Radix sort

means → base

sort by bit

* non-comparison based

Least significant bit
LSB

by appending 0

make all digits equal to largest

904	904	
46	046	→ arrange based on LSB
5	005	
74	074	
62	062	
1	001	

arrange

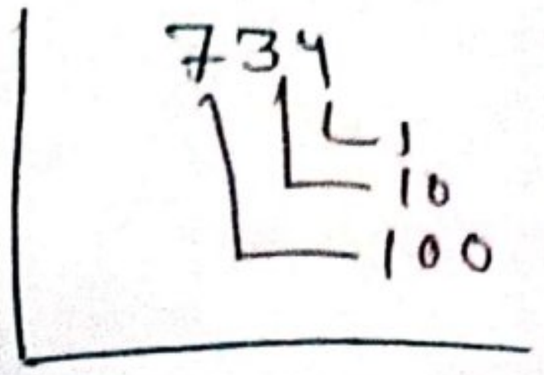
001	
904	
005	same bit
046	donot change relative position
062	
074	

when same bit donot change relative position

001
005
046
062
074
904

sorted

A = 97 . 57, 208, 699, 125, 734



(1) step-1 :- make all digit size same

0(9)7 0(5)7 2(0)8 6(9)9 12(5) 73(4)

always size 10 digit nums (0-9)

(2) step-2 :- → countarr [10]

