UNIT - IV

Sorting :-

- It is a task of rearranging data in an order, which may be ascending, descending (or) lexicographic (ie., data may be numerical, alphabetical (or) alphanumeric).

- The types of sorting techniques are,

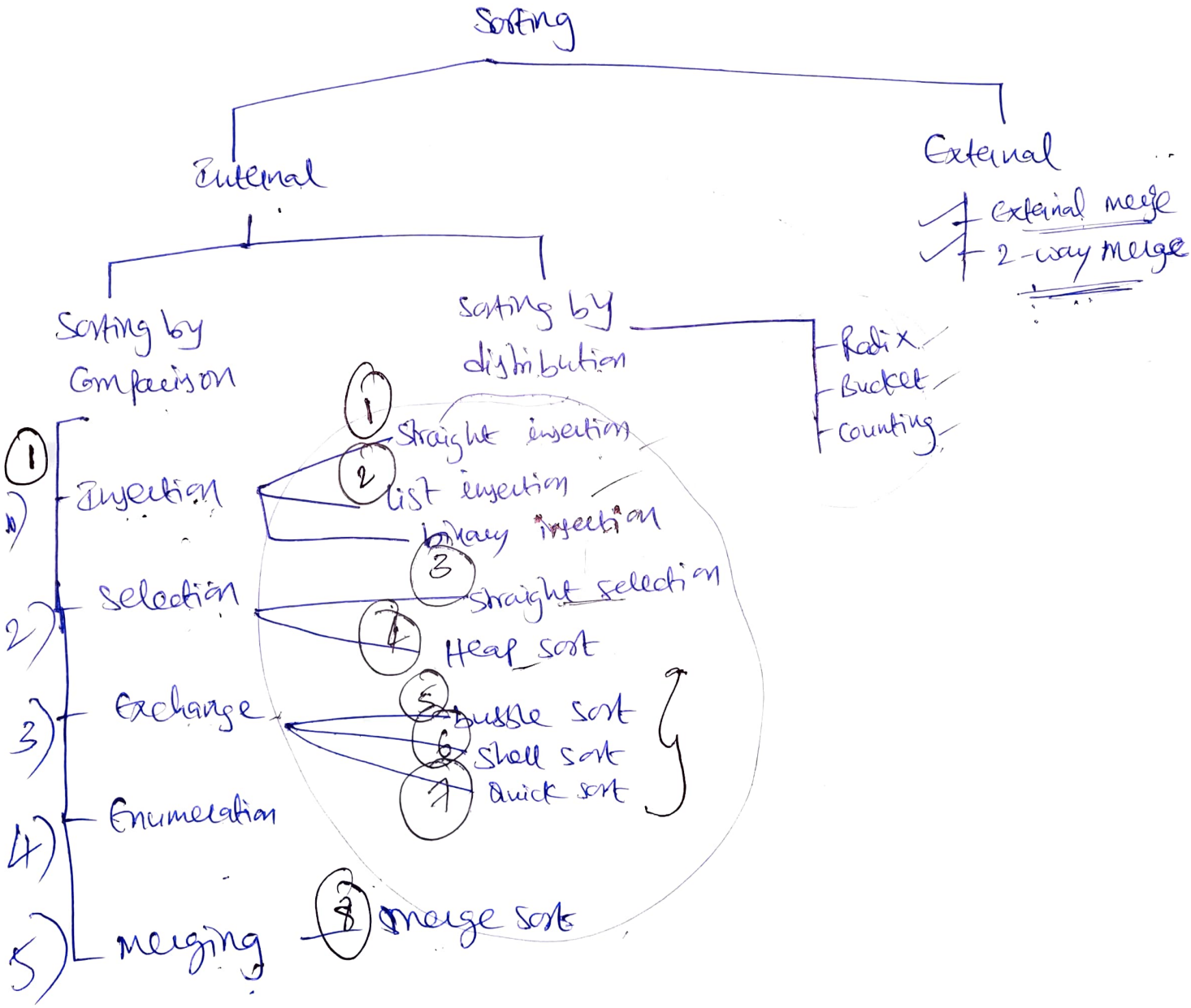


fig:- Sorting Techniques

⇒ Sorting :-

- Sorting refers to separating / arranging elements of a list (or) a set of records in ascending (or) descending order.
- These are classified into different types as sorting by exchange, sorting by insertion, sorting by distribution, sorting by selection etc.
- If the list is very small enough, then internal memory is enough to store the data and is called "internal sorting". And if the list is very huge then external devices are used for storage and is called "External sorting".
- There are many types of sorting techniques as,
  - Bubble sort
  - Insertion sort
  - Selection sort
  - Merge sort
  - Shell sort
  - Quick sort
  - Heap sort
  - Radix sort.

