7. SORTING

SORTING

It refers to the operation of arranging data in some given order such as increasing or decreasing. There are two types of sorting namely: internal and external sorting.

Internal sorting algorithms are applied where entire collection of data is to be sorted is small enough and complete data is in main memory. For eg. insertion sort, bubble sort, shell sort, quick sort and merge sort.

External sorting methods are applied to the larger data that exists on secondary storage device as they can accomodate in main memory. In this case read and write access time are major measure factor in determining the performance of external sorting algorithm.

1. INSERTION SORT

Insertion sort is one of the simpliest sorting algorithm. It requires n-1 steps or passes to sort a list and n data items. It is implemented by inserting a particular element at appropriate position. The first iteration starts with comparision of 1st element with the 0th element.

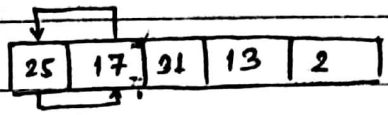
In the 2nd iteration, the 2nd element is compared with 0th and 1st element. In General, on every iteration, an element is compared with all elements before sorting. During comparision, if it is found that, a suitable position to be inserted, then the space is created for it by (shifting) the other elements one position to the

right and inserting the element at the suitable position. This procedure is repeated for all the element in the array.

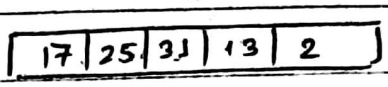
Eg. Using insertion sort, sort the following numbers:
25, 17, 31, 13, 2

Sol:

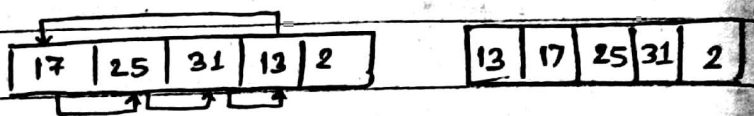
1st iteration



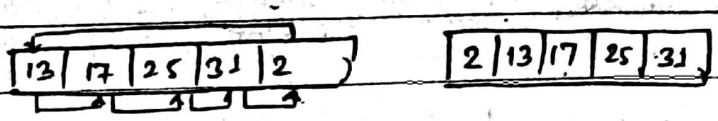
2nd iteration



3rd iteration



4th iteration



Q. Sort the following insertion sort.

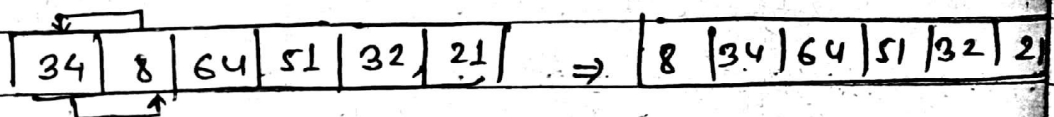
i) 34, 8, 64, 51, 32, 21

ii) 77, 33, 44, 11, 88, 22, 66, 55

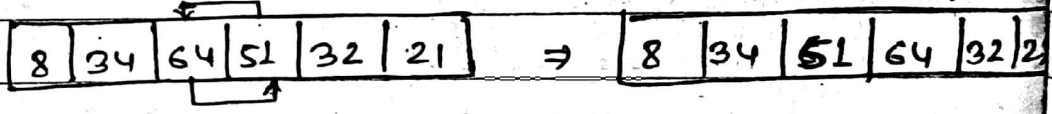
iii) 25, 57, 48, 37, 12, 92, 86, 33

i) 34 8 64 51, 32, 21

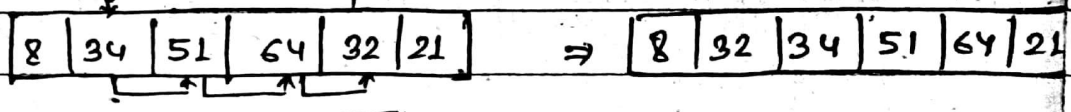
1st iteration



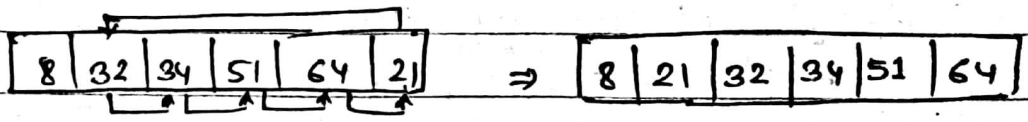
2nd iteration



3rd iteration

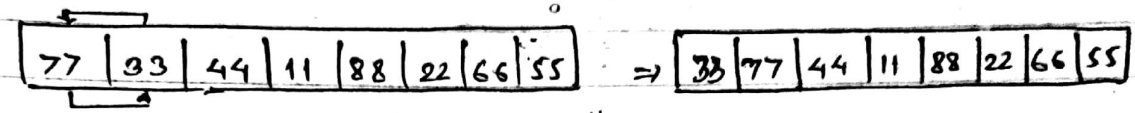


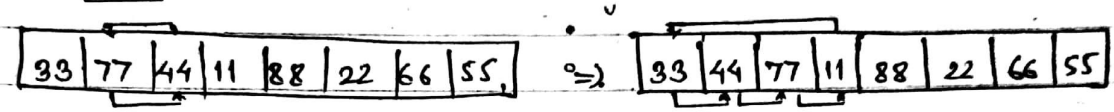
4th iteration

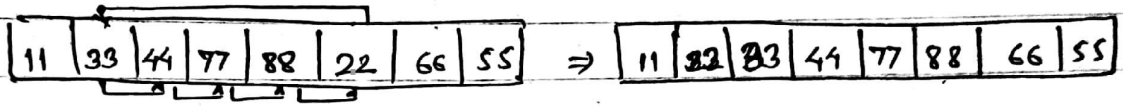


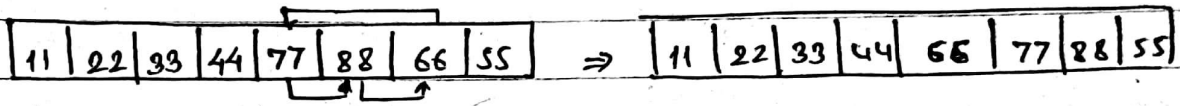
3

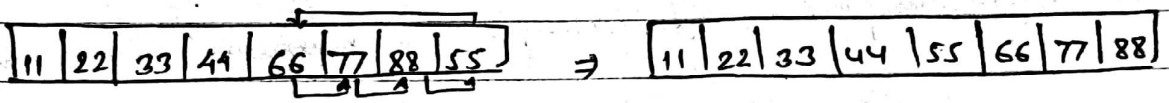
(ii) 77, 33, 44, 11, 88, 22, 66, 55

1st loop 

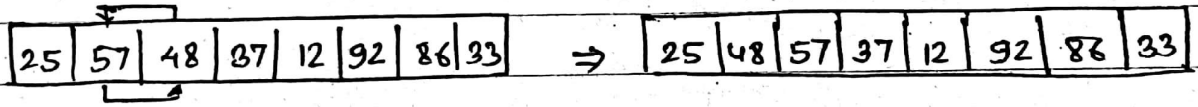
2nd loop 

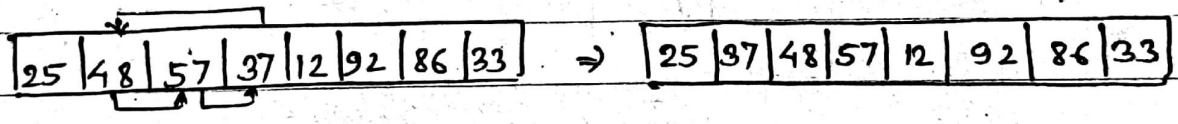
3rd loop 

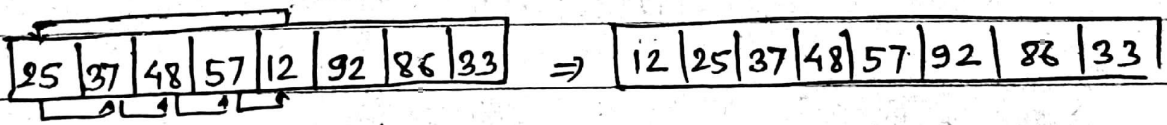
4th loop 

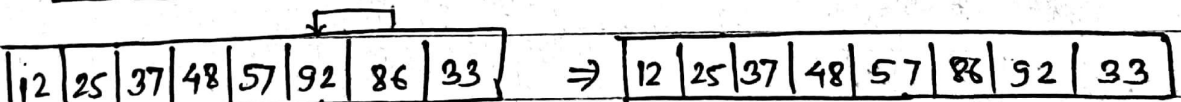
5th loop 

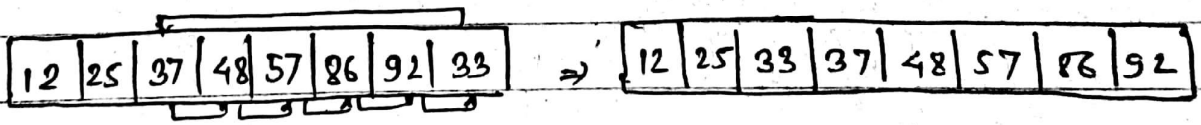
(iii) 25, 57, 48, 37, 12, 92, 86, 33

1st loop 

2nd loop 

3rd loop 

4th loop 

5th loop 

2. SELECTION SORT

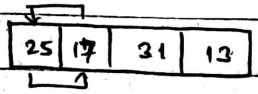
→ The 0th element (1st position) is compared with all other elements. If the 0th° element is found to be greater than the compared, then they are interchanged.

→ After the 1st iteration, ~~the smallest element is placed~~, the smallest element is placed at the 1st position.

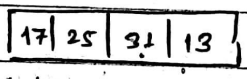
→ This process is repeated for 2nd no. and so on.

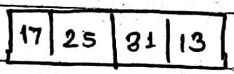
Eg. Sort the following using selection sort. 25, 17, 31, 13

Soln: 1st iteration:

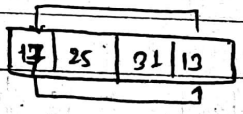


25 > 17

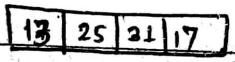




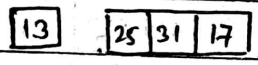
17 < 31



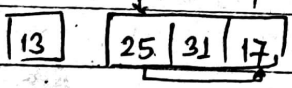
17 > 13



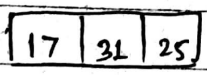
2nd iteration:



25 < 31

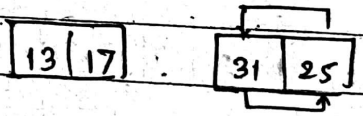


25 > 17

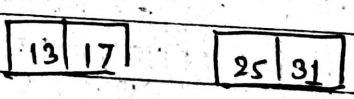


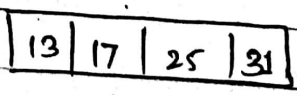
31 2

3rd iteration



31 > 25





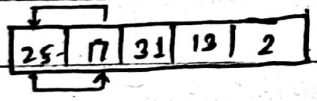
3. BUBBLE SORT (Suitable for sorting small list only)

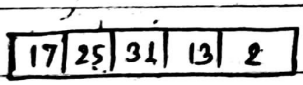
- In bubble sort each element is compared with its adjacent element.
- → If the 1st element is larger than 2nd one then the position of the element is interchanged, otherwise it is not changed.
- → Then the next element is compared to its adjacent element and the process is same and repeated for all elements in the array. At the completion, the last element contains the biggest number.
- → 2nd iteration is done upto the last to one step, and (last -1) contains 2nd large element. This process is repeated and the no will be sorted in ascending order.