→ init all value 0

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

(3) step-3 :- count the LSD digit and update countarray

countarr-

0	1	2	3	4	5	6	7	8	9
0	0	0	0	1	1	0	X2	1	1

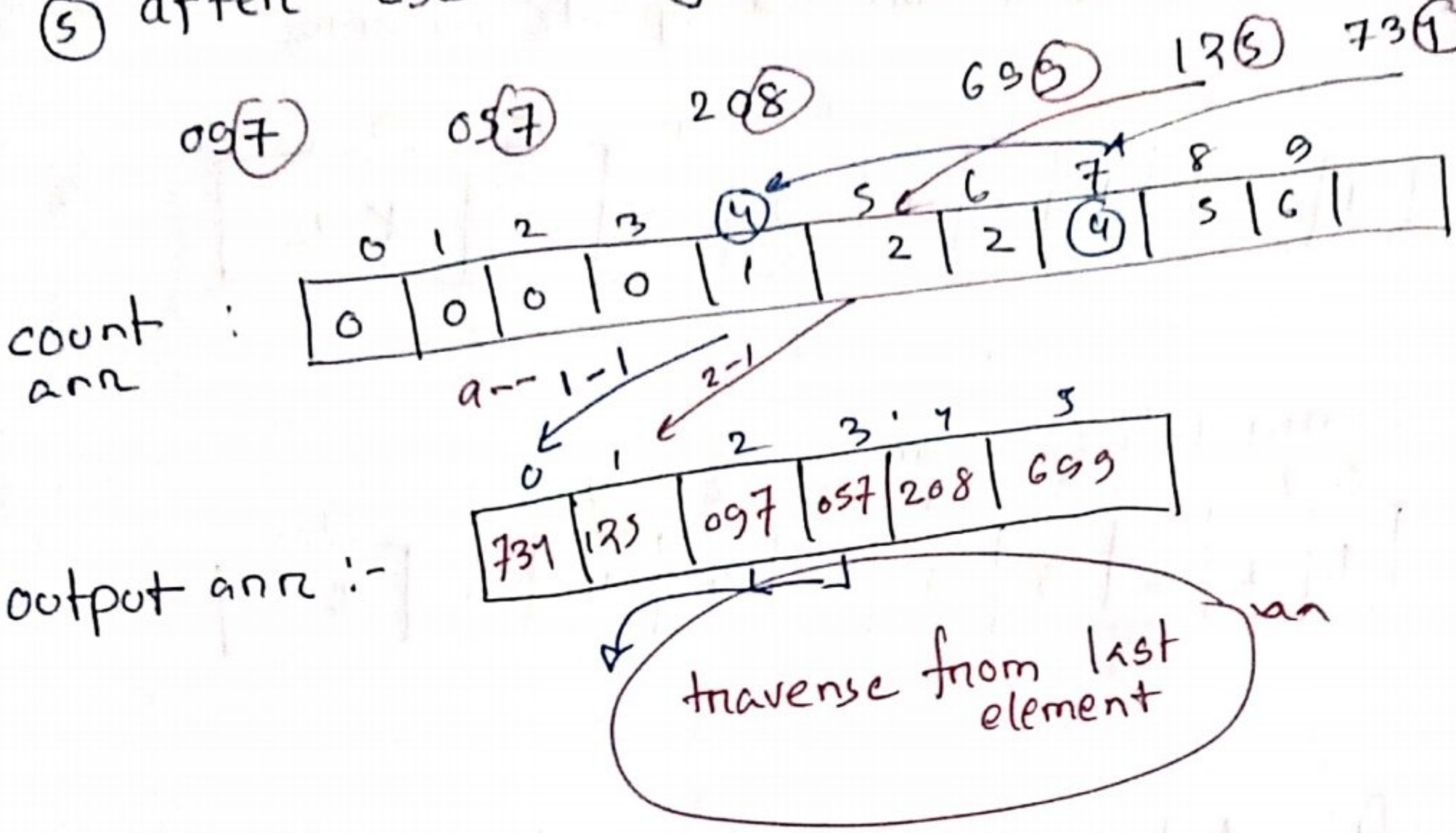
(4) Now commalitive sum of countarr

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

start from last

basic sorted arr

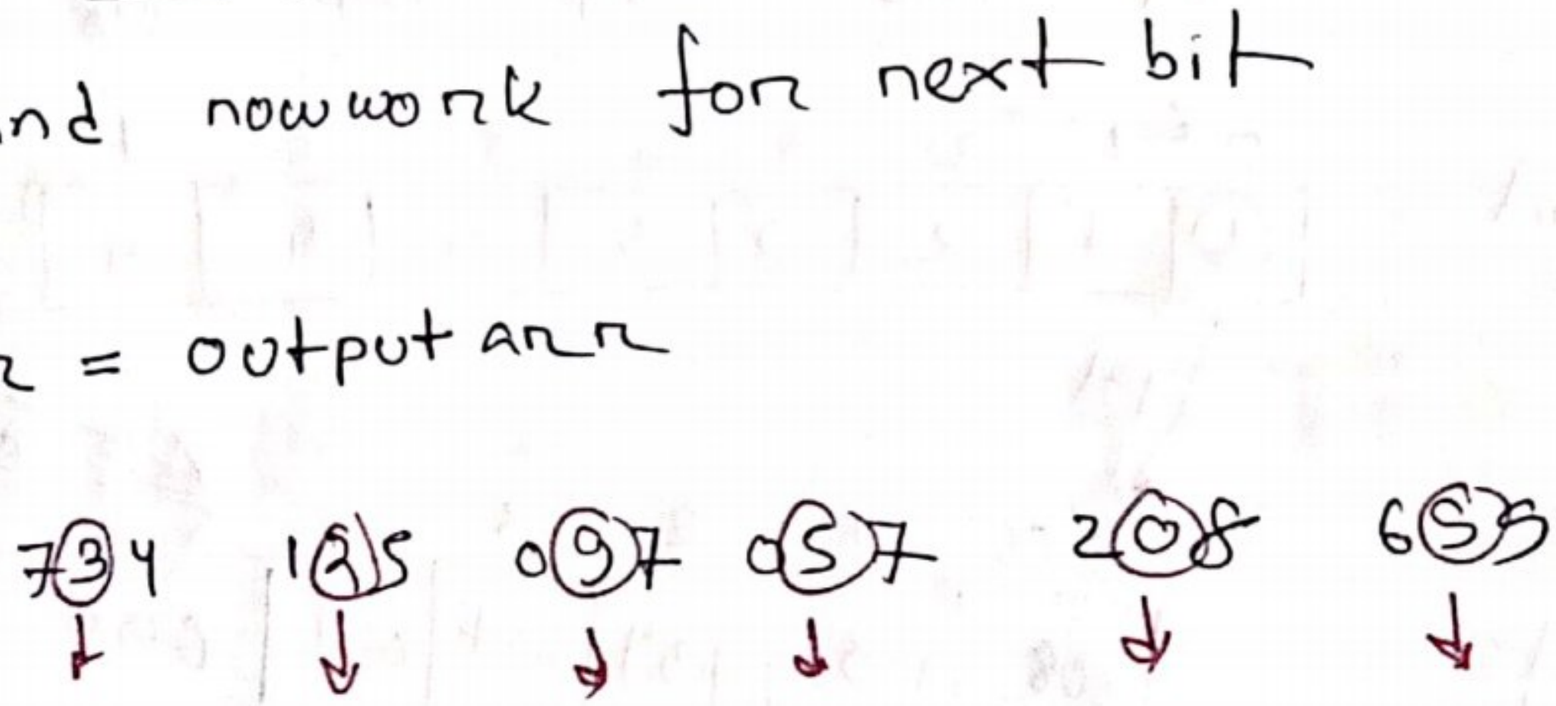
(5) after LSB arrange



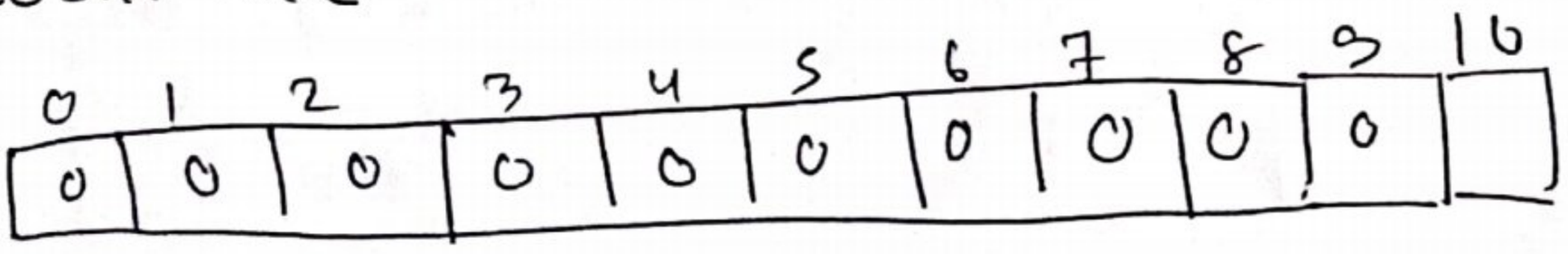
—— o ——

(4) do again and now work for next bit

(6) main arr = output arr



(7) initial count arr = 0



(8) count the 2nd bit and update in count arr

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

(9) cummalitive sum

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

(10) aften bit annange basic sont annay

Traverse from ← Left

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

count arr

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

1-1

6-1

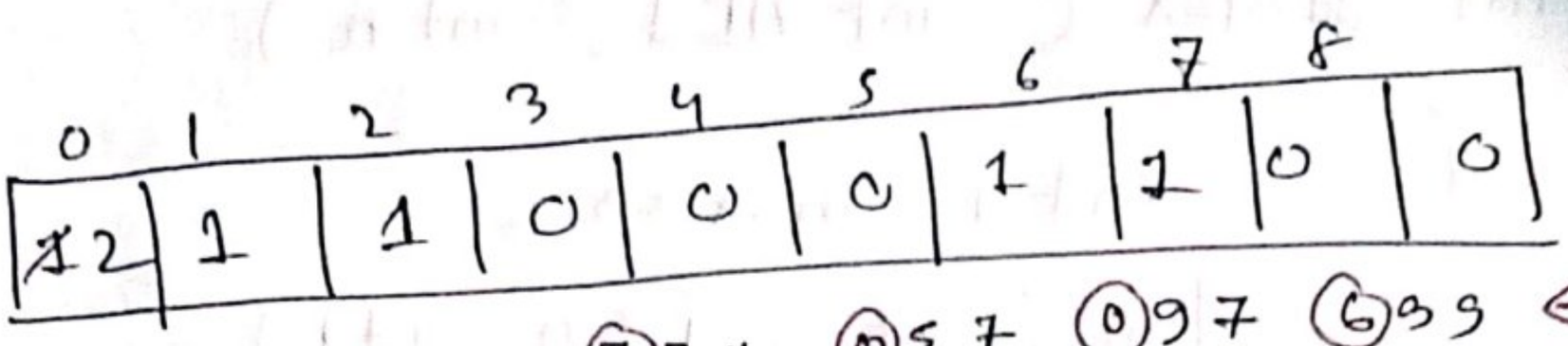
Out put arr

0	1	2	3	4	5
208	125	734	057	097	609

Same ...

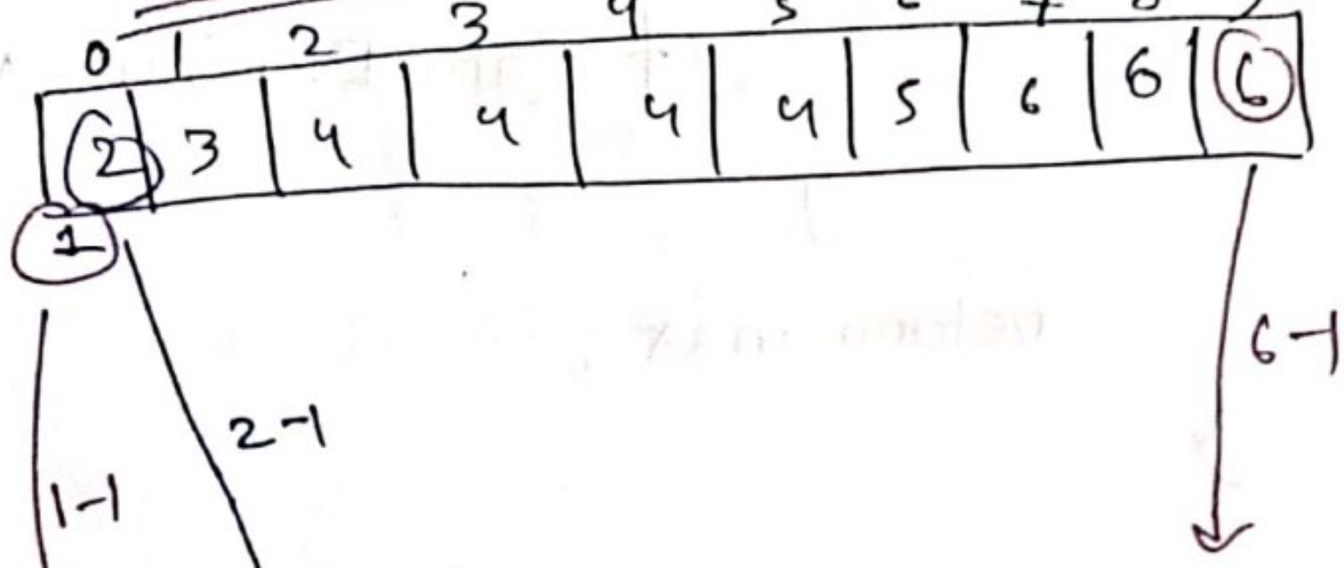
(2)08 (1)2 5 (7)3 4 (0)5 0 7 (0)9 7 (6)9 9

→ count arr :-

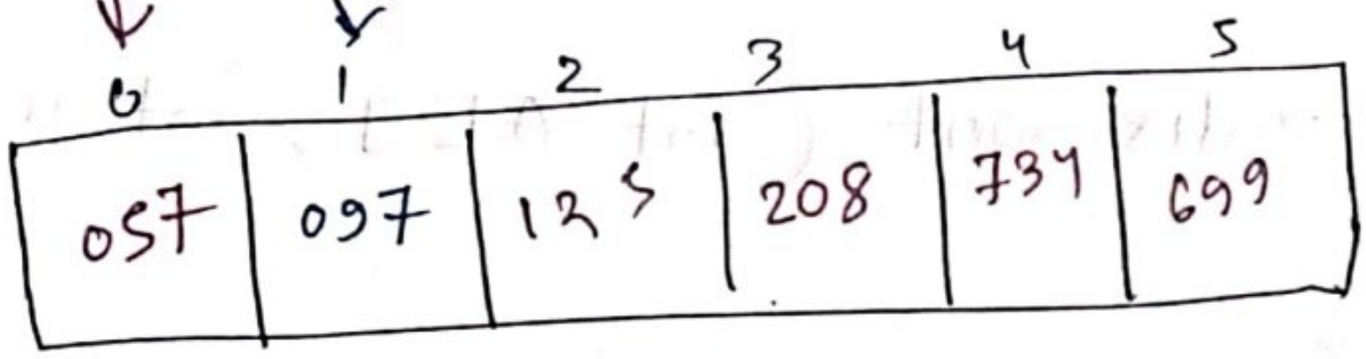


(2)08 (1)2 5 (7)3 4 (0)5 7 (0)9 7 (6)3 9

→ cummalitive :- sum



→ output arr



sorted array

d = highest number of digit

Time complex → $O(n^d)$

space → $O(n^d)$

Pseducode

```
int getMax ( int A[], int n )
{       int i, max=0 ;
        for (i=0 ; i<n; i++)
            { if (arr[i] >max) max = A[i]
            }
        return max ;
}

void radix sort ( int A[], int n )
{
    int m = getMax ( A , n)
                                    bit
    for (pos = 1 : (m/pos > 0); (pos *= 10))
        {
            countsont (A, n, pos)
        }
}
```

```
void countsort ( int A[], int n, int pos )
{
    int output[n]
    int count = {0}

for (i= 0;  i<n ; i++)
    {
    count[ (A[i]/pos) %.10 ] ++ ;
    }
```

getting desire bit

```
for (i = 1; i<= 9 ; i++)
    {
    count[i]= count[i] + count[i-1] ;
    }
```

cummulative

traverse

```
for (i =n-1 ;  i >= 0  ; i-- )
    {
    output[--count[ (A[i]/pos)%.10]] = A[i]
    }
}
```

