Unit - IV Searching and Sorting

Page No.
Date

Searching

It is a process of finding the element from a given set of elements. If found, display the location of the element else print item not found.

Types of Searching

→ Linear Search
→ Binary Search

* Linear Search

Linear search or sequential search is most simple searching method. The element which to be searched is compared with each element of list one by one. If a match exist, the search is terminated. If end of the list is reached, it mean that the search has failed and the element is not in the list.

Time Complexity

Best Case - $O(1)$ occur when the searched item is present in the first element of array.

Worst case - $O(n)$ occurs when the required element is not present at all.

Average case - Element is found in the middle of array. $O(n/2)$.

Example :-

Consider array A

A	29	75	69	22	2	95
i =	1	2	3	4	5	6

- Let us search for 69

Algorithm :-

Linear Search (A , N, item)

A = array
N = No. of element

1. LOC = -1
2. $i = 1$
3. Repeat while $i \leq N$ and $A[i] \neq item$
   $i = i + 1$
4. If $A[i] = item$ then
   $loc = i$
5. Return loc

Advantage :-
- It is the simplest Known technique
- Element in the list can be in any order.

Disadvantage :-
- Complexity is $O(n)$ in worst case
- Consume large time in case of large number of element

Binary Search

Binary search starts by comparing the target value to the middle element of the array. If the target is equal to the middle element, the search is complete.
If the target is smaller, the search continues in the lower half of the array; if larger in upper half it continues untill target is found or search space is exhausted.

Time Complexity

Best case - O(1), occurs when the target is middle element of the array.

Average case - O(log n), as the search space is halved with each step.

Worst case - O(log n), when the target is at end or not present.

Note :- Element should be in sorted order (Increasing order) and it should be array.

Example :-

Consider Array A of 9 element

A	24	36	39	47	78	87	92	112	156
i =	1	2	3	4	5	6	7	8	9

- Let us search for 87

Algorithm :-

Binary Search (A, N, item)

1. LOC = -1
2. B = 1 , E = N ; N = 9 {B = Begining , E = End}
3. while B ≤ E
4. Mid = [(B + E)/2]
5. If item = A[mid] then
   LOC = mid [Exit loop]
   else if item > A[mid]
   B = mid + 1
   else
   E = mid - 1
6. Return LOC

Advantage :-
- Simple to implement
- Efficient for searching large sorted arrays.

Step 1 :-

$$Mid = \frac{(B + E)}{2}$$

$$= \frac{(1 + 9)}{2}$$

$$= 5$$

24	36	39	47	78	87	92	112	156

B = 1

Mid

E = 9

item > mid

87 > 78

So search continues in upper half of array

B = mid + 1

B = 5 + 1

B = 6

Step 2 :-

$$Mid = \frac{B+E}{2} = \frac{6+9}{2} = 7.5$$

24	36	39	47	78	87	92	112	156

B = 6

mid

E = 9

item < mid

87 < 92

So search continues in lower half of array

E = mid - 1

E = 7 - 1

E = 6

Step 3 :-

$$Mid = \frac{B+E}{2} = \frac{6+6}{2} = 6$$

24	36	39	47	78	87	92	112	156

B = 6
E = 6
mid

item = mid

Disadvantage :-

- Require a sorted array
- May not be best choice for large array.

Difference b/w Linear Search and Binary Search

→ Linear Search

- In linear search input data need not to be in sorted.
- It is also called sequential search.
- Time complexity of linear search O(n).
- Multidimensional array can be used
- It is less complex
- It is very slow process

→ Binary Search

- In Binary search input data need to be in sorted.
- It is also called half - interval search
- Time complexity of Binary search O(log n)
- Only single dimensional is used
- It is more complex.
- It is very fast process.

Sorting

Sorting is a technique of organizing the data. It is a process of arranging the records either in ascending or descending order.

Sorting can be performed on any one or combination of one or more attribute present in each record.

- The sorting is performed according to the key value of each record

Types of Sorting techniques :-

→ Internal Sorting

Internal sorting method is used when small amount of data has to be sorted.

The data to be sorted is stored in the main memory

It can access record randomly.

Ex :- Quick sort, Bubble sort, Insertion Sort, Heap sort, selection sort.

→ External Sorting

External sorting method is used when large amount of data has to be sorted.