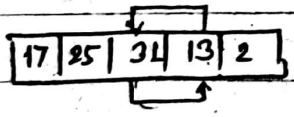
Eg. Sort the following using bubble sort 25, 17, 31, 13, 2

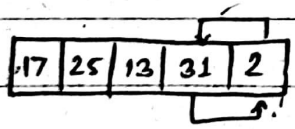
Sol^n:

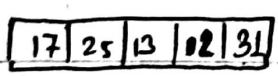
1st iteration:



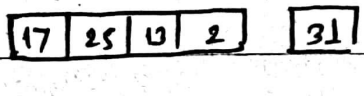


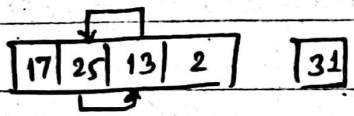


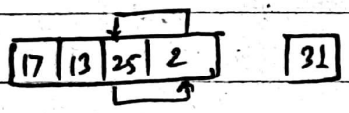


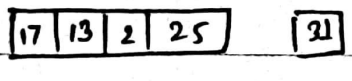


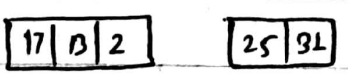
2nd iteration.



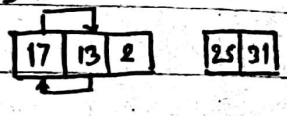


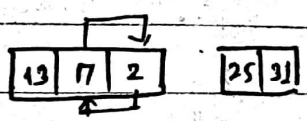


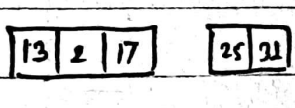


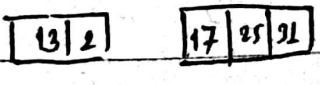


3rd iteration

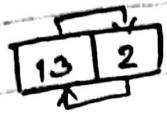


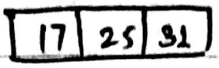


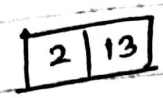


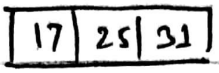


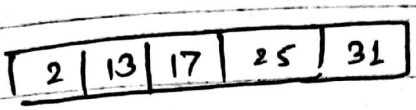
4th iteration:











4. MERGE SORT

It means combining two sorted list. Merge sort is a sorting technique which divides the array into sub array of size two and merge the adjacent pair. We have approximately $\frac{N}{2}$ array of size. This process is repeated until there is only one array remaining of size N.

Sort the following element using merge sort.

85, 76, 46, 92, 30, 41, 12, 19, 93, 3, 50, 51

1. First merge two numbers in pair and sorted in ascending order.

(85, 76), (46, 92), (30, 41), (12, 19), (93, 3), (50, 51)

⇒ 76, 85, 46, 92, 30, 41, 12, 19, 3, 93, 50, 51

2. Merge each pair to obtain the list of sorted element.

(76, 85, 46, 92), (30, 41, 12, 19), (3, 93, 50, 51)

7

⇒ 46, 76, 85, 92 , 12, 19, 30, 41 , 3, 50, 51, 92

3. Again merge two sub array to get the sorted list

46, 76, 85, 92, 12, 19, 30, 41 , 3, 50, 51, 93

⇒ 12, 19, 30, 41, 46, 76, 85, 92 , 3, 50, 51, 93

4. Merge the two sorted sub-array to obtain single sorted array.

12, 19, 30, 41, 46, 76, 85, 92, 3, 50, 51, 93

⇒ 3, 12, 19, 30, 41, 46, 50, 51, 76, 85, 92, 93

5. BUCKET (RADIX) SORT

This sorting technique can be used to sort only positive numbers. Suppose we have a list of positive numbers $a_1, a_2, a_3, \ldots\ldots a_n$, and suppose that all these elements are smaller than positive number 'm'. Now construct an array count [] of size m and initialized all array elements as zero. Therefore the array count has m cell of buckets which are initialized empty.

Algorithm:

1. In the first pass the unit digit are stored in the bucket. The numbers are collected by bucket from bucket 9 to bucket 0.

8

2. In second pass, the tenth digit are stored in the bucket ie. tenth place are sorted into the bucket.

3. In the third pass, the hundredth place digit are sorted into the bucket and the number are collected in sorted form.

Sort the numbers using bucket sort :-
499, 283, 491, 353, 726, 315, 125, 278, 329

1st Pass	0	1	2	3	4	5	6	7	8	9
Input										
499										499
283				283						
491		491								
353				353						
726							726			
315						315				
125						125				
278									278	
329										329

⇒ 491, 283, 353, 315, 125, 726, 278, 499, 329

9

2nd pass

Input	0	1	2	3	4	5	6	7	8	9
491										491
283									283	
353						353				
315		315								
125			125							
726			726							
278								278		
499										499
329			329							

⇒ 315, 125, 726, 329, 353, 278, 283, 491, 499

3rd pass

	0	1	2	3	4	5	6	7	8	9
315				315						
125		125								
726								726		
329				329						
353				353						
278			278							
283			283							
491					491					
499					499					

Then the numbers in ascending order is,

125, 278, 283, 315, 329, 353, 491, 499, 726

Shell sort

→ It is an improved version of sto insertion sort in which diminishing pass are used to sort the list.