Bucket Sort

* floating point
* Range (0.0 – 1.0)

① Uniform / Best case

n = 10

0.79 0.13 0.64 0.39 0.20 0.89 0.53 0.42 0.06 0.94

always B = 16

0	0.06
1	0.13
2	0.20
3	0.39
4	0.42
5	0.53
6	0.64
7	0.79
8	0.89
9	0.94

* check the first value after (.dot)
* then put that in bucket

time

$O(n+k)$

space

$O(n+k)$

n = number of element in A
k = number of bucket formed

Pesudocode

(1) Bucket sort (int A[ ])

(2) Let B[n] be new array as bucket

(3) n = A.Length()

(4) for (i=0 to n-1)
B[i] empty list

(5) for (i=1 to n)
instert A[i] into list
B[⌊nA[i]⌋]

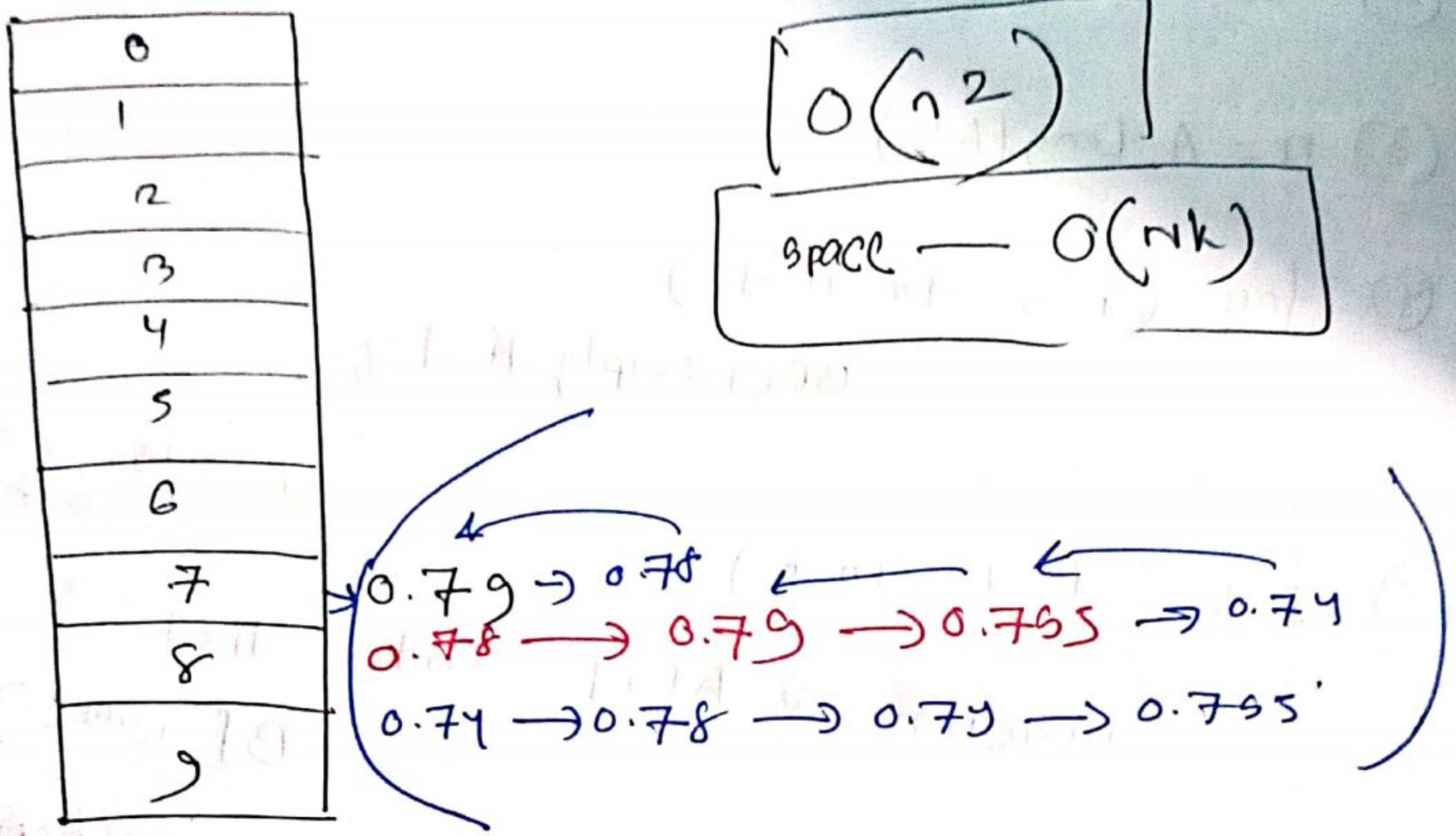
10x.79 = 7.9
=7
↓
lower/floor value

(6) for (i=0 to n-1)
sort List B[i] with insertion sort

(7) concatenate the list together

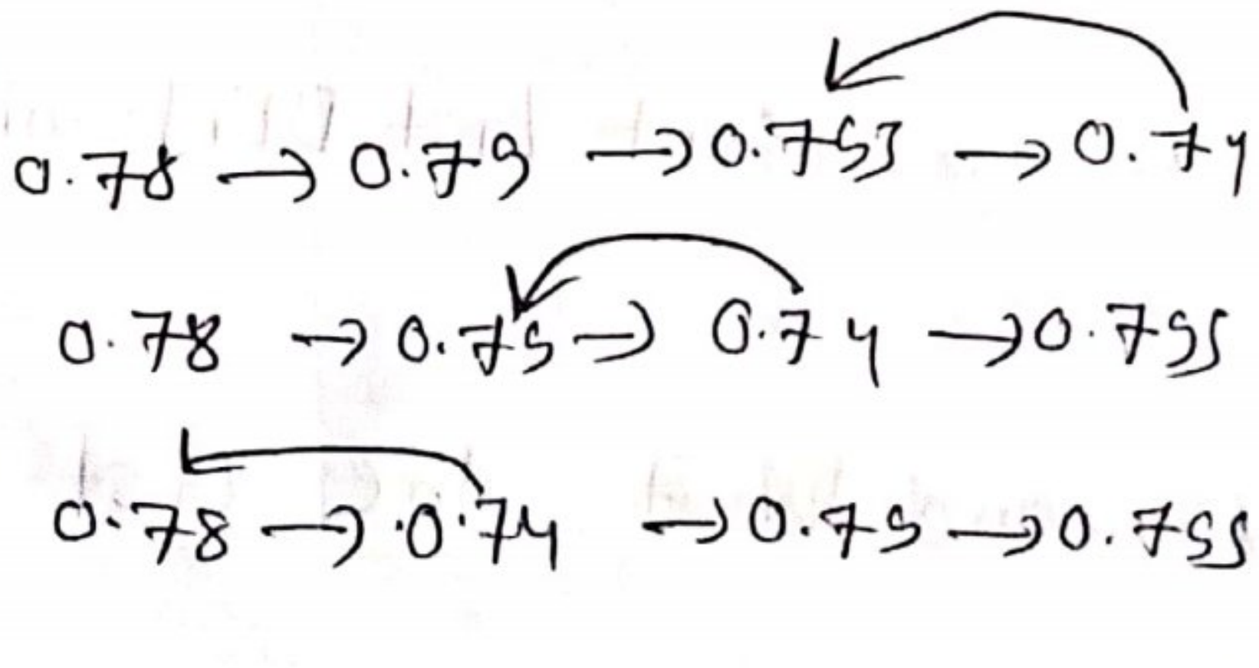
(*) Worst case

0.79 ✓ 0.78 0.795 0.74



Insertion sort

compare two elements and swap until desired pos found



0.74 → 0.78 → 0.79 → 0.795

Merge Sort

time : $O(n \log n)$ ← best / worst / avg.

space : $O(n)$

pseudocode :

```
void mergeSort (arr, l, r) {
    if (l > r) ret;
    mid = (l + r) / 2;
    mergeSort (arr, l, mid);
    mergeSort (arr, mid+1, r);
    merge (arr, l, mid, r);
}
```

