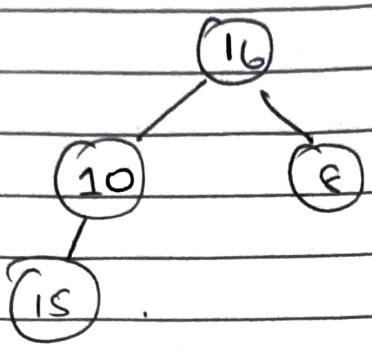
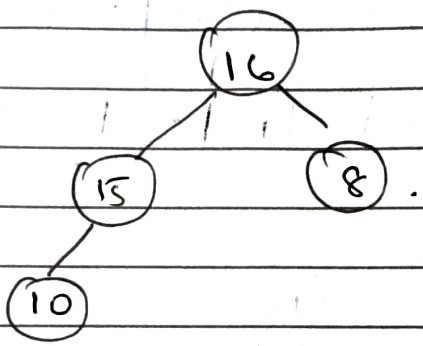


Step I: Compare child 15 with parent node 10.

∵ 15 > 10.

∴ Interchange.



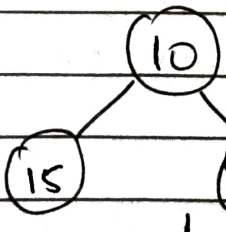
Now it is max heap.

Then, we repeat the step of removing the very first element....

step xi: Current BT :

16	15	8	10	20	30	50

After removing:



updated BT :

10	15	8	16	20	30	50

unsorted portion

sorted portion.

Step XII : compare sibling of parent node 10.

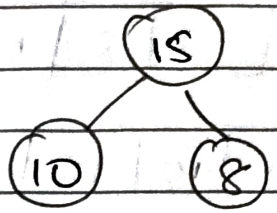
∵ 15 > 8.

∴ compare parent 10 with child 15

∵ 15 > 10

∴ Interchange.

Step XIII :

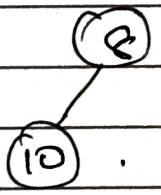


Since, it is a max heap now.
we repeate the step of removing very first node.

Step XIX : current BT:

15	10	8	16	20	30	50

removing Root node :



updated list :

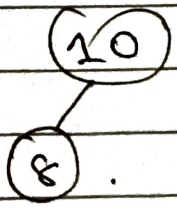
8	10	15	16	20	30	50

Step XX : compare child 10 with parent 8.

∵ 10 > 8.

∴ Interchange.

Tree :



Now, it is max heap
then we repeat the step of removing the root node.

Step XXI : (8)

current BT :

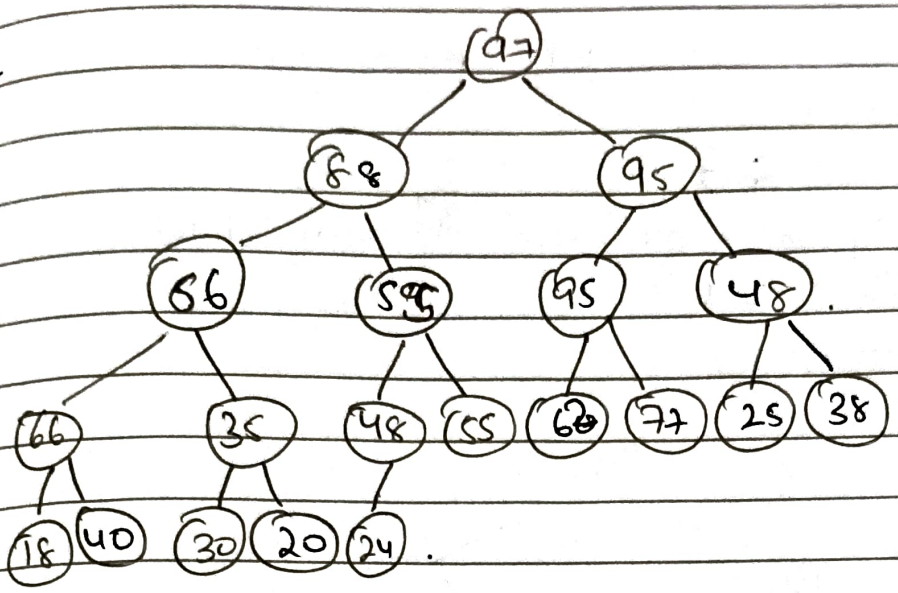
10	8	15	16	20	30	50

updated BT :

8	10	15	16	20	30	50

Now it is sorted.