The data to be sorted is stored in the main memory as well as secondary memory

It can access record in sequential order.

Ex :- Merge Sort.

Selection Sort

In selection sort, the smallest value among the unsorted element of the array is selected in every pass and inserted into its appropriate position into the array.

first, find the smallest element of array and place it on the first position. Then, find second smallest ~~p~~ element of array and place it on the second position. The process continues untill we get the sorted array.

- In 1st pass, smallest element of array is to be found along with its index position, swap A[0] and A[pos]

- In 2nd pass, second smallest element of array is to be found along with its index position, swap A[1] and A[pos]

- In (n-1)th pass, position of smaller element between A[n-1] and A[n-2] is to be found Then swap, A[pos] and A[n-1].

Then element A[0], A[1], A[2] . . . . A[n-1] is sorted.

Ex :- Consider array with 5 elements

A	55	33	22	11	44
i =	1	2	3	4	5

Pass 1 :-

Smallest element = 11

Swap A[1] and A[4]

11	33	22	55	44
1	2	3	4	5

Pass 2 :-

Smallest element = 22

Swap A[2] and A[3]

22	33	55	44
2	3	4	5

Pass 3 :-

smallest element = 33

No need to swap

33	55	44
3	4	5

Pass 4 :- smallest element = 44 , swap A[5] and A[4]

44	55
4	5

Therefore, Sorted array

11	22	33	44	55
1	2	3	4	5

Algorithm :-

Selection_Sort()

N is number of element

A[] is array of elements

```
1. for i = 1 to N-1
        Min = A[i]
        Loc = i
2.   for j = i+1 to N
3.      If A[J] < Min then
             Min = A[J]
             loc = j
4.    If  loc ≠ i then
          [swap or Interchange element]
5. Exit
```

Time complexity

- $O(n^2)$

* Bubble Sort

Bubble sort is also known as exchange sort. It is simplest sorting algorithm.

- If array contains N element then (N-1) iteration or passes are required to sort the array

- In each pass two adjacent number are compared, if they are out of order then swap these number

- At the end of first pass, highest value is placed at last position. End of second pass, next highest no. is placed at last position and so on.

Algorithms :-

Bubble_sort( )

1. Repeat step 2 and 3 for $i = 1$ to $N-1$
2. set $j = 1$
3. Repeat while $j <= n$
   if A[i] < A[j] then
   interchange a[i] and a[j]
4. Set $j = j + 1$
   [End of inner loop]
   [End of step 1]
5. Exit

Ex:- Consider array with 5 elements

55 33 22 11 44

Pass 1:-

55 33 22 11 44

$55 > 33$ swap

33 55 22 11 44

$55 > 22$ Swap

33 22 55 11 44

$55 > 11$ swap

33 22 11 55 44

$55 > 44$ swap

33 22 11 44 55

↳ Sorted

Pass 2:-

33 22 11 44

$33 > 22$ swap

22 33 11 44

$33 > 11$ swap

22 11 33 44

$33 < 44$ no swap

22 11 33 44

↳ Sorted

Pass 3 :-

22 11 33

22 > 11 swap

11 22 33

22 < 33 no swap

11 22 33

↳ sorted

Pass 4 :-

11 22

11 < 22 no swap

11 22

↳ sorted

At the End of each pass :-

Sorted element :-

A	11	22	33	44	55
i =	1	2	3	4	5

Time Complexity

- $O(n^2)$

Insertion Sort

Insertion sort is one of the best sorting techniques. It is twice as fast as Bubble sort.

In this comparison the value untill all prior elements are less than the compared values is not found. This means that all the previous values are lesser than compared values.

- Insertion sort is good choice for small values and for nearly sorted values.

Algorithm :-

1. Repeat step 2 and 5 for $i = 2$ to N
2. set New = A[i]
3. Repeat step 4 for $j = i - 1$
4. If New < A[j] then

   A[j+1] = A[j]

   Else

   break

   [End of If statement]

   [End of step 3 for statement]
5. A[j+1] = New
6. Exit

Page No.
Date

Ex :- Consider an array of 5 element

7	33	20	11	6

Step 1 :- first value 7 is sorted itself

Step 2 :- Second value 33 is compared to first value 7
Since 33 is greater so no change

Step 3 :- Third element 20 is compared to previous value, which is less than previous value {33 > 20}.
for this 33 is shifted toward right

7	20	33	11	6

Step 4 :- fourth element 11 is compared with its previous element which is less than 20 and 33 and greater than 7 so it is placed between 7 and 20 and also 20 and 33 is shifted toward right

7	11	20	33	6

Step 5 :- Last element 6 is compared with all its previous element.