Time Complexity Analysis :

$$T(n) = 2T(n/2) + c \cdot n$$

$$= 2\left\{2T(n/4) + c\frac{n}{2}\right\} + cn$$

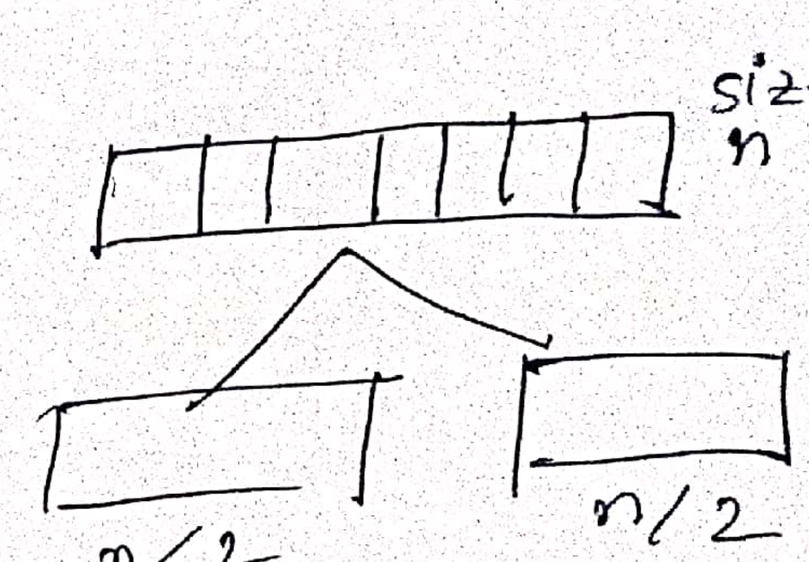
$$= 4T(n/4) + 2cn$$

$$= 4\left\{2T(n/8) + c\frac{n}{4}\right\} + 2cn$$

$$= 8T(n/8) + 3cn$$

$$\vdots$$

$$= 2^k T(n/2^k) + kcn$$



Let $2^k \to n$

$\therefore \log_2 n = k$

$= n T(n/n) + \log n . cn$

$= n T(1) + \log n . cn$

$= n \{ T(1) + c \log n \}$

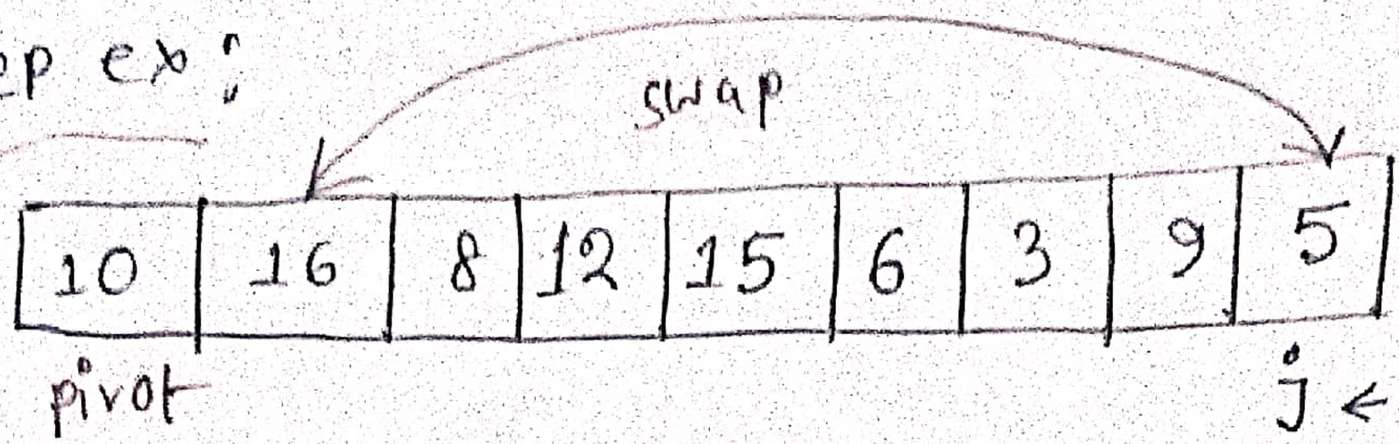
$= O(n \log n)$

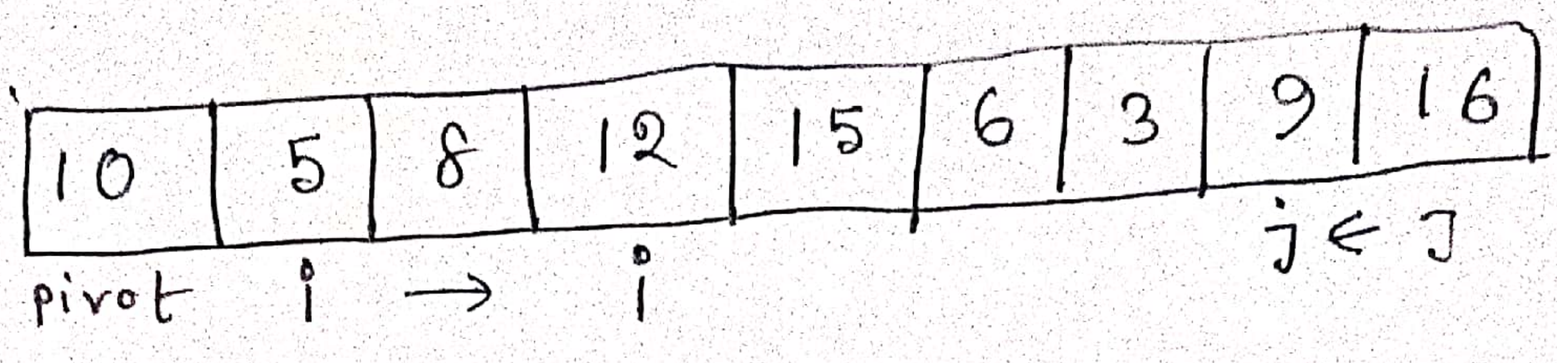
QuickSort

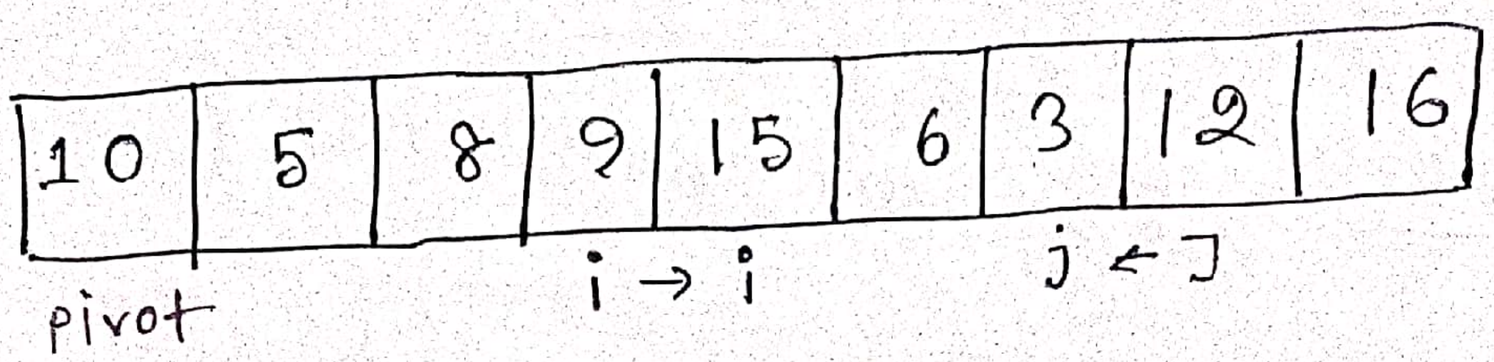
Pseudo:

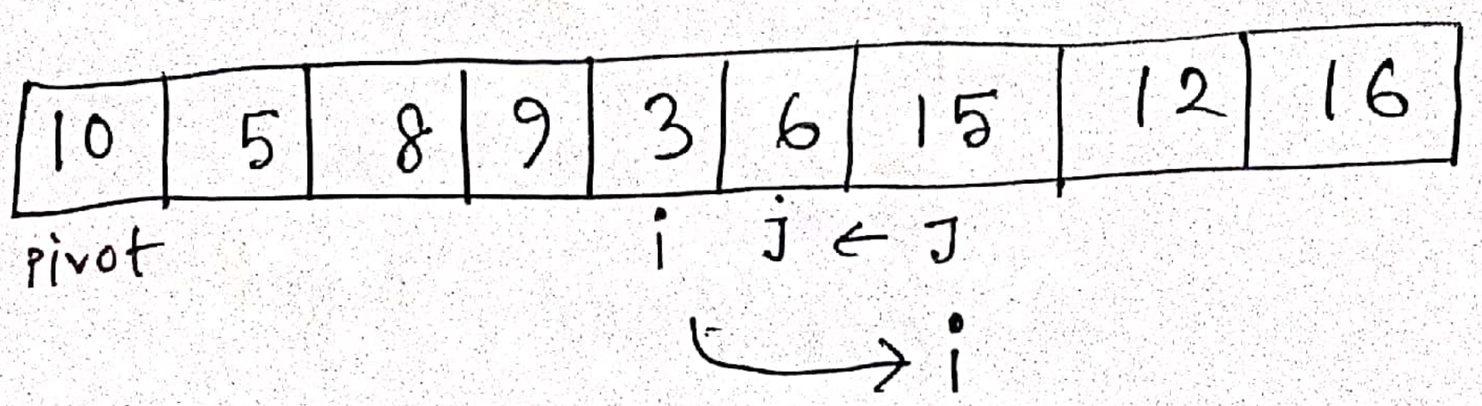
```
partition (l, h) {
    pivot = A[l], i = l, j = h
    while (i < j) {
        do { i++; } while (A[i] ≤ pivot);
        do { j--; } while (A[j] > pivot);
        if (i < j) swap (A[i], A[j])
    }
    swap (A[l], A[j])
    ret j;
}

Qsort (l, h) {
    if (l < h) {
        j = part (l, h)
        Qsort (l, j)
        Qsort (j+1, h)
    }
}
```