Q.

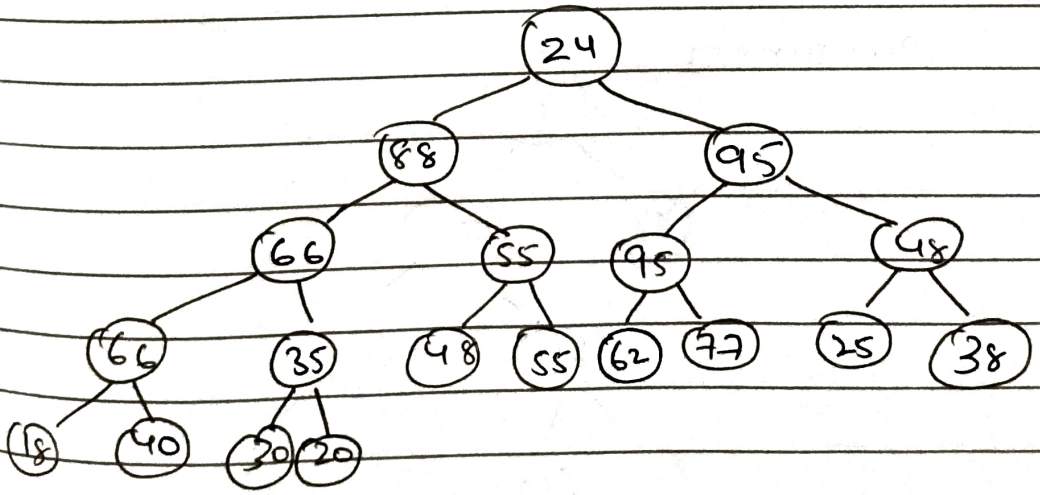


97	88	95	66	55	95	48	66	35	48	55	62	77	25	38	18	40

30	20	24

⇒ Objective is to remove the very first root node from the tree and placed at the end of the list and rearranged the tree in a max heap form.

⇒ will replace root node with last node.



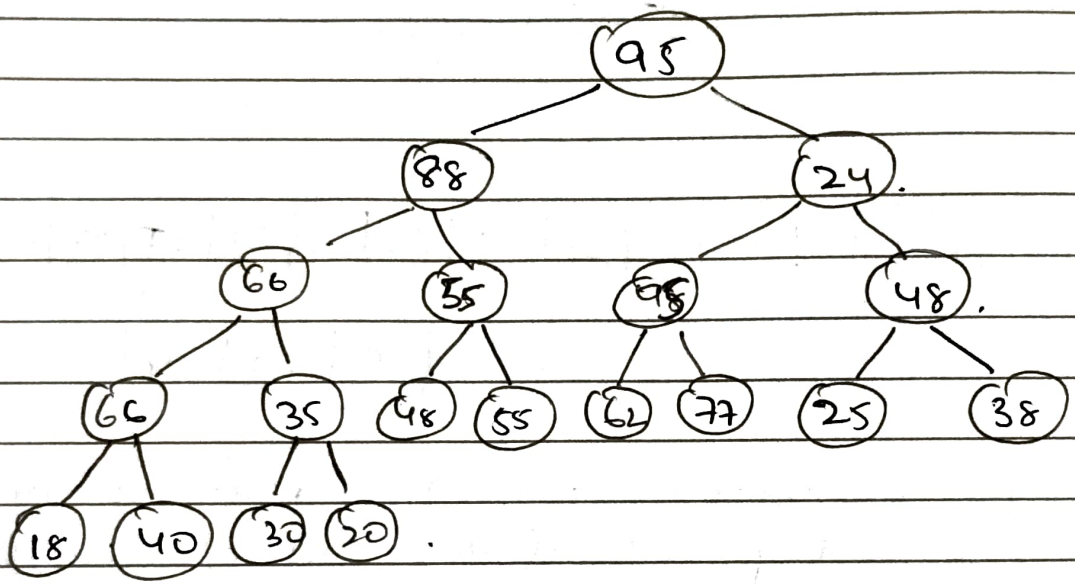
Step 1: Compare siblings of every first root node 24.