→ Compare elements that are distant apart rather than adjacent

→ we start by comparing elements that are at a certain distance apart. So, if there are N element then we start a value gap < N

i.e gap = floor(N/2)

where N = no. of element in array.

→ In each pass, we keep reducing the value of gap till we reach the last pass when gap is 1.

→ In last pass, shell sort is like insertion sort

gap = floor(N/2)

⇒ gap1 = floor(gap/2)

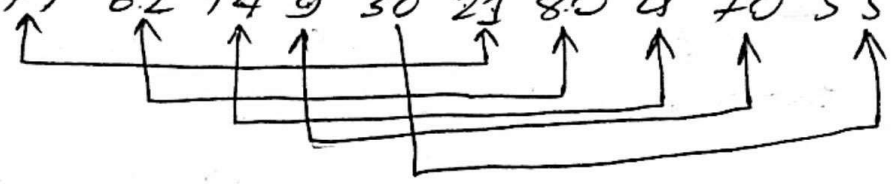
⇒ gap2 = floor(gap1/2)

original list = 77 62 14 9 30 21 80 25
70 55

N = 10

⇒ gap = floor(N/2) = floor(10/2) = 5

pass 1:- 77 62 14 9 30 21 80 25 70 55



21 62 14 9 30 77 80 25 70 55

pass 2:- gap = floor(5/2) = 2

(21) 62 (14) 9 (30) 77 (80) 25 (70) 55

14 9 21 25 30 55 70 62 80 77

pass 3: gap = floor(2/2) = 1

14 9 21 25 30 55 70 62 80 77

9 14 21 25 30 55 62 70 77 80

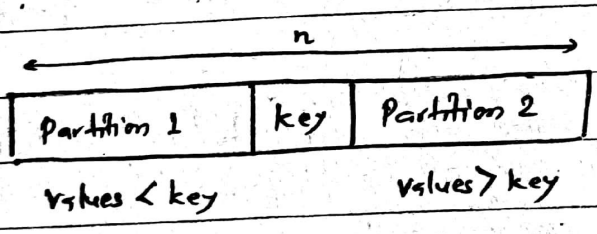
Quick sort

It is one of the most popular sorting technique. It possess a very good average case behaviour among all the sorting techniques. The quick sort algorithm works by partitioning the array to be sorted and each partition is turned sorted recursively. In partition, one of the array element is choosen as a key value and this key value can be the first element of an array.

ie. If A is an array then, key = A[0]

and rest of the array elements are grouped into two partition such as :-

(i) One partition contain element smaller than key value.
(ii) Another partition contain elements larger than the key value.



Eg. 25, 37, 17, 81, 43, 19, 9, 23

The basic of quick sorting algorithm is "divide and conquer strategy." It means divide the original list to be sorted into sublist and solve the sublist until sorted list is found.

We have :-

25, 37, 17, 81, 43, 32, 19, 9, 23

down pointer (greater step) → key value a = 25 ← up pointer (small, equal step)

Let us store the number in an array (x[5]) and let us take a key value a = n[0] = 25.

The array is divided into two part such that the left part has less than 25 (i.e. key value), and the right part has greater than or equal to 25. For doing this let us take two pointer variable called 'up' and another called 'down'. Up pointer is move up one position when the number is greater than or equal to 'a'. And down pointer is move down one position when the number is less than 'a'.

The two pointer up and down are move toward each other in the following way:-

1. Repeatedly increase the down pointer by 1 until n[down] > a

2. Repeatedly decrease the up pointer by 1 position until a >= n[up]

3. If up > down, interchange n[down] with n[up].

4. else up < down

interchange n[up] with key value 'a' then the

will be two sublist, left side of 'a' and right side of 'a'. Continue the same process for both side of 'a' till each array are sorted in order.

25, (37), 17, 81, 43, 32, 19, 9, (23)

down up

25, (23), (17), (81), 43, 32, 19, (9), (37)

→ down up ←

25, (23), (17), (9), (43), 32, (19), (81), (37)

→ down up ←

25, 23, 17, 9, (19), (32), 43, 81, 37

up down

After crossing 'up' and 'down' then exchange 'up' and key value.

19, 23, 17, 9	25	32, 43, 81, 37

$k_1 = 19$ $k_2 = 32$

19, (23), 17, (9) 25 (32) (43) 81 37

down up up down

19 9 (17) (23) 25 32 | 43 81 37 |

up down with $k_4 = 43$

with $k_3 = 17$

17, (9)	19	23	25 32 43 (81) (37)

up down up

16

9, 17, 19 23 25 32 43 (37) (81)
up dam

9, 17, 19 23 25 32 37 [43 81]

9, 17, 19, 23, 25, 32, 37, 43, 81

Sort the following using quick sort:
25, 57, 48, 37, 12, 96, 86, 33

Let us choose the first element 25 as pivot element, so placing 25 in its proper position, the original list is decomposed into two sub-list as:-

a = 25

(12) 25 (57 48 37 96 86 33)

Left sublist (12) does not require any sorting because it has only one element. So

12 25 (57 48 37 96 86 33)

Now repeat the same process for right sublist choosing 57 as pivot element

12 25 (48 37 33) 57 (96, 86)

Similarly we have

Q → 77, 33, 44, 11, 88, 2, 66, 1

17

12 25 (37, 33) 48, 57, (98, 86)

12, 25, (33), 37, 48, 57, (98, 86)

12, 25, 33, 37, 48, 57, (86), 98.