7 11 20 33 6

33 > 6 shifting

7 11 20 6 33

20 > 6 Shifting

7 11 6 20 33

11 > 6 shifting

7 6 11 20 33

7 > 6 shifting

6 7 11 20 33

All elements are shifted toward right and 6 is placed at first position.

Step 6 :- Finally sorted array :-

A	6	7	11	20	33
i =	1	2	3	4	5

Time Complexity

Best case - $O(n)$
Average case - $O(n^2)$
Worst case - $O(n^2)$

* Merge Sort

Merge sort is easier and faster to sort two smaller arrays than one large array. It follows the principal of "Divide and Conquered".

In this sorting the list is first divided into two halves. The right and left sub list obtained are divided into two sub list untill each sub list contain not more than one element. The sub list containing one element do not require any sorting. After that merge the two sorted sub list to form a combined list it applies untill sorted array achieved.

Ex:- Consider an array of 7 element

13	42	36	20	63	23	12

step 1 :- Divide the combined list into two sub list

13	42	36	20

63	23	12

step 2:- Now divide left sub list into smaller sub list

13	42

36	20

step 3:- Similarly divide the sub list til one element is left in the sub list

13

42

36

20

Step 4 :- Sort the element in their appropriate position and then combined the sub list

13	42	20	36

Step 5 :- Now these two sub list are again merged to give sorted sub list

13	20	36	42

Step 6 :- After sorting left array, same process continues in right array

12	23	63

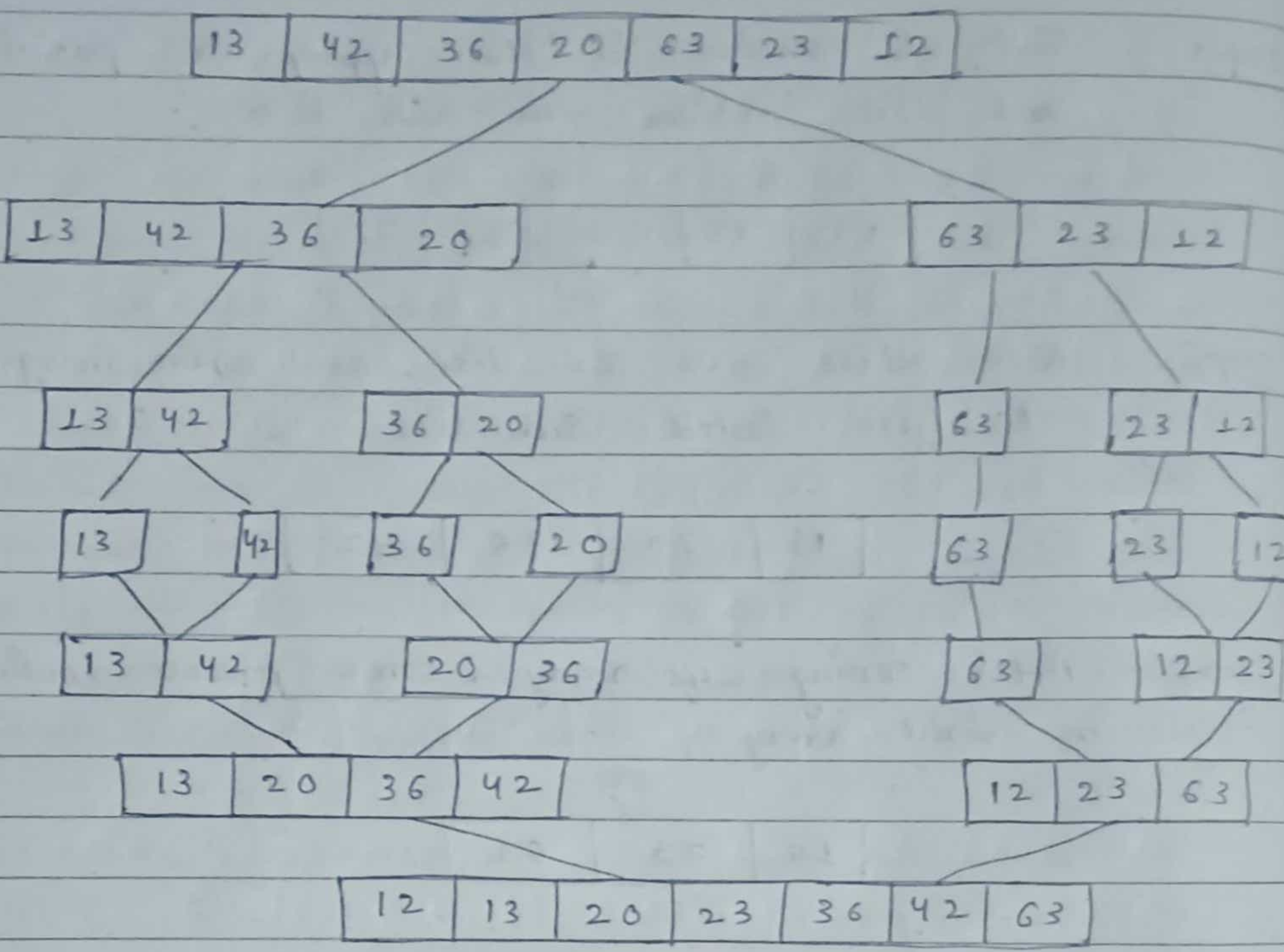
Step 7 :- finally left and right halves of the arrays are merged to the sorted array as follows :-