∵ 95 > 88.

∴ compare root node 24/95 with child 95.

∵ 95 > 24

∴ Interchange.



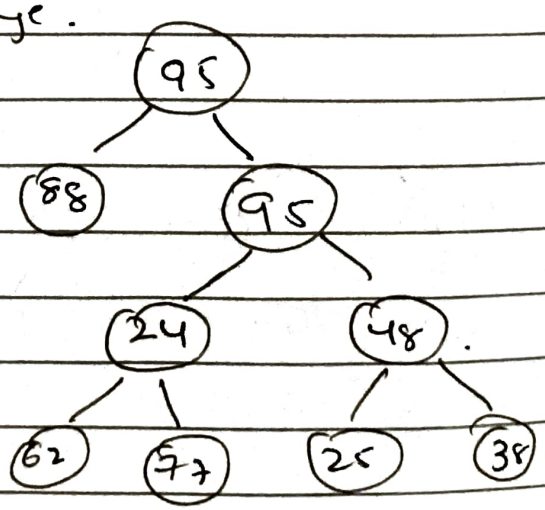
Step II: Compare siblings of 24.

∵ 95 > 48.

∴ compare parent 24 with child 95

∵ 95 > 24.

∴ Interchange.



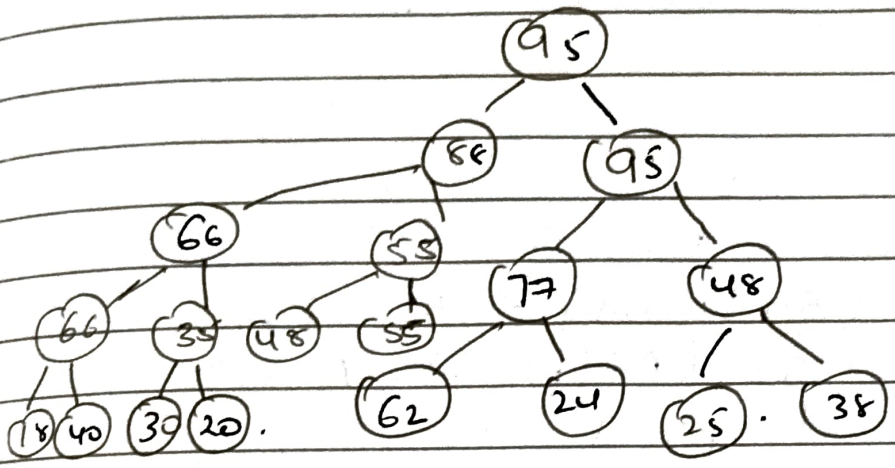
Step III: compare sibling of 24.

∵ 77 > 62

∴ compare parent 24 with child 77.

∵ 77 > 24.

∴ interchange.



Now it is max heap.