12, 23, 33, 37, 48, 57, 86, 98

PRIORITY QUEUE (HEAP)

A priority queue is a data structure that allows at least the following two operations:-

(i) Insert, which insert a new element into the rear of the priority queue

(ii) Delete minimum, which find the minimum elements from the priority queue and remove it.

The insert operation is equivalent to Enqueue and delete minimum is equivalent to d-queue operation of the key.

Binary Heap

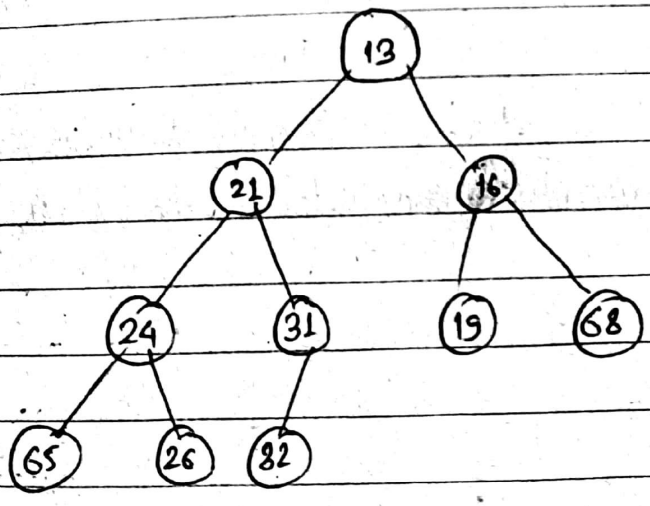
A heap or binary heap is defined to be a binary tree with the key value in each node such that :-

(i) All the leaves of the tree are on two adjacent level.

(ii) All the leaves on the lower level occured to the left, and all the leaves except possibly the last one are filled

(iii) The key in the root is at least as large as the key in its children (in case of maximum binary heap) or at least as small as the keys in its children (in case of minimum binary heap). And again the left and right subtrees are binary heap.

The first two conditions which are also known as structural property of binary heap, ensures that the contiguous representation of the tree element is in memory and the third condition which is known as the heap order property determines the ordering of elements in the tree.



→ all the roots are less than its children.

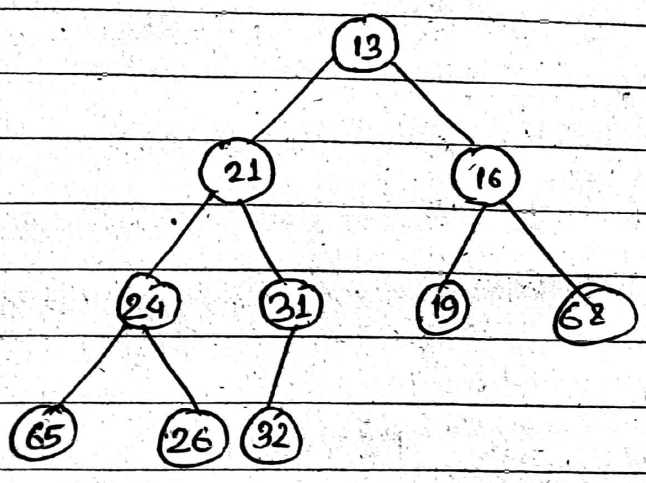
→ while making binary heap, place the elements always from left and then to right.

Basic operation of Binary heap

1. Insert :

To insert an element 'x' into a heap, we create a hole in the next available location. If x can be placed in the hole without violating heap order property, then we do so. Otherwise we slide the element of parent node into the hole. Therefore bubbling of the whole elements towards the root. Continue this process until x is placed in its proper position.

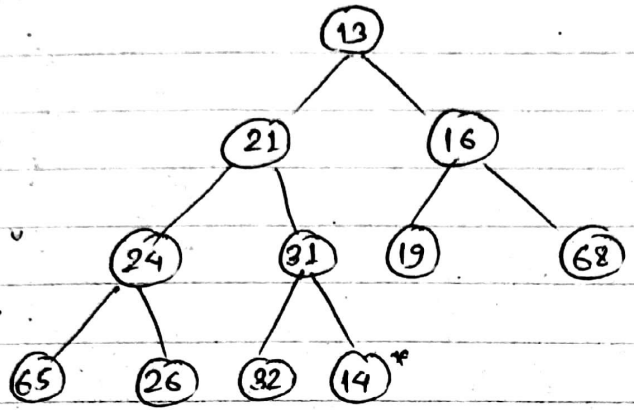
For eg. Suppose we are given a minimum binary heap as :-



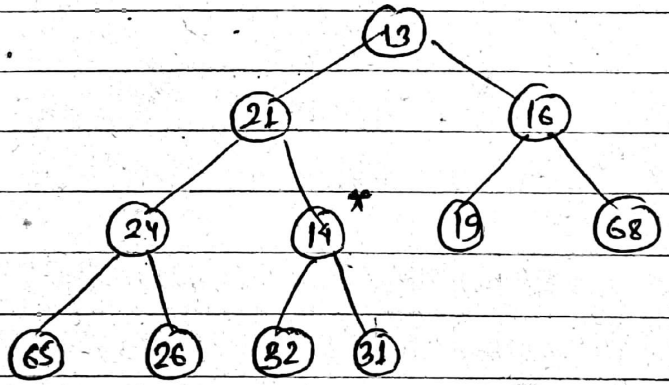
Now insert 14

STEP 1: Create a hole in the right of 31 and place 14 there.