⇒ Sorting by Insertion :-

- This is based on bridge player method, which includes selecting data & inserting it at its own right position.
- This sorting by Insertion includes various types of Insertion Principles and some of them are,
  - i) Straight Insertion Sort
  - ii) List Insertion Sort
  - iii) Binary Insertion Sort.

i) Straight Insertion Sort :-

- This sort is usually called as "insertion sort".
- This need 'N' iterations to sort the list given.
- Also no. of Comparisons also vary in various situations as,
  - a) If i/p list is already sorted order then,
    $$C(n) = n-1$$
  - b) If i/p list is sorted but in reverse order then,
    $$C(n) = \frac{n.(n-1)}{2}$$
  - c) If i/p list is in random order then,
    $$C(n) = \frac{(n-1)(n+4)}{4}$$

Note :- But maximum it takes $(n-1)$ no. of iterations only to sort elem

→ Algorithm :-

- If there are 'j' number of keys in i/p list as, $k_1 \leq k_2 \leq k_3 \leq \dots\dots \leq k_j$. Then the steps to be followed are,

a) Select $(i+1)^{th}$ key k from i/p, which has to be inserted into o/p list.

b) Compare the key k, with k_j, k_{j-1} .... in o/p list until k is inserted b/w k_i & k_{i+1}, ie., $k_i \le k \le k_{i+1}$.

c) move keys $k_{i+1}, k_{i+2} \dots k_j$ in o/p list.

d) put key k into position i+1.

Ex:-

Consider a list of elements and let '5' need to be inserted. Then newly inserted element is inserted at position 'i' as,

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

-> 5 // insert new element //

Iteration (1) -> 5 6 // Start comparing with $(i+1)^{th}$ key & place it in order //.

(2) -> 4 5 6 // repeat same process till it reaches list end //

(3) -> 4 5 6 7

(4) -> 4 5 6 7 9

(5) -> 4 5 6 7 8 9

(6) -> 3 4 5 6 7 8 9

(7) -> 1 3 4 5 6 7 8 9

(8) -> 1 2 3 4 5 6 7 8 9

so, after straight insertion sort the element '5' is inserted in list in an order after (n-1) ie., (9-1) = 8 iterations.

ii) List Insertion sort :-

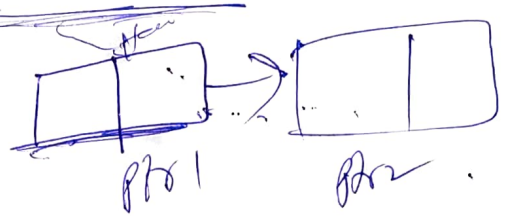
- This follows single linked list data structure.
- Here data is stored in array with SLL structure.
- To insert a node an iteration in list insertion sort these steps are used.

Algorithm :-

Let the key inserted is k. The o/p list is with the header node HEADER.

a) Initially get a new node / allocate memory for node to be inserted.

```
new = GetNode()
header -> link = new
```



b) Then scan the list from header node to find its place,

```
ptr1 = Header,
ptr2 = Header -> link.
if (ptr2 -> Data ≤ k) then
    ptr2 = ptr2 -> link,
    ptr1 = ptr1 -> link;
Else
    end.
```

c) Insert the node with the new key k.

Ex:- suppose we need to insert element 5 into list. Then new element key 'k', '5' is placed at the beginning of list at header.

// << Same as above but represent in LL way //

⇒ Selection Sort :- / Straight selection Sort :-

- selection Sort belongs to Sorting by selection.
- This follows principle of repeated selection of elements satisfying specific criteria.
- The steps involved are,
  - (i) For given unordered List $L = \{k_1, k_2, \ldots k_n\}$, select the minimum key k.
  - (ii) Swap the key k with element in first ~~element~~ position of List L, & perform the same by swapping k & if it is sorted then continue with next position of list.
  - (iii) So, Continue the same process by excluding the sorted elements in a list. This entire process of swapping gets done in (n-i) passes.

Algorithm :-

```
Procedure selection-sort (L, n)
    for i=1 to n-1 do
        min-index = Find_Min(L, i, n);
        swap (L[i], L[min-index]);
    end
end selection-sort.
Procedure Find-Min(L, i, n)
    min-index = i;
    for j = i+1 to n do
        if(L[j] < L[min-index])
            min-index = j;
```

(or)

```
for i=1 to (n-1) do
j = select-min (i,n)
if(i ≠ j) then
    swap (A[i], A[j])
end if
end for
stop.
```

end

return (min-index)

end Find-Min.

swap
temp = x
x = y
y = temp
stop

} algorithm for swap.

Note:-

- The selection sort is not stable.
- Also this requires (or) selection procedure represents totally (n-1) number of passes to sort an array.

Example :-

Let $L = \{70, 15, 84, 101, 55, 32\}$

1 2 3 4 5 6

Pass	List L (Process)	i = 1
1	i=1 70 15 84 101 55 32	
	(Here min element is 15 so, Compare with first indexed element & swap.)	
	15 70 84 101 55 32	
2	15 70 84 101 55 32	i=2
	i=2 (Now, next min ele is 32 · so, Compare 32 with second indexed element & swap)	
	15 70 84 101 55 32	
	15 32 84 101 55 70	
3	i=3 Here next min ele is 55. Compare with next indexed element (exclude already sorted ones).	