Code:

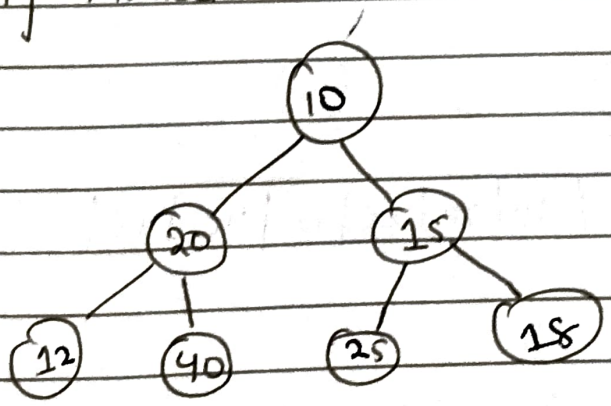
```
def insert_heap (Arr, n, value):

    n = n + 1
    Arr[n] = value
    i = n

    while (i > 1):
        parent = math.floor (i/2)
        if (Arr[parent] < Arr[i]):
            swap(Arr[parent], Arr(i))
            i = parent
        else:
            return

H = [50, 30, 20, 15, 10, 8, 16]
n = len(H)
value = 25
insert_heap (H, n, value)
```

$1 < \log n < \sqrt{n} < n < n\log n < \dots n^2 < n^3 < \dots 2^n < 3^n < n^n$

Heap Sort 08-Oct-2024

Insertion	Deletion
1. Insertion is done from leaf node	Deletion is done from Root node.
2. Direction of adjustment is from leaf to root node	Direction of adjustment is from root to leaf node.
3. Perform on complete BT.	perform on complete BT.
4. The height of BT is $O(\lceil \log(n) \rceil)$ while inserting node to heap.	The height of BT is $O(\lceil \log(n) \rceil)$ while deleting node from heap.

*Heapify Method



1	2	3	4	5	6	7
10	20	15	12	40	25	18

To find all the leaf node. formula. $n/2+1$ upto n.

$n = \text{len}(arr) = 7$! $\left(\left\lfloor \frac{7}{2} \right\rfloor + 1 : n\right) : (4 : n)$

from 4th position to n are the leaf node.

Step 1 : find left and Right child of node 3 node

left child : $2 * i$.

$= 2 * 3$

$= 6 . \rightarrow [25]$

Right child = $2 * i + 1$

$= 2 * 3 + 1$

$= 6 + 1$

$= 7 \rightarrow [18]$

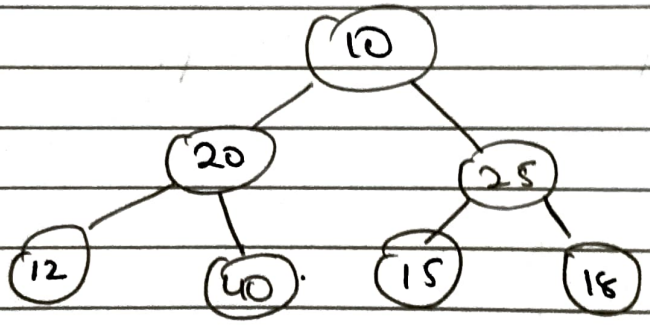
Compare siblings of node at index 3 [15]

$\because$ 25 > 18.

$\therefore$ Compares parent with child 25

$\because$ 25 > 15

$\therefore$ interchange.



Current updated list = | 10 | 20 | 25 | 12 | 40 | 15 | 18 |

Step 2: decrement i i.e, i = 2

left child = $2 * i$

$= 2 * 2$

$= 4$ — [12]

Right child = $2 * i + 1$

$= 2 * 2 + 1 = 5 \rightarrow [40]$

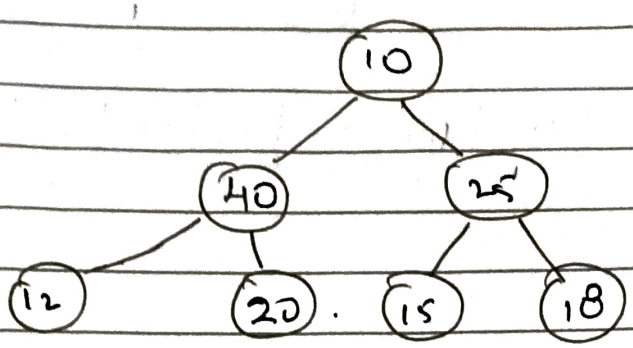
compare siblings of node at index 2 [20]

∵ 40 > 12

∴ compare parent [20] with child 40.