12	13	20	23	36	42	63

Advantage :-
- It is easy to understand
- It gives better performance

Disadvantage :-
- It requires extra memory space.
- It is slow process.

Time Complexity :- $O(n \log n)$


13 42 36 20 63 23 12
13 42 36 20
63 23 12
13 42
36 20
63
23 12
13
42
36
20
63
23
12
13 42
20 36
63
12 23
13 20 36 42
12 23 63
12 13 20 23 36 42 63

Merge Sort

* Quick Sort

The Quick Sort algorithm follow the principal of divide and conquer. It first picks up the partition element called 'Pivot', which divides the list into two sub list such that all the element in the left sub list are smaller than pivot and all elements in the right sub list are greater than the pivot. This process is repeated untill each sub list containing more than one element.

- The simplest way is to choose the first element as the pivot.

Algorithm :-

1. Select first element of array as Pivot.
2. Initialize i and j to Beg and End element
3. Increment i untill A[i] > Pivot
   Stop
4. Decrement j untill A[j] > Pivot
   Stop
5. If i < j interchange with A[i] and A[j]
6. Repeat step 3, 4, 5 untill i > j
7. Exchange the Pivot element with element placed at j. which is correct place for pivot.

Ex:- An array of 5 element

8	33	6	21	4

Step1 :- Initially the index '0' in the list is choosen as Pivot, Beg and End initialized with index '0' and (n-1).

8	33	6	21	4
↑ Pivot Beg				↑ End

Step2 :- Scanning of element start from end (Right to left)

A[Pivot] > A[End]

8 > 4 (so they are swapped)

4	33	6	21	8

Step3 :- Scanning of element start from begining of list. Since A[Pivot] > A[Beg]. So Beg is increment by one and list remain unchanged

4	33	6	21	8
	↑ Beg			↑ Pivot end

Step 4 :- The element A[Pivot] is smaller than A[Beg]. So they are swapped.

33 > 8 (swapped)

4	8	6	21	33

↑ Pivot Beg (under 8)   ↑ End (under 33)

Step 5 :- Again the list is scanned from right to left. since A[Pivot] is smaller than A[End] so the value of End decreased by one and list remain unchanged.

4	8	6	21	33

↑ Pivot Beg (under 8)   ↑ End (under 21)

Step 6 :- Next the element A[Pivot] is smaller than A[End] so value of End decreased by one and list remain unchanged.

4	8	6	21	33

↑ Pivot Beg (under 8)   ↑ End (under 6)

Step 7 :- A[Pivot] > A[End] they are swapped

4	6	8	21	33

Beg ~~Pivot~~ ~~Beg~~   ~~End~~ Pivot End

Step 8 :- Now list is scanned from left to right since A[Pivot] > A[Beg], value of Beg is increased by one and list remain unchanged.