Step by step ex:

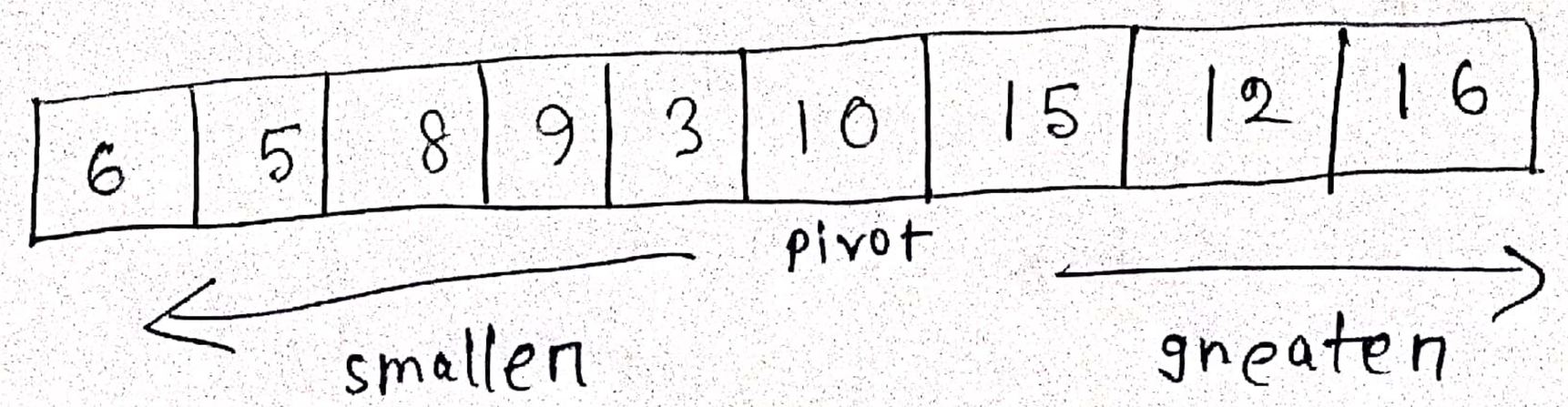








now i > J, swap with pivot



Time complexity analysis:

Avg — $O(n \log n)$

$$T(n) = T(i) + T(n-i-1) + cn$$ (divide into i and n-i-1 items)

$$= \frac{1}{n} \sum_{n=0}^{n-1} \{T(i) + T(n-i-1) + cn\}$$

$$= \frac{1}{n} \{T(0) + T(1) + \ldots T(n-1) + T(n-1) + T(n-2) + \ldots T(0)) + cn\}$$

$$= \frac{2}{n} \{T(0) + T(1) + \ldots T(n-1)\} + cn^2$$

$$\therefore nT(n) = 2\{T(0) + T(1) \ldots T(n-1)\} + cn^2$$

$n \to n-1$:

$$(n-1)T(n-1) = 2\{T(0) + T(1) + \ldots T(n-2)\} + c(n-1)^2$$

now, $nT(n) -$

$$(n-1)T(n-1) = 2T(n-1) + c(2n-1)$$

$$\approx 2T(n-1) + 2cn$$

$$nT(n) \approx (n-1)T(n-1) + 2T(n-1) + 2cn$$

$$= (n+1)T(n-1) + 2cn$$

divide by $n(n+1)$

$$T(n) = \frac{T(n-1)}{n} + \frac{2c}{n+1}$$

$$\Rightarrow \frac{T(n)}{n+1} - \frac{T(n-1)}{n} = \frac{2c}{n+1}$$

$$\Rightarrow \frac{T(n)}{n+1} + \frac{T(n-1)}{n} + \cdots \frac{T(1)}{2} - \frac{T(n-1)}{n} - \frac{T(n-2)}{n-1} -$$

$$\cdots \frac{T(0)}{1} = \frac{2c}{n+1} + \frac{2c}{n} \cdots \frac{2c}{2}$$

$$\Rightarrow \frac{T(n)}{n+1} - \frac{T(0)}{1} = 2c\left(\frac{1}{n+1} + \frac{1}{n} + \cdots \frac{1}{2}\right)$$

$$\Rightarrow \frac{T(n)}{n+1} = \frac{T(0)}{1} + 2c\left(H_{n+1} - 1\right)$$

$$\Rightarrow \frac{T(n)}{n+1} = 2c\left(\ln(n+1)\right)$$

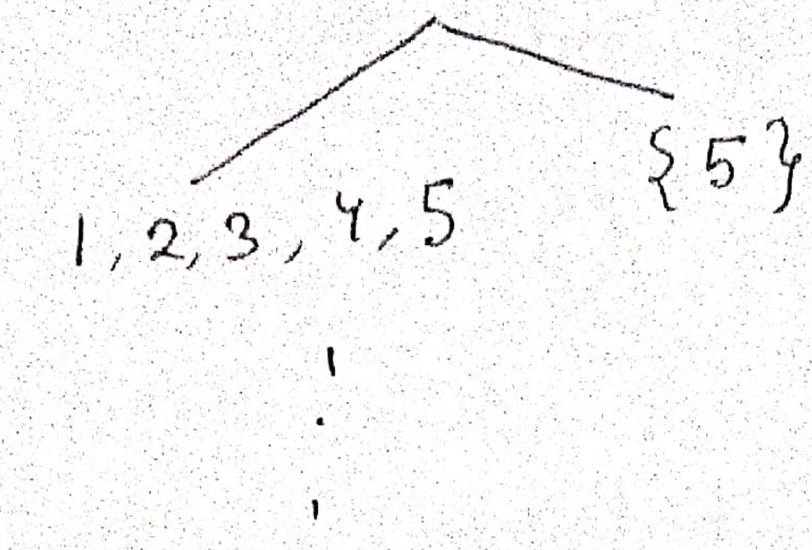
$$\Rightarrow T(n) = (n+1)\, 2c \ln(n+1)$$

$$\approx (n+1) \log_2 (n+1)$$

$$\approx n \log(n)$$ ∎

[worst — $O(n^2)$] — if already sorted

Ex: 1 2 3 4 5 (6) → pivot $T(n)$



Time Complexity Analysis:

$$T(n) = T(n-1) + cn$$
$$= T(n-2) + c(n-1) + cn$$
$$= T(n-2) + 2cn - c$$
$$= T(n-3) + c(n-2) + 2cn - c$$
$$= T(n-3) + 3cn - 2c - c$$
$$= \vdots$$
$$= T(n-k) + kcn - (k-1)c \dots - 2c - c$$

$n = k$

$$T(n) = T(0) + n^2c - c\{(k-1)+(k-2)+\dots 2+1\}$$
$$= T(0) + n^2c - c\,\frac{n(n-1)}{2}$$
$$= n^2 - \frac{n(n-1)}{2}$$

$\therefore T(n) = O(n^2)$

Heap sort

Time : O(nlog n) - {best/worst/avg} space : O(1)

Pseudocode: (using max heap)

```
heapify (arr, n, i)
{
    max = i
    leftchild = 2i + 1;
    rightchild = 2i + 2
    if (leftchild <= n && arr[i] < arr[leftchild])
        max = leftchild
    else max = i

    if (rightchild <= n && arr[max] < arr[right])
        max = rightchild

    if (max != i) {
        swap (arr[i], arr[max])
        heapify (arr, n, max)
    }
}
```