∵ 40 > 20

∴ interchange.



current updated list =

10	40	25	12	20	15	18
1	2	3	4	5	6	7

Step 3: decrement i, i.e i = 1

left child = 2 * i

= 2 * 1

= 2 → [40]

Right child = 2 * i + 1

= 2 * 1 + 1

= 2 + 1

= 3 → [25].

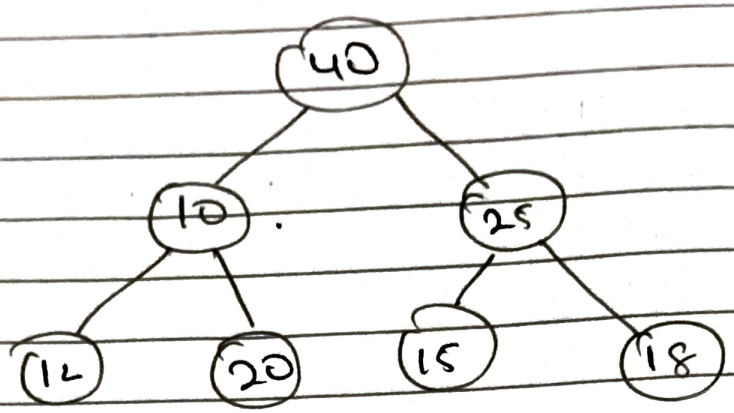
compare siblings of node at index 1 [10]

∵ 40 > 25

∴ compare parent [10] with child 40.

∵ 40 > 10.

∴ interchange



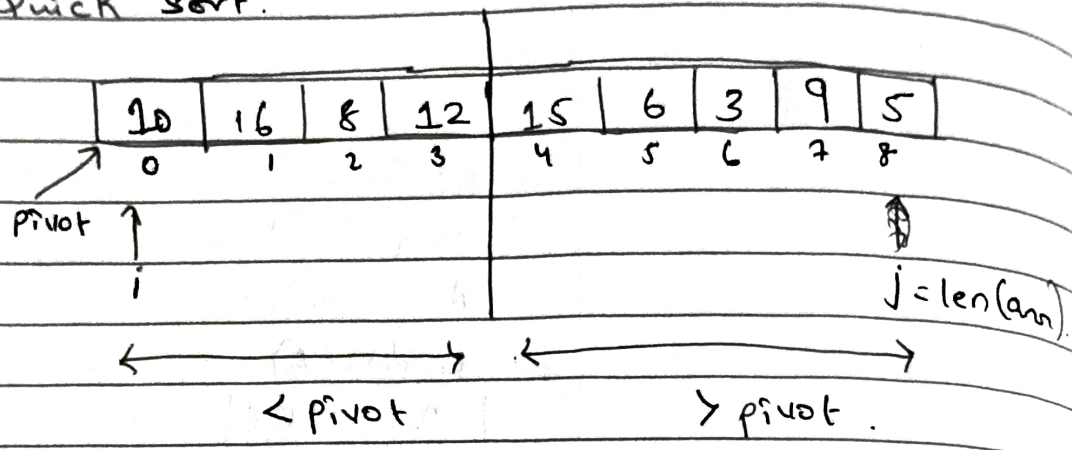
updated list :

40	10	25	12	20	15	18
1	2	3	4	5	6	7

Ds using py

14-Oct-2024

* Quick Sort.



↳ we have to find out the pivot point of the array, and pivot point should always be ~~the~~ ~~a~~in the middle of the array. Such that, all the element in the left side should be smaller than pivot, and all the element on the right side should be the greater than pivot point.

↳ In quick sort, we always try to keeep the lesser number than the pivot are its left side, and greater number on right side.

↳ We've to find all the numbers, which is greater than pivot on the left portion of the array.