15 32 84 101 55 70

15 32 55 101 84 70

---

4 $i=4$

Now next ~~min~~ ele is compared with next indexed element as,

15 32 55 101 84 70

15 32 55 70 84 101

---

5 $j=5$

Now next min ele is 84 & Compare with next index position i=5, as,

15 32 55 70 84 101

Hence, $L = \{15, 32, 55, 70, 84, 101\}$ is the obtained sorted list using selection sort by $(n-1)$ number of passes.

Bubble sort :-

- Bubble sort belongs to sorting by exchange (or) transposition.
- In the process of Bubble sort, pairs of elements are checked.
- The pairs that are out of order are interchanged until the whole list is ordered.
- So, if there are "n" number of elements in the list, then (n-1) passes will be resulted finally.
- At each pass the next largest element of list called "bubbles" are sent to appropriate positions of the list.

- Then smallest element is bubbled from unsorted sublist and moved to sorted sublist.

Algorithm :-

```
Procedure Bubble_sort(L,n)  // Here L[1:n] is an unordered list of elements //
for i=1 to n-1 do           // totally n-1 passes //
    for j=1 to n-i do
        if (L[j]>L[j+1])
            Swap (L[j], L[j+1]);  // Swap Paircoise elements //
    end
end                         // Send next longest element (or) bubble to last position //
end Bubble_sort
```

Example:- → The list is taken as pass(1) with 1..10

Let, $L = \{92, 78, 34, 23, 56, 90, 17, 52, 67, 81, 18\}$

(Pass) (i)	j	List L at pass i
2	1:9	{78, 92, 34, 23, 56, 90, 17, 52, 67, 81, 18}
		{78, 34, 92, 23, 56, 90, 17, 52, 67, 81, 18}
		{78, 34, 23, 92, 56, 90, 17, 52, 67, 81, 18}
		{78, 34, 23, 56, 92, 90, 17, 52, 67, 81, 18}
		{78, 34, 23, 56, 90, 92, 17, 52, 67, 81, 18}
		{78, 34, 23, 56, 90, 17, 92, 52, 67, 81, 18}
		{78, 34, 23, 56, 90, 17, 52, 92, 67, 81, 18}
		{78, 34, 23, 56, 90, 17, 52, 67, 92, 81, 18}
		{78, 34, 23, 56, 90, 17, 52, 67, 81, 92, 18}
		{78, 34, 23, 56, 90, 17, 52, 67, 81, 18, 92}
3	1:8	{34, 78, 23, 56, 90, 17, 52, 67, 81, 18, 92}
		{34, 23, 78, 56, 90, 17, 52, 67, 81, 18, 92}
		{34, 23, 56, 78, 90, 17, 52, 67, 81, 18, 92}
		{34, 23, 56, 78, 17, 90, 52, 67, 81, 18, 92}
		{34, 23, 56, 78, 17, 52, 90, 67, 81, 18, 92}
		{34, 23, 56, 78, 17, 52, 67, 90, 81, 18, 92}
		{34, 23, 56, 78, 17, 52, 67, 81, 90, 18, 92}
		{34, 23, 56, 78, 17, 52, 67, 81, 18, 90, 92}
4	1:7	{23, 34, 56, 78, 17, 52, 67, 81, 18, 90, 92}
		{23, 34, 56, 78, 17, 52, 67, 81, 18, 90, 92}
		{23, 34, 56, 17, 78, 52, 67, 81, 18, 90, 92}

		{23,34,56,17,52,78,67,81,18,90,92}
		{23,34,56,17,52,67,78,81,18,90,92}
		{23,34,56,17,52,67,78,18,81,90,92}
5	1:6	{23,34,17,56,52,67,78,18,81,90,92}
		{23,34,17,52,56,67,78,18,81,90,92}
		{23,34,17,52,56,67,18,78,81,90,92}
		{23,34,17,52,56,67,18,78,81,90,92}
6	1:5	{23,17,34,52,56,18,67,78,81,90,92}
7	1:4	{17,23,34,52,56,18,67,78,81,90,92}
		{17,23,34,52,18,56,67,78,81,90,92}
8	1:3	{17,23,34,18,52,56,67,78,81,90,92}
9	1:2	{17,23,18,34,52,56,67,78,81,90,92}
10	1:1	{17,18,23,34,52,56,67,78,81,90,92}

- Hence, by this bubble sort all the bubbles (ie, elements that are largest in list) are sent to the extreme end of list.
- This entire process of sorting is performed with (n-1) number of passes.

⇒ Shell sort :-

-> This is an array A with n elements sorted successively whose entries are intermingled in whole array, called increments.