4	6	8	21	33

↑
Pivot
Beg
End.

Advantage :-

- Fastest sorting techniques among all.
- Recquires small amount of memory

Disadvantage :-

- It is hard to implement.
- Complex method for sorting.

Time Complexity

Best case - O(n log n)
Average case - O(n log n)
Worst case - O(n^2)

* Heap Sort

Heap sort is a comparison - based sorting techniques. It is similar to selection sort.

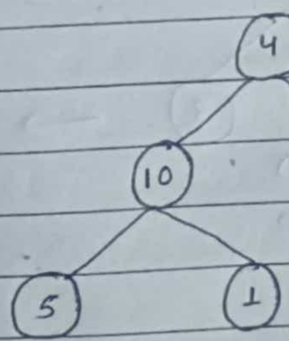
first convert the array into heap data structure using heapify, than one by one delete the ~~s~~root node of the Max-heap and replace it with last node is the heap and then heapify the root of the heap.

Ex:-

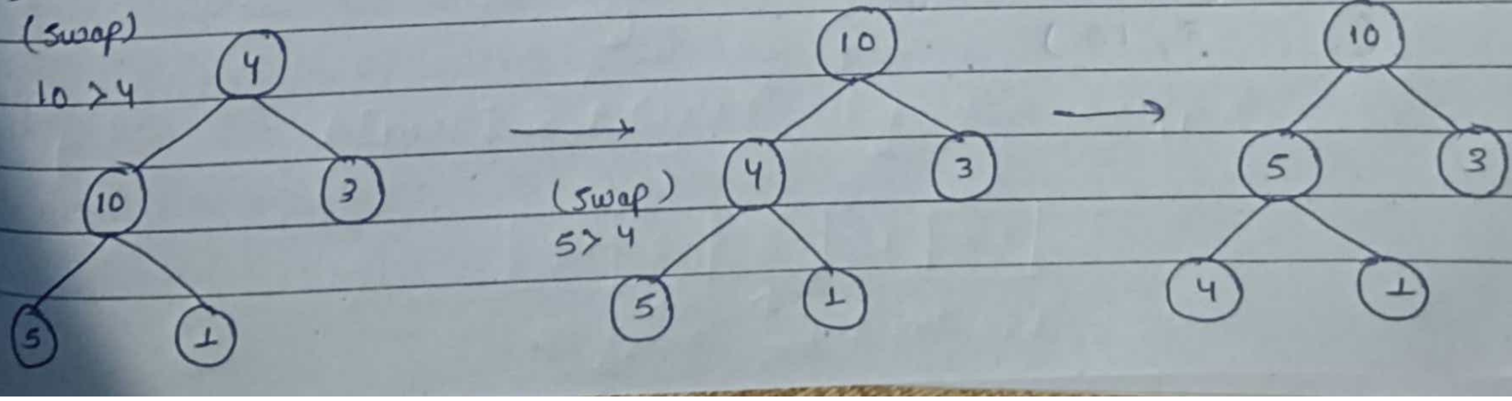
Consider the array

4	10	3	5	1

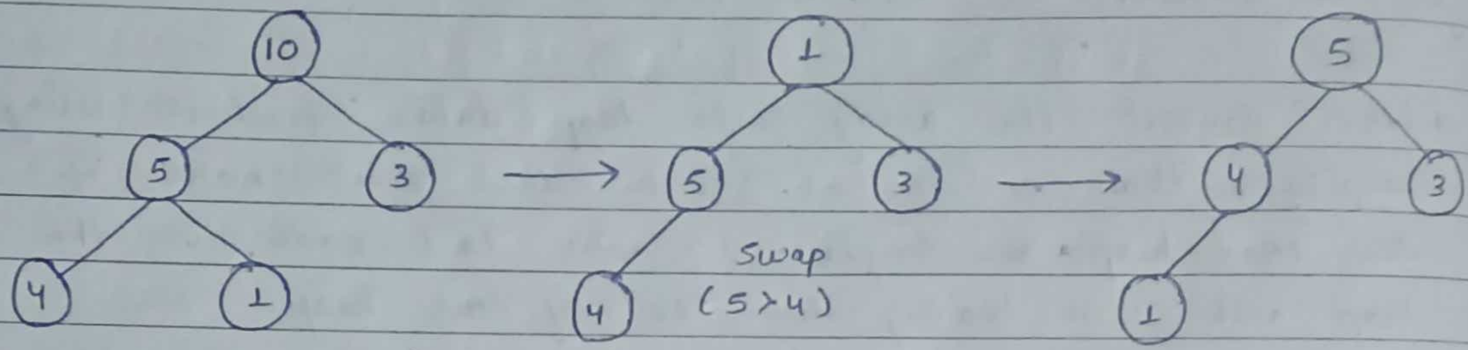
Step 1:- Build complete Binary tree from given array