↳ we've to find all the numbers, which is lesser than pivot on the right portion of the array.

Step I:

```
do {
    i++;
} while (A[i] < pivot);
```

```
j = len(arr)
do {
    j--;
} while (A[j] > pivot);
```

Step I → i = 0 + 1.

A[i] = 16 < 10 ✗ pivot

stop

j-- = 8.

A[8] > pivot

5 < pivot ✓

swap (A[i], A[j])

i<j	10	5	8	12	15	16	3	9	16
	0	1	2	3	4	5	6	7	8

Step II :- i = 1.

A[1] = 5

5 < 10. ✗ pivot

yes.

i++ = 2.

A[2] = 8.

8 < 10. ✗ pivot

yes.

i++ = 3.

A[3] = 12.

12 < 10.

NO.

stop.

j = 7.

A[7] = 9.

9 > 10.

NO.

stop.

3 9 < i 7 ✓

swap (A[3], A[7].

swap (12, 9)

updated list

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

Step III :

i = 3

i++ = 4.

A[4] = 15

15 < 10 ← pivot

NO.

stop.

j = 7

j-- = 6.

j[6] = 3

3 > 10

NO.

stop.

4 6

i < j ✓

swap (A[4] A[6])

swap (15, 3).

updated list

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

Step IV :

i = 4

i++ = 5

A[5] = 6.

6 < 10 ← pivot

yes.

i++ = 6.

A[6] = 15

15 < 10 ← pivot

NO.

stop

j = 6

j-- = 5

A[5] = 6.

6 > 10

NO.

stop.