-> This shell sort can be sorted by,

$gap = floor(N/2)$ and again for next passes it is calculated as,

$gap_1 = floor(gap/2)$

$gap_2 = floor(gap_1/2)$

$\vdots$

$gap_n = floor(gap_n/2)$

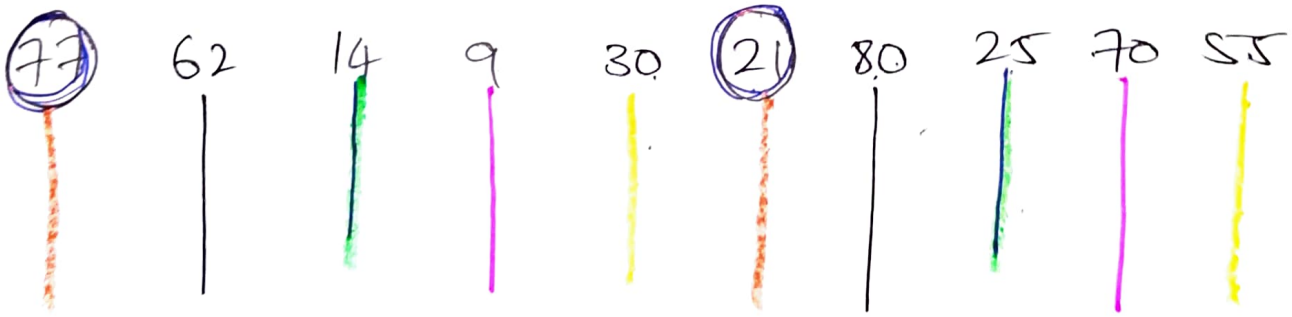
Ex:-

77 62 14 9 30 21 80 25 70 55

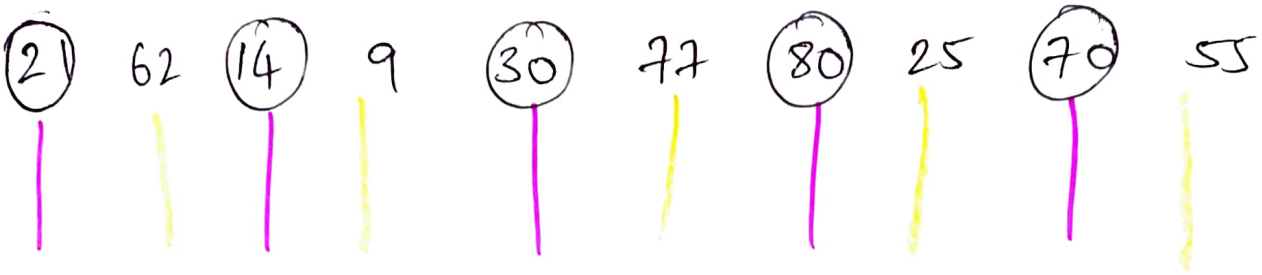
Sort out all the above 10 elements using Shell sort.

$n = 10$.,

$gap = floor(10/2) = 5$



Now Compare alike group elements {77, 21}, {62, 80}, {14, 25}, {9, 70}, {30, 55} .... & exchange their positions by placing minimum element first.