Step 2:- Max Heapify Constructed Binary tree



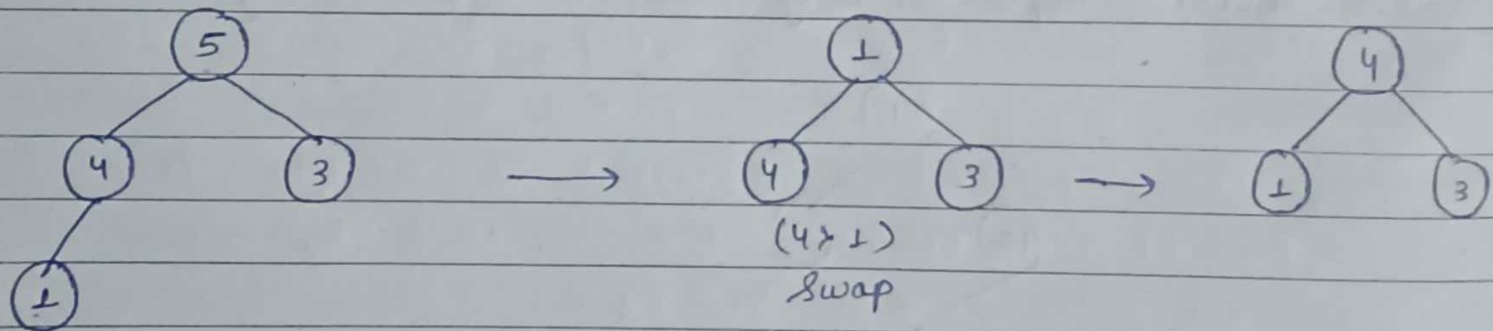
Step 3 :- Remove Maximum from root and max heapify
Delete root element from the max heap and swap it with the last element



Remove max element and insert at final array

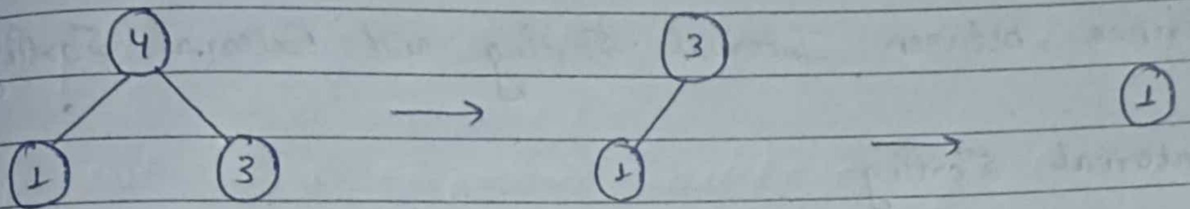
Max heapify remaining tree.

Step 4 :- Remove Next Maximum from root and Max heapify



Remove max element and insert last vacant position of final array
(. . . 5, 10)

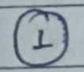
Max heapify remaining tree



Remove max element
and insert in final array
(... 4, 5, 10)

Shift last
element in place of
removed element
(No heapify needed)

Step 5:- Remove Last element and Return Sorted array

 → Sorted array

Remove max element

A =

1	3	4	5	10

Final sorted array

Advantage :-

- It is simpler to understand
- Memory usage is minimum

Disadvantage :-

- Heap sort is unstable
- It is not very efficient

Time Complexity :- O(nlogn)

Difference between Internal Sorting and External Sorting

→ Internal Sorting

- Internal sorting takes place in main memory
- It takes small input
- It does not make use of extra resource
- Internal sorting methods applied to small collection of data.
- Ex- Quick sort , selection sort etc

→ External Sorting

- It take place in both main memory and secondary memory.
- It take as much as large input
- It make use of extra resource
- External Sorting method applied to large collection of data.
- Ex - Merge Sort .