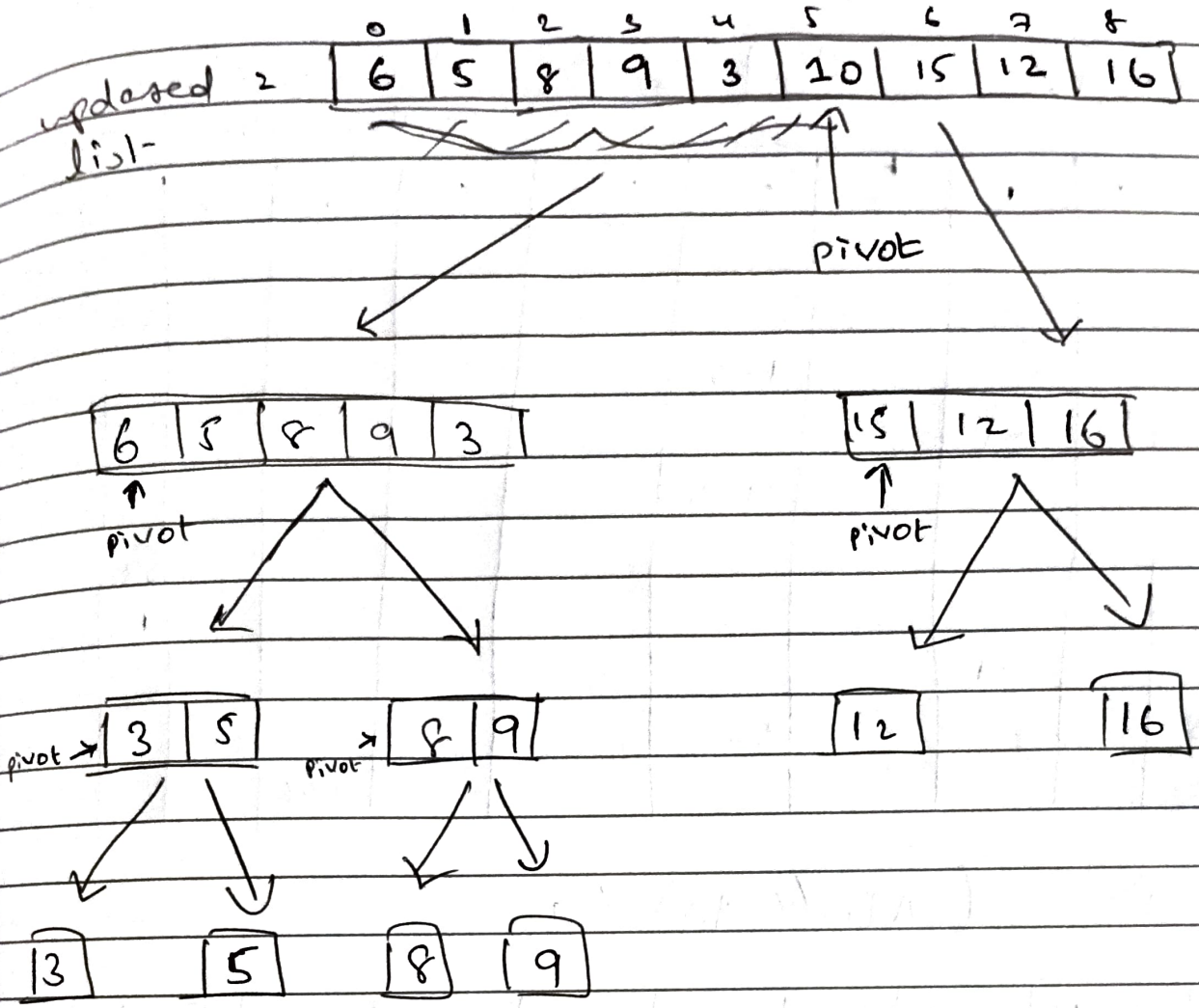
6 5

i < j ✗

NO swapping

swap (~~A[0]~~ pivot, A[j].)

swap (10, 6).



* ~~Matrix~~ Radix Sort algorithm

Bin size of the list should be the base of the type of the element.

No. of passes will be the max num of digit hold's the element in the list.

pass 1) A[i] % 10. , will place each elem... on their respective mod V... positions.

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361		361								
423				423						
538									538	
128									128	
321		321								
543				543						
366							366			

pass 2) (A[i] / 10) % 10.

361	321	143	423	543	366	348	538	128
0	1	2	3	4	5	6	7	8

Input	0	1	2	3	4	5	6	7	8	9
361							361			
321			321							
143					143					
423			423							
543					543					
366							366			
348					348					
538				538						
128			128							

321	423	128	538	143	543	348	361	366
0	1	2	3	4	5	6	7	8

pass 3 $[(A[i]/100) \% 10]$

input	0	1	2	3	4	5	6	7	8	9
321				321						
423					423					
128		128								
538						538				
143		143								
543						543				
348				348						
361				361						
366				366						

128	143	32	348	361	366	423	538	543
0	1	2	3	4	5	6	7	8

Now it is sorted.

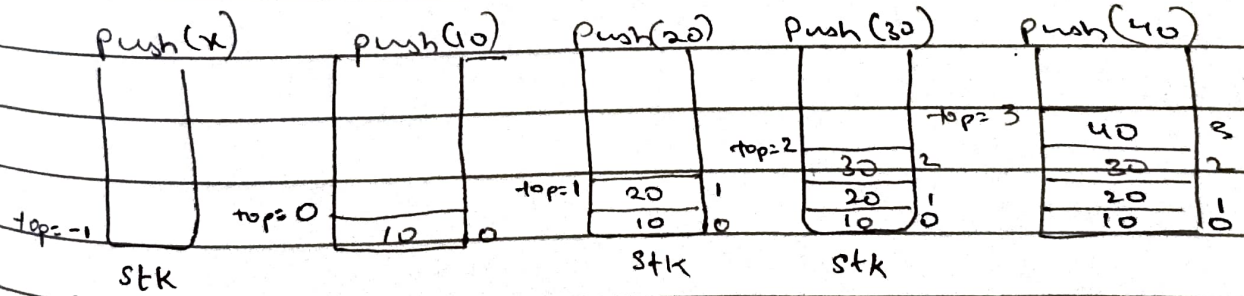
DS using Python 22-Oct-2024

Stack Data Structure

Definition: It is a list which a restriction, that insertion and deletion can be perform only from one end called as top.