Now, $gap_1 = floor(5/2) = 2$

Now again exchange & rearrange the same group elements. we get:,

Now,

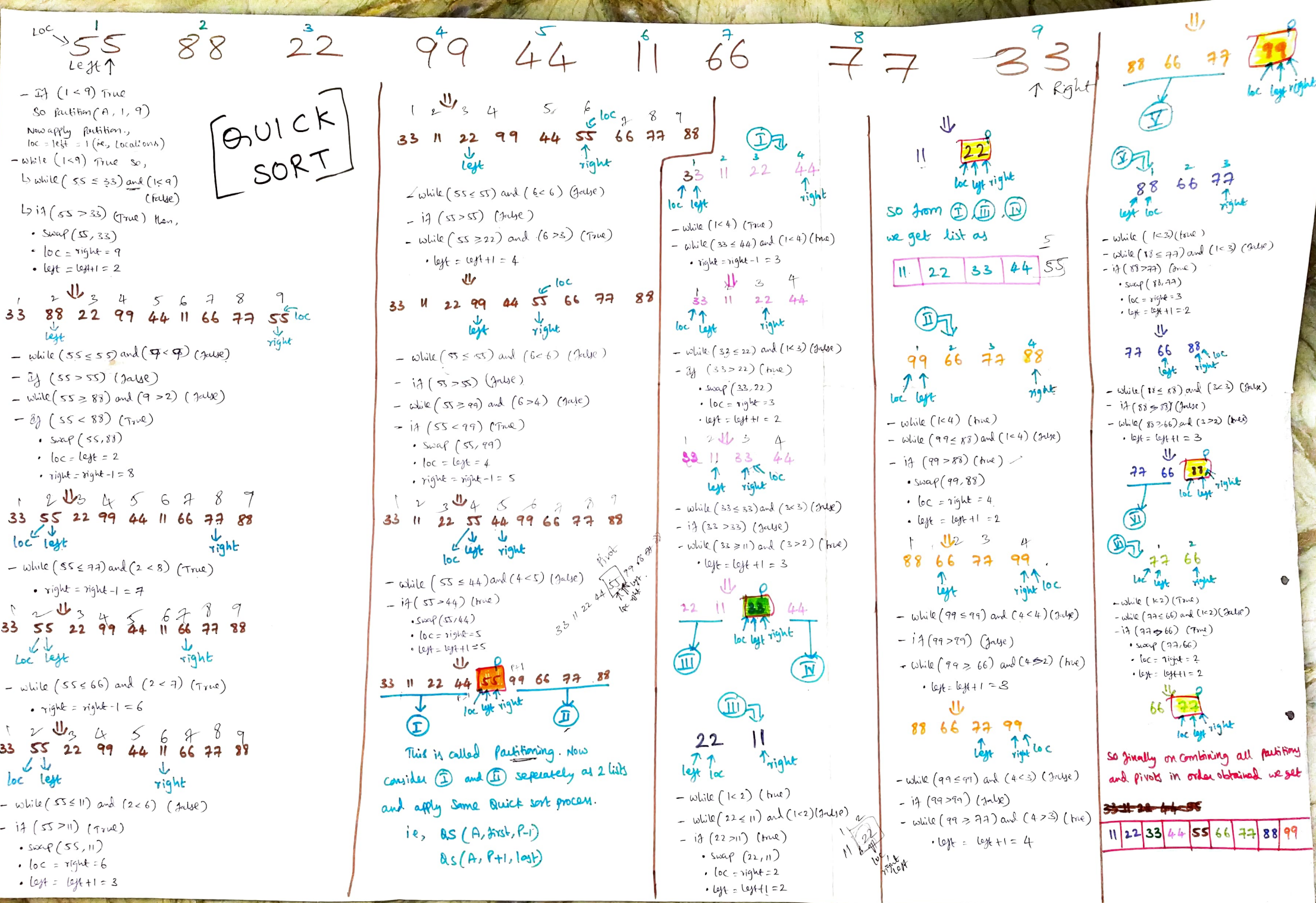
$gap_2 = floor(2/2) = 1$

Here in the next final iteration elements are grouped as 1 (ie, individually). so now Compare, exchange & rearrange group of elements we get,

9 14 21 25 30 55 62 70 77 80

So, finally the given elements have been sorted using shell sort.

QUICK SORT


loc
left
55 88 22 99 44 11 66 77 33
right
– If (1 < 9) True
So Partition(A, 1, 9)
Now apply Partition.,
loc = left = 1 (ie, locations)
– while (1<9) True so,
↳ while (55 ≤ 33) and (1≤9) (False)
↳ if (55 > 33) (True) then,
• Swap (55, 33)
• loc = right = 9
• left = left+1 = 2
33 88 22 99 44 11 66 77 55
– while (55 ≤ 55) and (9<9) (false)
– if (55 > 55) (false)
– while (55 ≥ 88) and (9 > 2) (false)
– if (55 < 88) (True)
• Swap (55, 88)
• loc = left = 2
• right = right-1 = 8
33 55 22 99 44 11 66 77 88
– while (55 ≤ 77) and (2 < 8) (True)
• right = right-1 = 7
33 55 22 99 44 11 66 77 88
– while (55 ≤ 66) and (2 < 7) (True)
• right = right-1 = 6
33 55 22 99 44 11 66 77 88
– while (55 ≤ 11) and (2<6) (false)
– if (55 > 11) (True)
• Swap (55, 11)
• loc = right = 6
• left = left+1 = 3
33 11 22 99 44 55 66 77 88
– while (55 ≤ 55) and (6<6) (false)
– if (55 > 55) (false)
– while (55 ≥ 22) and (6 > 3) (True)
• left = left+1 = 4
33 11 22 99 44 55 66 77 88
– while (55 ≤ 55) and (6<6) (false)
– if (55 > 55) (false)
– while (55 ≥ 99) and (6 > 4) (false)
– if (55 < 99) (True)
• Swap (55, 99)
• loc = left = 4
• right = right-1 = 5
33 11 22 55 44 99 66 77 88
– while (55 ≤ 44) and (4<5) (false)
– if (55 > 44) (true)
• Swap (55, 44)
• loc = right = 5
• left = left+1 = 5
33 11 22 44 55 99 66 77 88
Pivot
33 11 22 44 55 99 66 77 88
This is called Partitioning. Now consider (I) and (II) seperately as 2 lists and apply same Quick sort process.
ie, QS (A, first, P-1)
QS (A, P+1, last)
33 11 22 44
– while (1<4) (True)
– while (33 ≤ 44) and (1<4) (true)
• right = right-1 = 3
33 11 22 44
– while (33 ≤ 22) and (1<3) (false)
– if (33 > 22) (true)
• Swap (33, 22)
• loc = right = 3
• left = left+1 = 2
22 11 33 44
– while (33 ≤ 33) and (3<3) (false)
– if (33 > 33) (false)
– while (33 ≥ 11) and (3>2) (true)
• left = left+1 = 3
22 11 33 44
22 11
– while (1<2) (true)
– while (22 ≤ 11) and (1<2) (false)
– if (22 > 11) (true)
• Swap (22, 11)
• loc = right = 2
• left = left+1 = 2
11 22
so from (I) (III) (IV)
we get list as
11 22 33 44 55
99 66 77 88
– while (1<4) (true)
– while (99 ≤ 88) and (1<4) (false)
– if (99 > 88) (true)
• Swap (99, 88)
• loc = right = 4
• left = left+1 = 2
88 66 77 99
– while (99 ≤ 99) and (4<4) (false)
– if (99 > 99) (false)
– while (99 ≥ 66) and (4≤2) (true)
• left = left+1 = 3
88 66 77 99
– while (99 ≤ 77) and (4<3) (false)
– if (99 > 99) (false)
– while (99 ≥ 77) and (4>3) (true)
• left = left+1 = 4
88 66 77 99
– while (1<3) (true)
– while (88 ≤ 77) and (1<3) (false)
– if (88 > 77) (true)
• Swap (88, 77)
• loc = right = 3
• left = left+1 = 2
77 66 88
– while (88 ≤ 88) and (3<3) (false)
– if (88 > 88) (false)
– while (88 > 66) and (3>2) (true)
• left = left+1 = 3
77 66
– while (1<2) (True)
– while (77 ≤ 66) and (1<2) (false)
– if (77 > 66) (True)
• Swap (77, 66)
• loc = right = 2
• left = left+1 = 2
So finally on combining all partitions and pivots in order obtained we get
11 22 33 44 55 66 77 88 99