20



STEP 2: Since the child (14) is less than its parent 31, So ordering property violates. So bubble up the child and slide down the parent.



STEP 3: Since 14 is smaller than (21). So once again bubble up (14) and slide down (21)

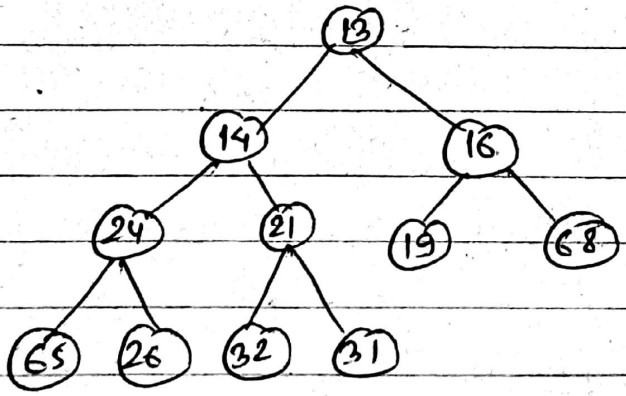


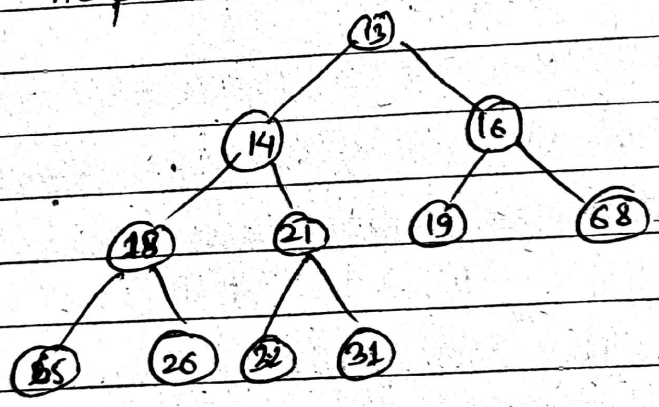
fig. final binary heap.

2.

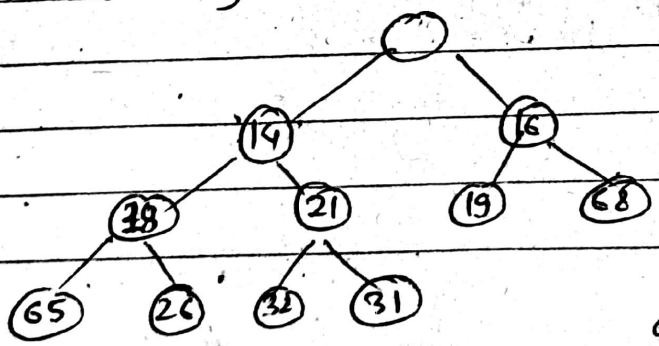
2. Delete minimum

To delete an element from binary heap, first of all minimum element is selected and it is deleted. Since in the minimum heap, the minimum element is the root element So it is easy to find it (minimum number). But it is difficult to maintain the ordering after removing it. For this we bubble up the smaller of the hole children into the hole. Therefore pushing the hole one level down we repeat this step until the last element can be placed in the hole. Therefore our action is to place the last element in its correct position along a path from the root containing a minimum children. This strategy is also known as 'perpolate down'.

Perform the delete minimum action on the following binary heap :-

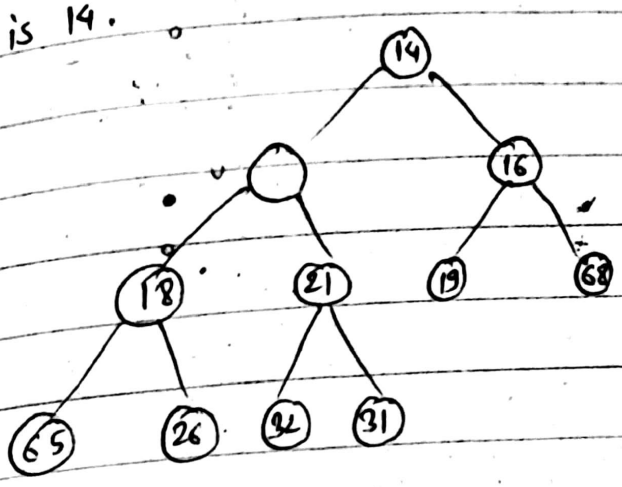


STEP 1 : Deleting the minimum element from root we have :-

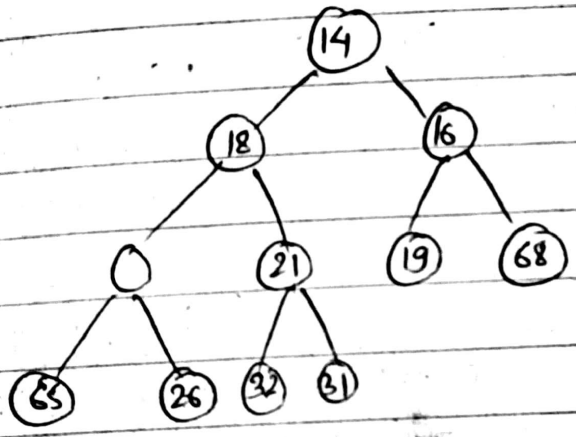


Creation of hole at root

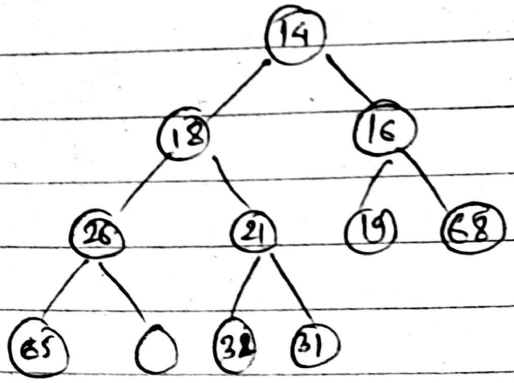
STEP 2: Bubbling up holes smaller element. In this case it is 14.