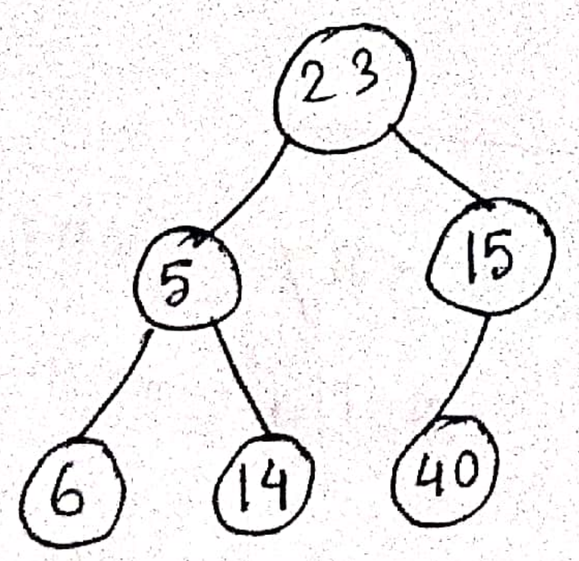
* create max/min heap
* delete node

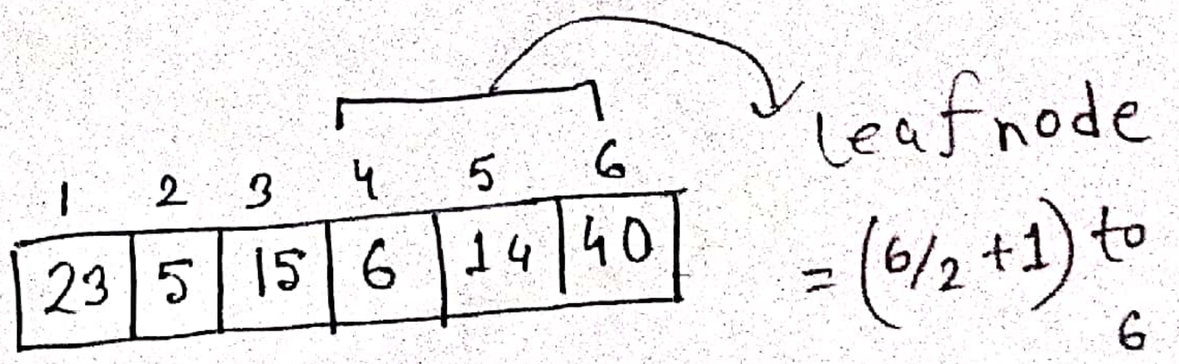
```
hsort (arr) {
    n = length (arr)
    for (i = n/2 → 1)   heapify (arr, n, i)
    for (i = n → 2) {
        swap (arr[1], arr[i])
        heapsize --;
        heapify (arr, i, 0)
    }
}
```

Step 1: Heapify method (create fast max/min heap) → $O(n \log n)$

- heapify don't work with leaf nodes

Leaf nodes : $\boxed{n/2 + 1 \text{ to } n}$

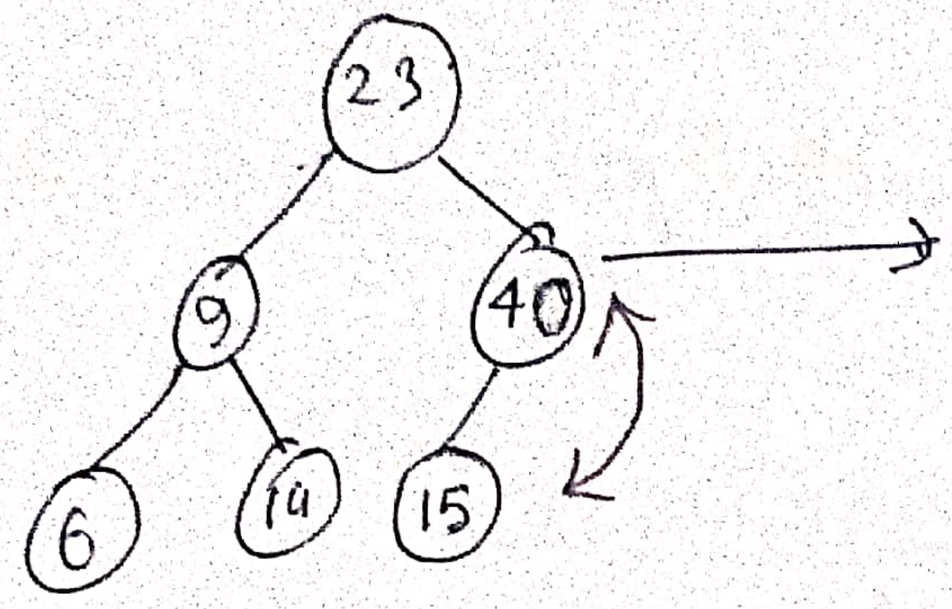




* start from last non-leaf node

idx = n/2 = 6/2 = 3

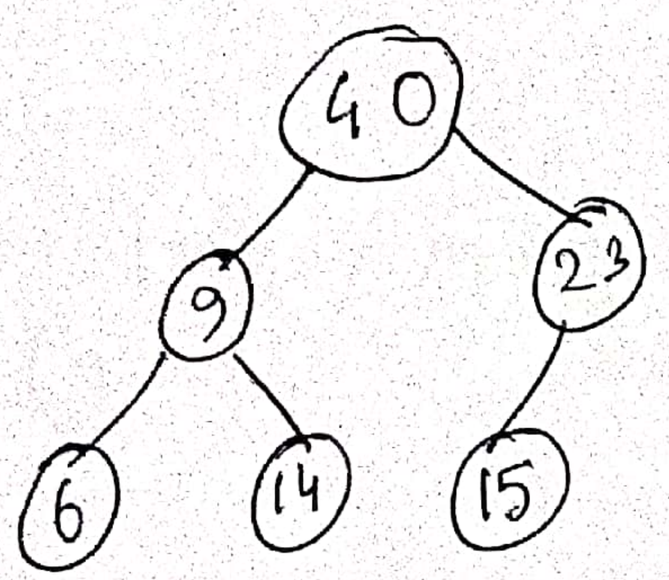
∴ a[3] = 15



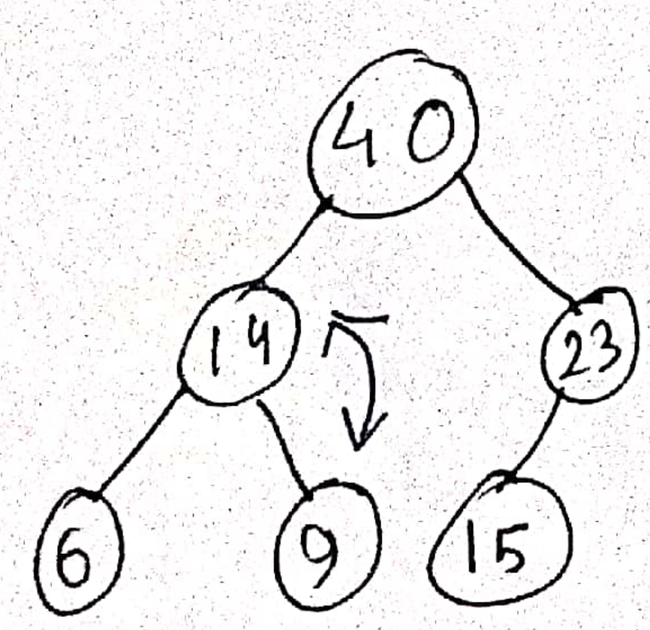
→ first choose this node

compare 23 and 40

swap with greaten child.



* now work with previous node (9)

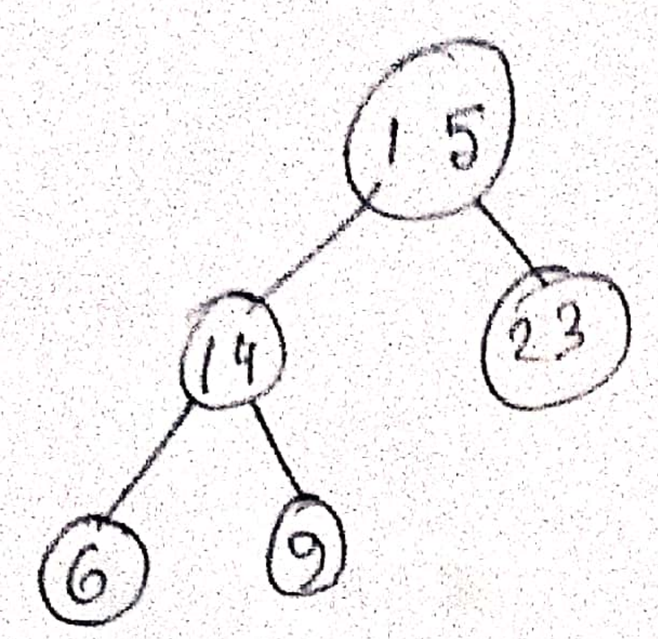


now, every parent > children

- max heap created

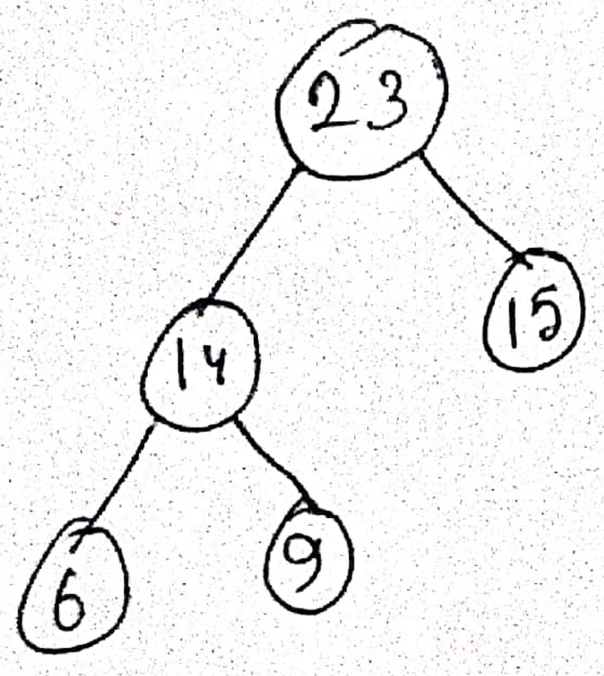
step 2 : heap sort (deletion) → O(nlogn)