Functions: 1. push(x)
2. pop()
3. is_empty()
4. peek() → Top → elem present top.
5. is_full → check whether stack is full or not.

The relation between top and index are same, so we can say that top = index.



Method 1.
```
if (top == -1)
    print("Stack is empty! Can't remove further").
```

How to check stack is empty.

~~def isfull():~~

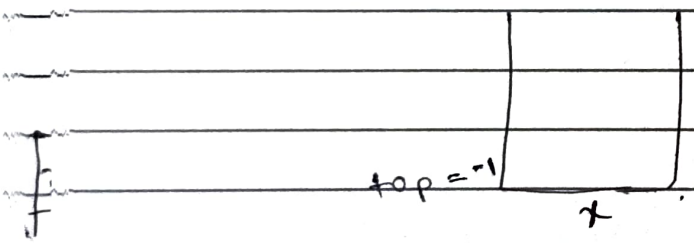
```
if (top == len(stk) - 1):
    print ("stack is full")
```

```
def pop(stk):
    if (top == -1):
        print ("stack is empty");
    else:
        popped_element = stk.pop()
        top -= 1
```

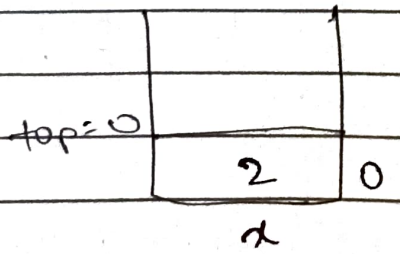
Assignment: push 8, 3, 1, 4 into a stack 'x', which is initially empty

Soln:

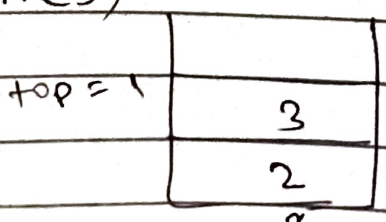
Step 1 : x = []



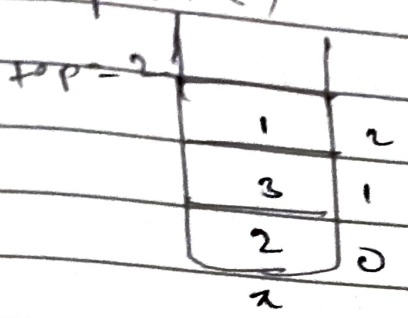
Step 2: push (2).



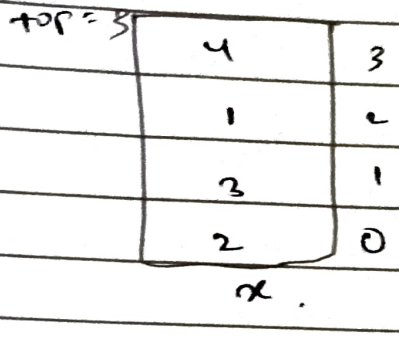
Step 3: push (3)



Step 4 : push (1)

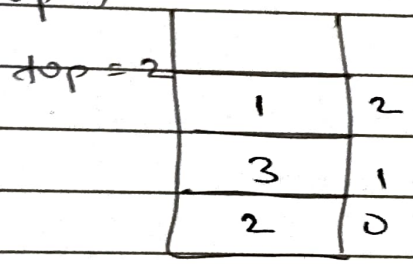


Step 5 : push (4)

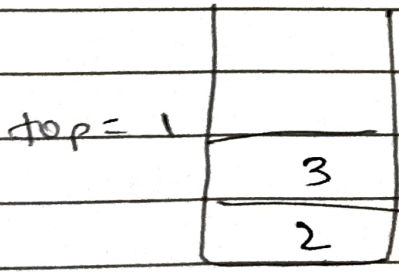


Q. Find the elem 1

Step 4 : pop()



Step 5 : pop()



↳ found the element 1.