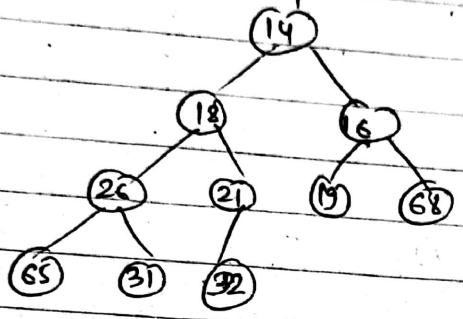
STEP 3: Again bubbling up the holes smaller element which is 18 in this case.



STEP 4: Again bubbling up the holes smaller element which is 26



STEP 5: Since the hole has no children. So, now placing the last element (31) in this newly created hole, the final binary heap will be as:



8. HEAP SORT

The operation for sorting a list using heap sort are :-

(i) Construct a binary heap from the list of given elements (maximum heap)

(ii) Swap the last element of the list to the root element by preserving heap order property and delete the last element.

(iii) Repeat step 2 until the list is sorted (ie. until all elements are deleted)

24

Sort the following list using heap sort:

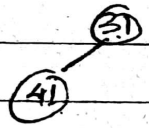
31, 41, 59, 26, 53, 58, 97

Sol^n: Let us ~~consider~~ construct a binary heap for the given list (Maximum heap)

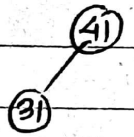
STEP 1:

On inserting 31,

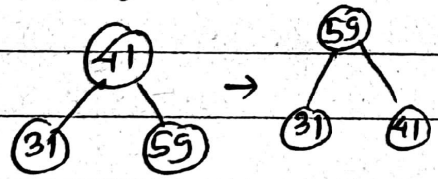
On inserting 41,



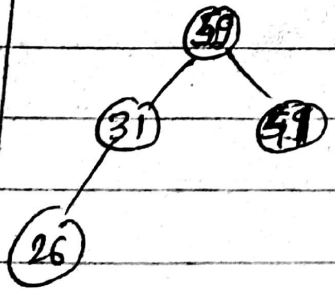
On maintaining order



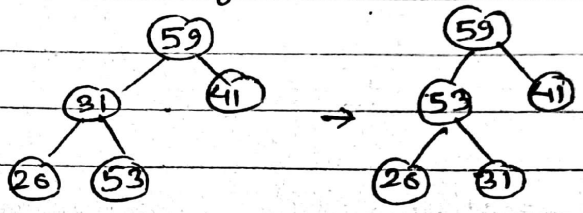
On inserting 59



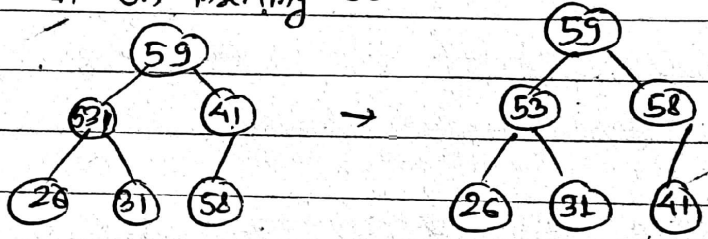
On inserting 26



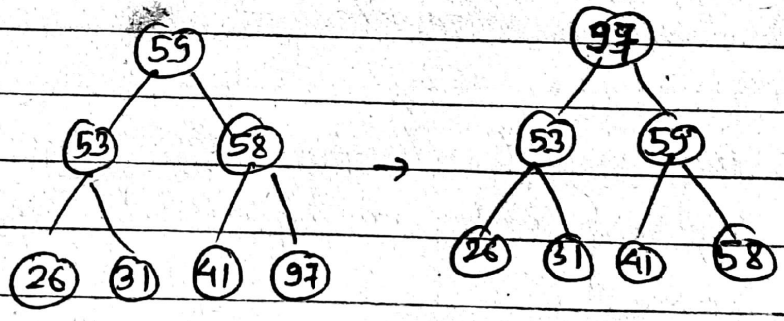
On inserting 53



On inserting 58



On inserting 97

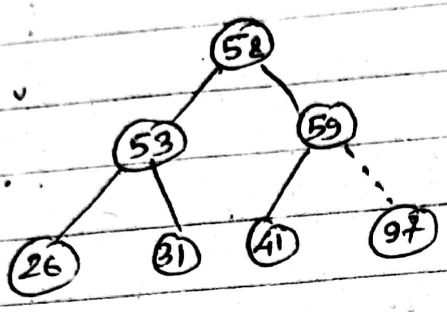


Final binary heap

97	53	59	26	31	41	58

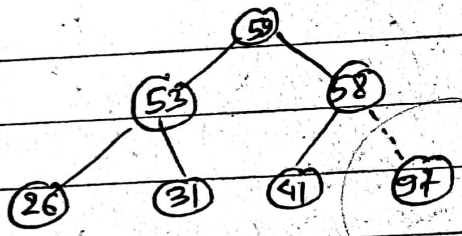
STEP 2:

(a) Swap 58 with 97, delete new last element, then we have;



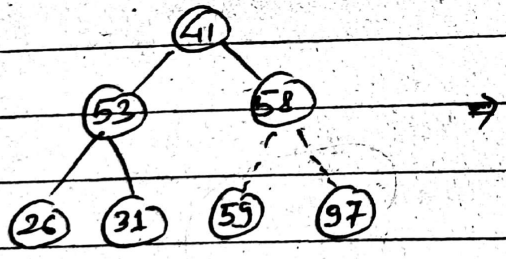
58	53	59	26	31	41	97

On maintaining heap order property swap 58 with 59

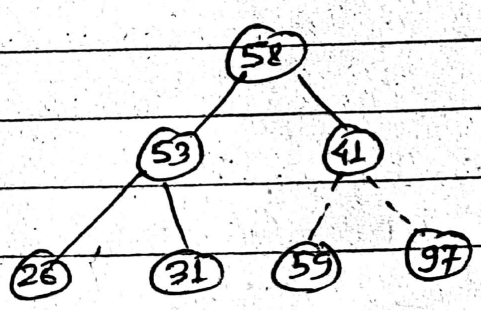


59	53	58	26	31	41	97

(b) Swap last element 41 with the root 59, delete last one and maintain heap order.

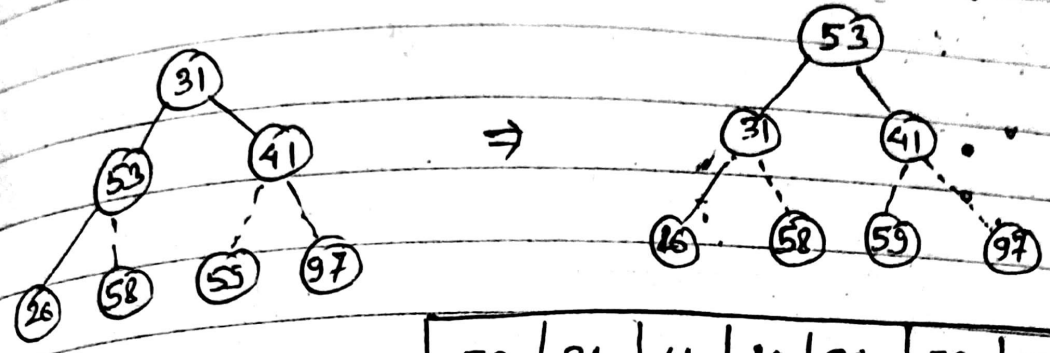


On maintaining,



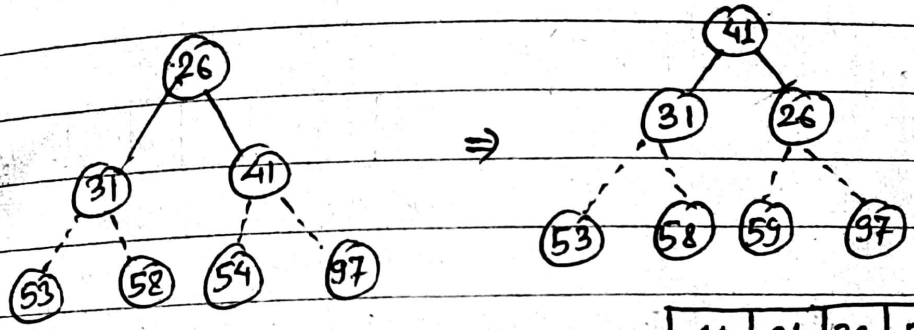
58	53	41	26	31	59	97

(c) Swap last element (31) with root 58, delete last element and maintain heap order.



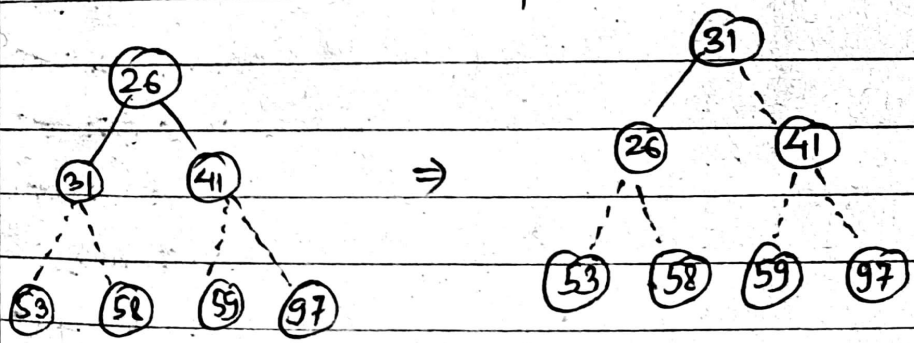
53	31	41	26	58	59	97

(d) Swap last element 26 with root element 53, delete last element and maintain heap order.



41	31	26	53	58	55	97

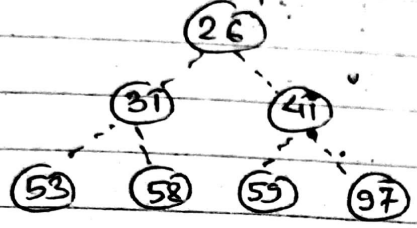
(e) Swap last element 26 with the root 41, delete the last one and maintain heap order.



31	26	41	53	58	55	97

27

(b) Swap last element 26 with root 31, delete last element and maintain heap order.



26	31	41	53	58	59	97

final sorted list