– delete 40 and swap with last node (15)

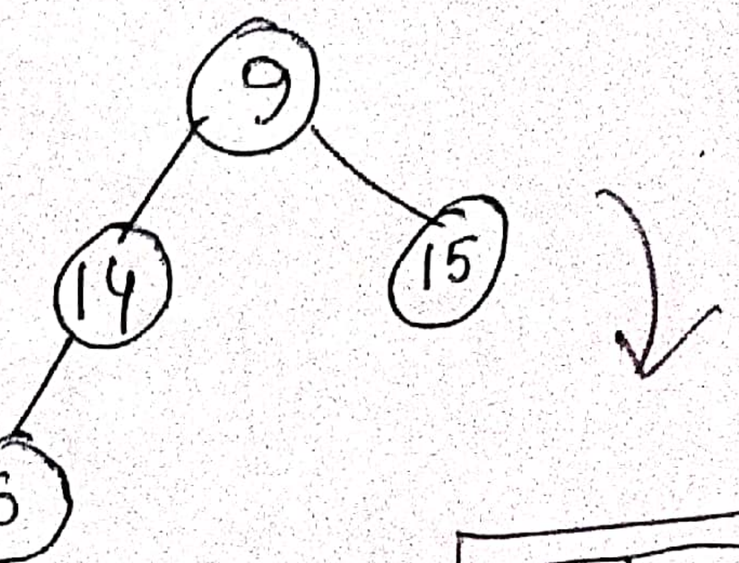


15	14	23	6	9	40

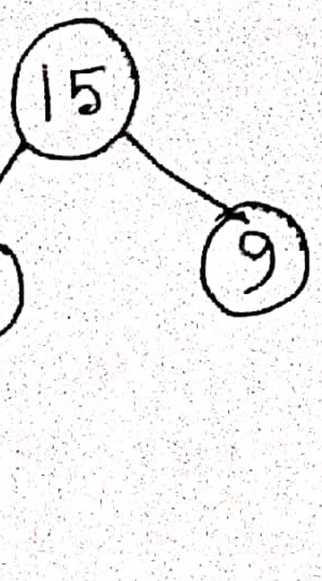
– now swap root (15) with greaten child (23)



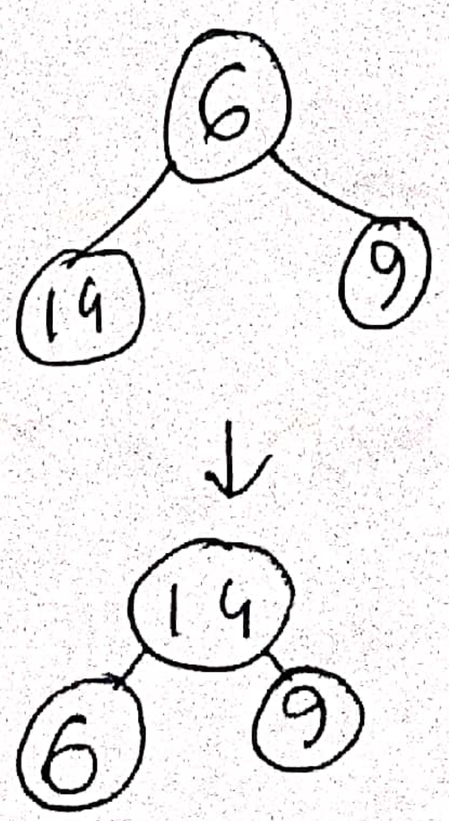
delete 23 →



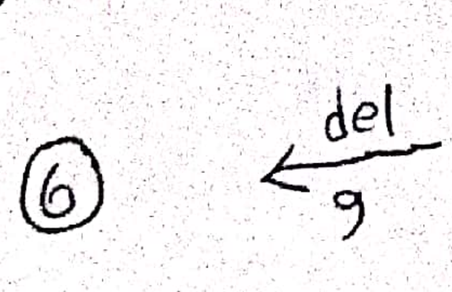
9	14	15	6	23	40

– 

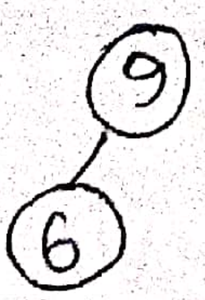
del 15 →



6, 14, 9, [15] [23, 40]



del 14 ←



del 9 ←

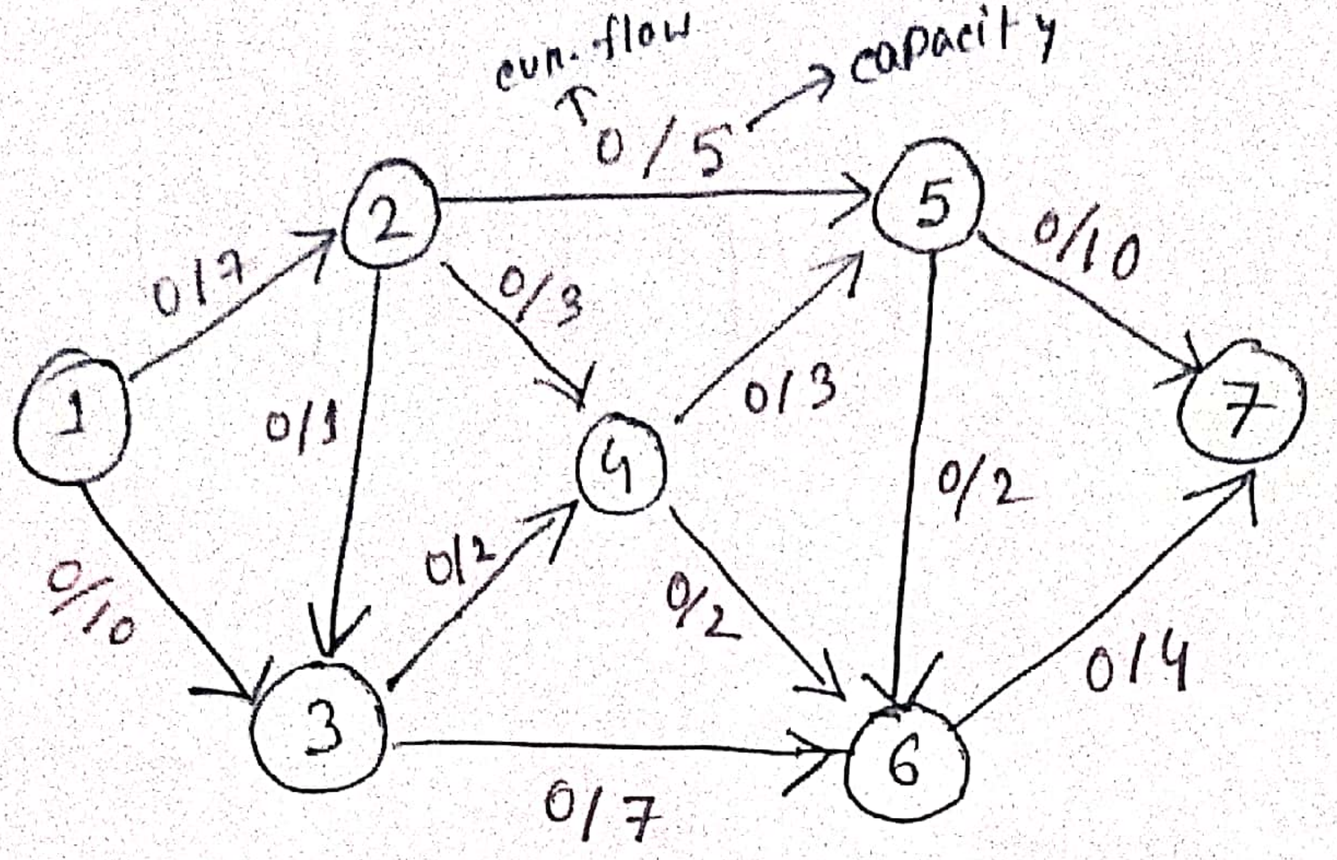
6, 9, 14, 15, 23, 40

9, 6 [14, 15, 23, 40]