⇒ Quick sort :-

- This is developed by C.A.R. Hoare in 1962.
- This belongs to Sorting by Exchange (or) transposition.
- This follows process of Partitioning unordered list into sublists based on pivot element (or) LOC
- The pivot element determines appropriate position in sorted list.

Algorithm :-

QS (A, 1, 9)

```
Quicksort (A, first, last)
if (first < last) then
    P = Partition (A, first, last)
    Quicksort (A, first, P-1)
    Quicksort (A, P+1, last)
return.
```

Quicksort

position / pivot

a[first] left
last right
x[loc]

```
loc = left    // leftmost is pivot element //
① while (left < right) do
    i) while (A[loc] ≤ A[right]) and (loc < right) do
           right = right - 1
       end while
    ii) if (A[loc] > A[right]) then
           swap (A[loc], A[right])
           loc = right
           left = left + 1
       end if
```

Partition

Q52

```
while (A[loc] ≥ A[left]) and (loc > left) do
    left = left + 1
Endwhile
if (A[loc] < A[left]) then
    Swap(A[loc], A[left])
    loc = left
    right = right - 1
end if
end while
return (loc)
stop
```

(ii)

(iv)

(i)

10.7.3 Merge Sort

So far we have discussed the merging operation. In this section, we shall discuss the sorting by such a merging operation. All sorting method based on merging can be divided into two broad categories:

- Internal merge sort and
- External merge sort

Divide: Partition the list midway, that is, at $\left\lfloor \frac{l+r}{2} \right\rfloor$ into two sub lists with $\frac{n}{2}$ elements in each, if n is even or $\left\lceil \frac{n}{2} \right\rceil$ and $\left\lceil \frac{n}{2} \right\rceil - 1$ elements if n is odd.

Conquer: Sort the two lists recursively using the merge sort.

Combine: Merge the sorted sub lists to obtain the sorted output.

The divide-and-conquer strategy used in the merge sort is illustrated with a schematic diagram as shown in Figure 10.56.

