[A[i]]] = A[i];
}
```

Counting-Sort (A, B, K)

1. for i = 0 to K
2.   do C[i] = 0
3. for j = 0 to n-1   /* length[A] = n */
4.   do ++C[A[j]];
5. for i = 2 to K
6.   do C[i] = C[i] + C[i-1]
7. for i = n-1 down to 0
8.   B[--C[A[i]]] = A[i];
9.

Q.

0	1	2	3	4	5	6	7	
3,	5,	4,	1,	3,	4,	1,	4	← A

C

0	1	2	3	4	5
0	2	0	2	3	1

C ← updated

0	1	2	3	4	5
0	2	2	4	7	8

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Q :- A = 2, 1, 3, 5, 4, 2, 1, 4, 4, 3, 2, 1, 1, 3, 5

Radix - Sort

* It is the method that many people use when alphabetizing a large list of names. Specifically, the list of names is first sorted according to the first letter of each name.
* For the case of numbers, Least Significant digit is sorted first.

Example :-

* Radix Sort algo. requires the no. of passes which are equal to the no. of digits present in the largest no. among the list of nos.
* if largest no. is a 3-digit no., the list is sorted with 3 passes.

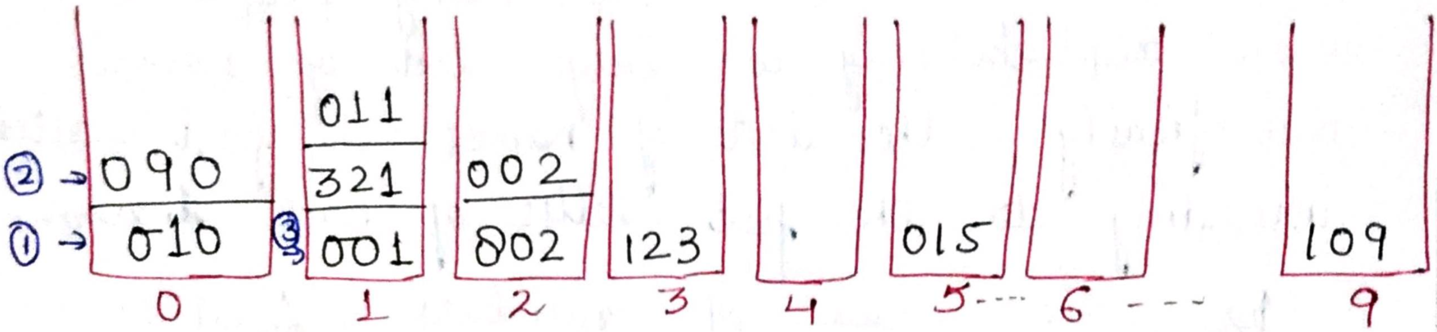
M L	Pass 1	Pass 2	Pass 3
326	690	704	326
453	751	608	435
608	453	326	453
835	704	835	608
751	835	435	690
435	435	751	704
704	326	453	751
690	608	690	835

SBG Study : Download Free Study Material
WWW.SBGSTUDY.COM

Example ②:-

015, 001, 321, 010, 802, 002, 123, 090, 109, 011

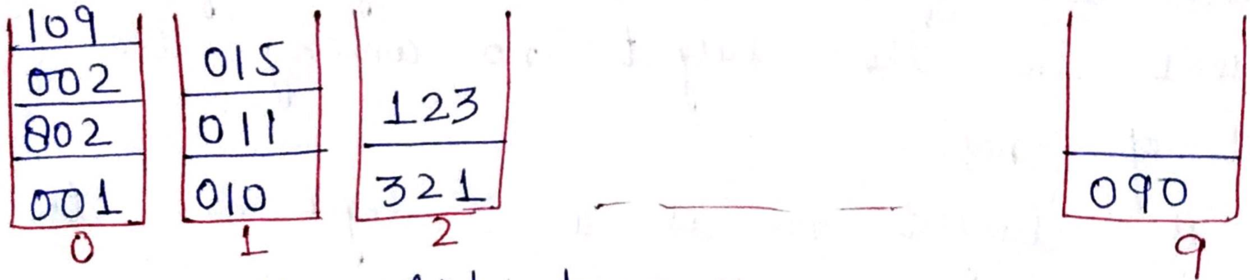
Pass 1:- consider LSB of all nos.



Now the list becomes,

010, 090, 001, 321, 011, 802, 002, 123, 015, 109

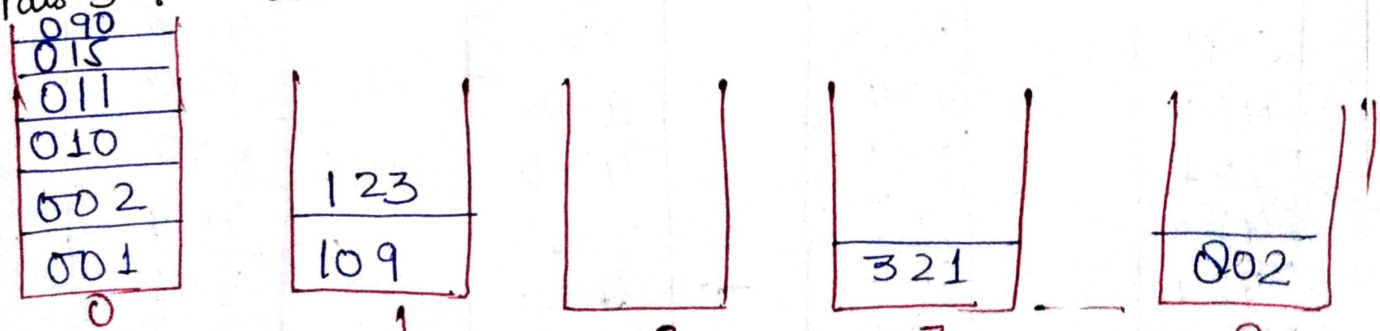
Pass 2:- consider II digit



Now, the list becomes,

001, 802, 002, 109, 010, 011, 015, 321, 123, 090

Pass 3:- consider MSB



o/p:- 1, 2, 10, 11, 15, 90, 109, 123, 321, 802 Sorted