visit

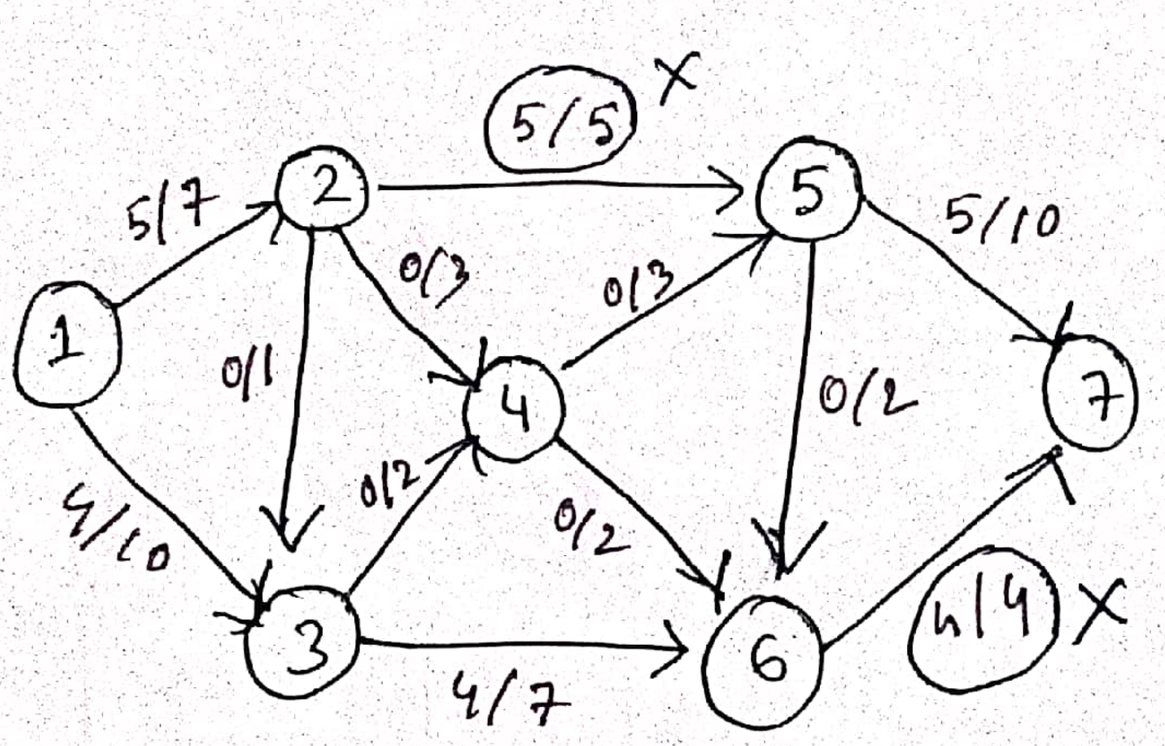
Ford Fulkenson

— max flow in a network

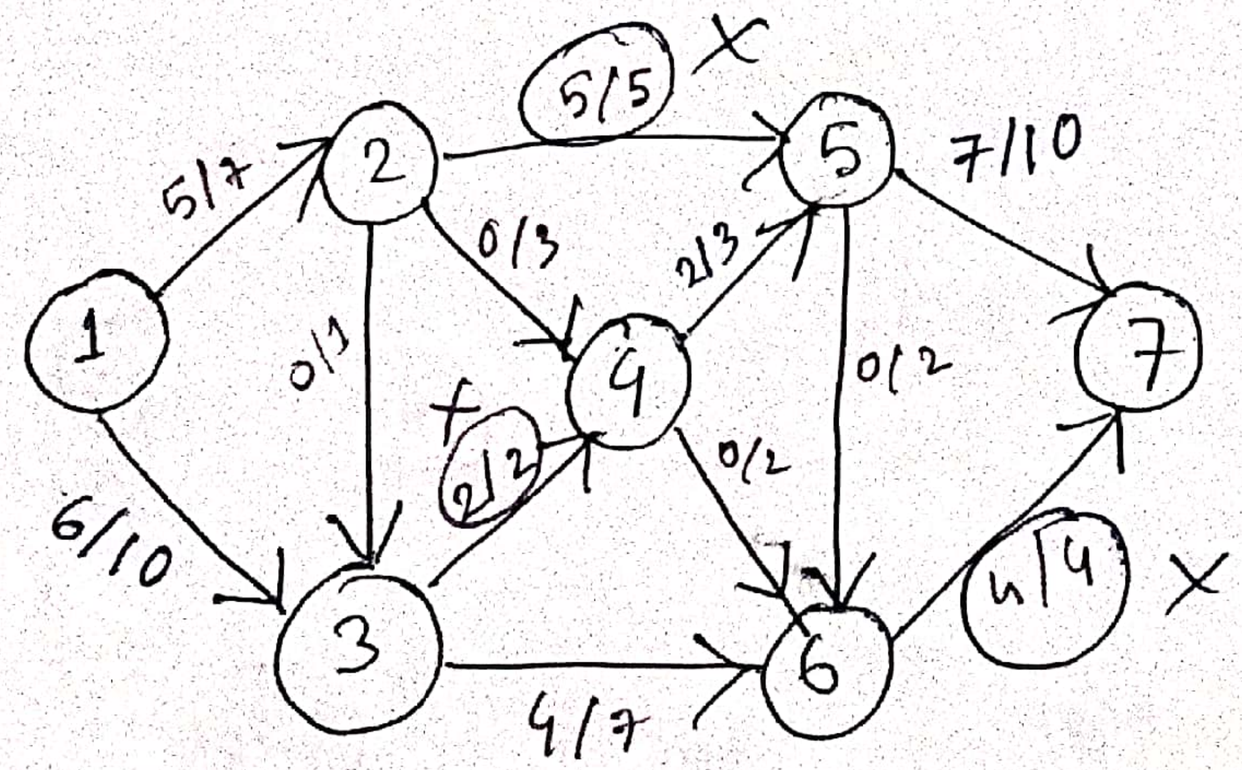


* 1 → 2 → 5 → 7 is an augmented path.

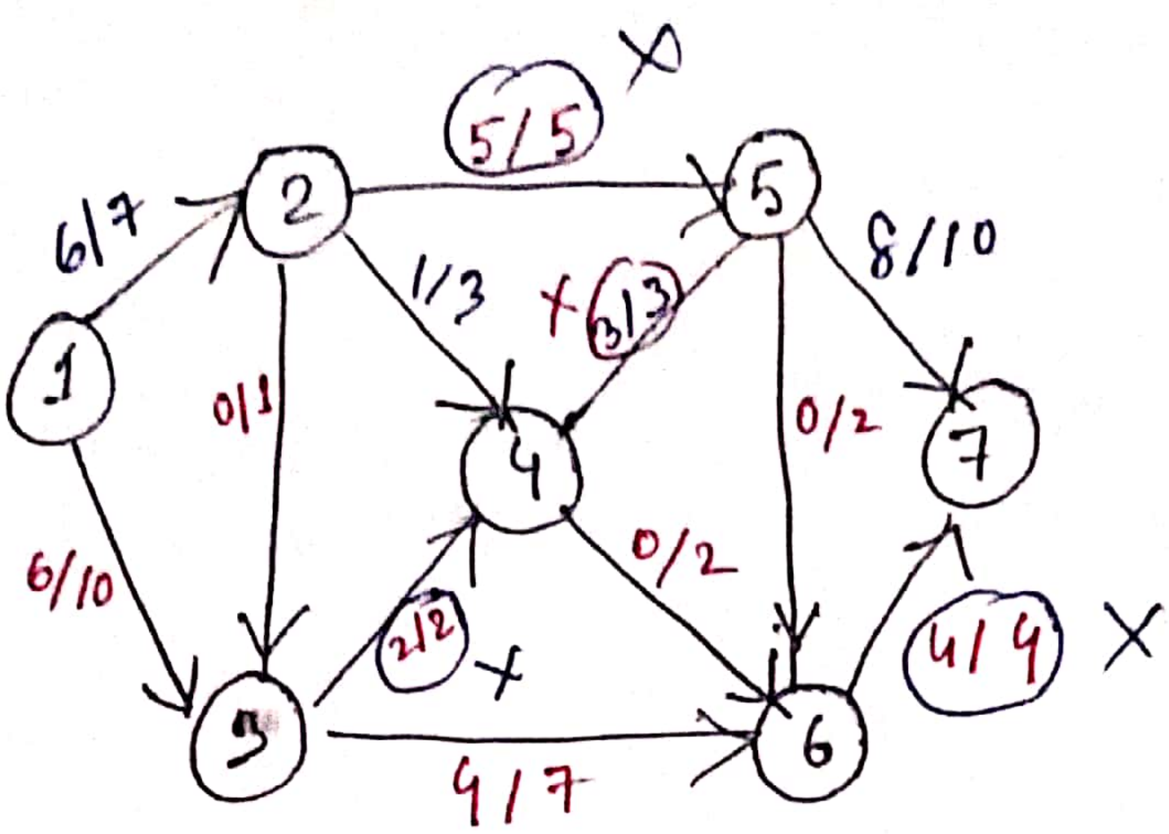
* min capacity = bottleneck "



Flow = 5 + 4 = 9



Flow = 9 + 2 = 11



Flow = 11 + 1
= 12

Now, no more path exist from source to sink.