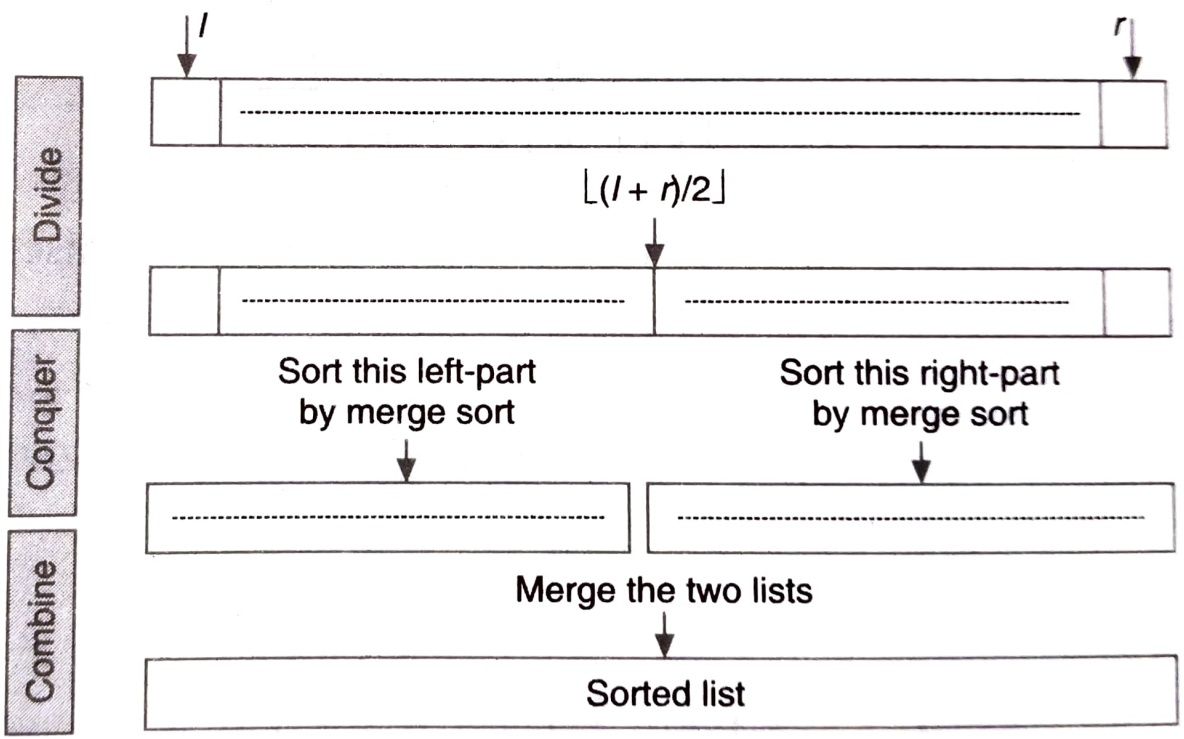


Figure 10.56 Divide-and-conquer strategy in the internal merge sort.

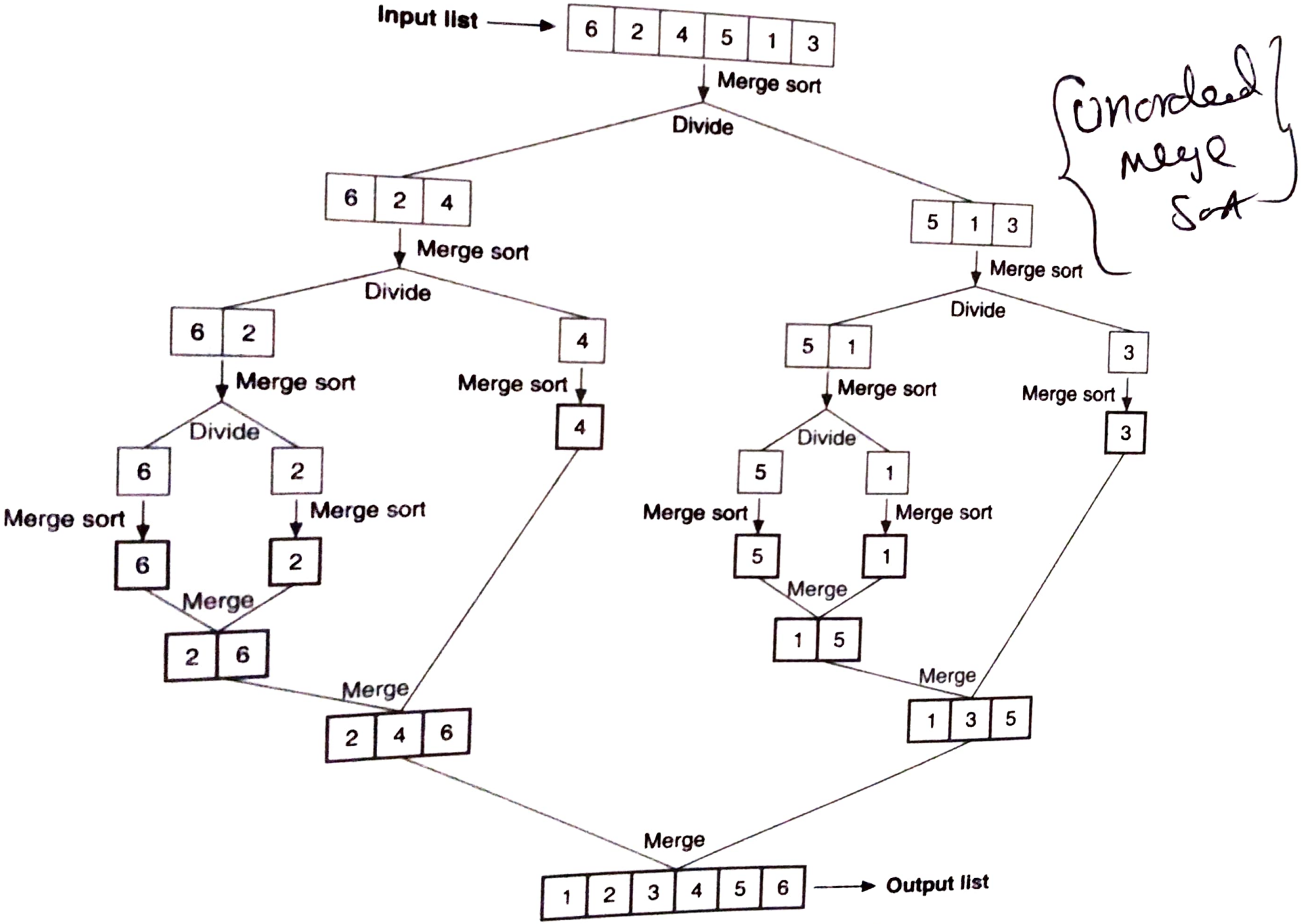


Figure 10.57 Illustration of the merge sort.

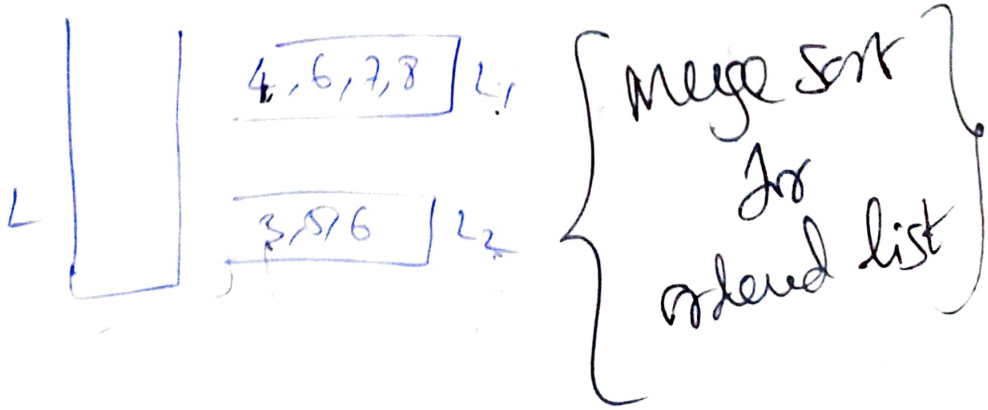
Algorithm :-

```
Algorithm Mergesort(low, high)
{
  if(low < high) then
  {
    mid := ⌈(low + high)/2⌉;
    mergesort(low, mid);
    mergesort(mid+1, high);
    merge(low, mid, high);
  }
}
Algorithm merge(low, mid, high)
{
  h := low; i := high; j := mid+1;
  while((h <= mid) and (j <= high)) do
  {
    if(a[h] ≤ a[j]) then
    {
      b[i] := a[h];
      h := h+1;
    }
    else
    {
```

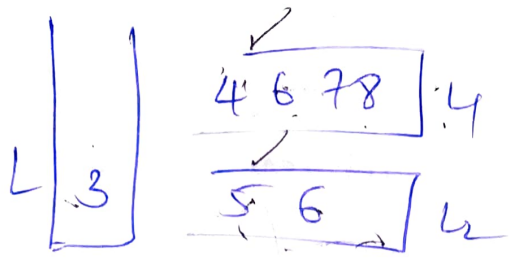
```
        b[i] := a[j];
        j := j+1;
      }
      i := i+1;
    }
    if (h > mid) then
    for( k := j to high do
    {
      b[i] := a[k];
      i := i+1;
    }
  Else
    for k := h to mid do
    {
      b[i] := a[k];
      i := i+1;
    }
    for k := low to high do
      a[k] := b[k];
  }
```

Example :-

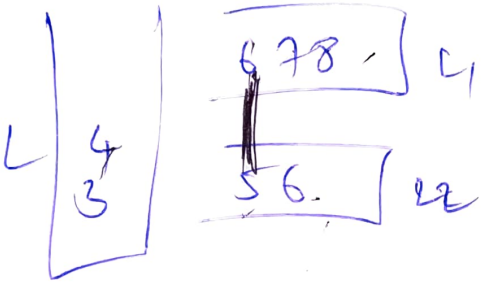
Consider two lists $L_1 = \{4,6,7,8\}$ & $L_2 = \{3,5,6\}$



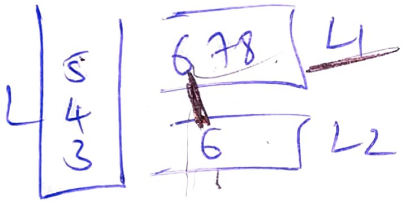
Now Compare 4, 3.



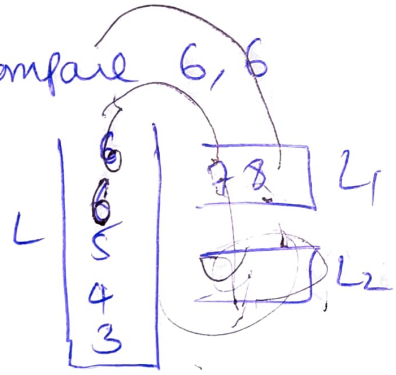
Compare 4, 5



Compare 6, 5



Compare 6, 6



L2 is exhausted. So drop all remaining elements of L1 to L.