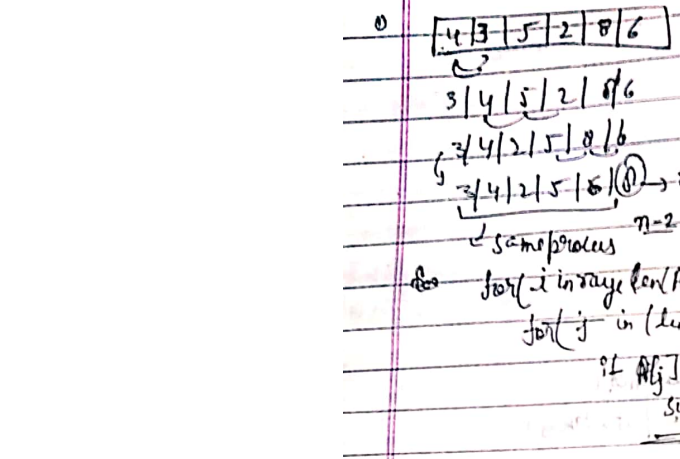
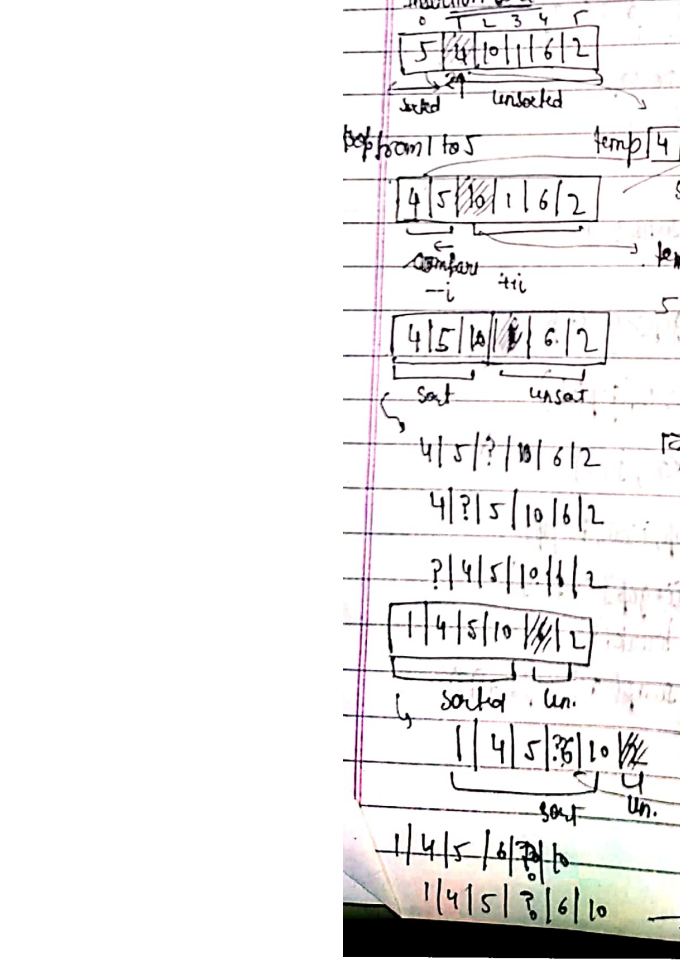
Sorting

Bubble sort



Insertion Sort



Algo Insertion

```
for(i=1; i<n, i++)
{
    temp = a[i];
    j = i-1;
    while(j >= 0 && a[j] > temp)
    { a[j+1] = a[j];
      j--;
    }
    a[j+1] = temp;
```

checked ↳ i=2
temp 10
j = 2-1 = 1
j≥0 ✓ && 5>10 ✗
a[2] = 10

4 5 10 1 6 2
j = 3-1 = 2
1<10 ✓

Short program

```
for(i in range(1, len(A)):
    j = i
    while(j>0 and ar[j-1]>a
        swap(j, j-1)
        j--
```

1 4 ? 5 6 10 → 1 2 4 5 6 10 ✓

Quicksort (Divide & Conquer)

(35) 50 15 25 80 20 90 45 (+∞) ← for stopping
↑ p ↑ q
pivot

p move right
stop at element ≥ pivot element > pivot

q move left
stop ele < pivot

(35) 50 15 25 80 20 90 45
p↑ ↑q ↑q
50 > 35 stop
90 < 45 ✗
20 < 35 ✓

check p, q crossed or not

35 20 15 25 80 50 90 45
p p q
15 > 35 p
25 > 25 ✗
80 > 35 ✓
50 < 35 ✗
80 < 35 ✗
25 < 35 ✓

p ⊇ q Yes so replace swap pivot with q

(25) 20 15 | (35) 80 50 90 45
p q
Quicksort recursion Quicksort

(80) 50 90 45
p p q

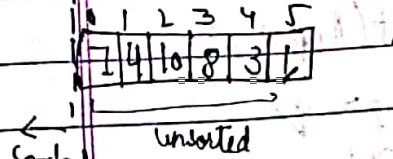
20 > 25 p
15 > 25 ✗
15 < 25 ✓
p ⊃ q Yes
x ⇆ q
15 20 25

80 > 80 p
90 > 80 ✓
45 < 80 ✓
80 50 45 90
p p q q
90 > 80 ✓
90 < 80 ✗
45 < 80 ✓
45 50 80 90

p → check p > x
q ← check q < x
after p ⊇ q x ⇆ q

```
Quicksort(A, p, q):
    if p >= q:
        return
    piv = A[q]
    pivindx = p
    for i in range(p, q):
        if A[i] < piv:
            swap(i, pivindx)
            p += 1
    swap(q, pivindx)
    Quicksort(A, p, pivindx-1)
    Quicksort(A, pivindx+1, q)
```

Selection Sort



find minimum and swap with first element

Pass 1:- 1 4 10 8 3 7 (sorted | unsorted)

1 3 10 8 4 7

1 3 4 8 10 7

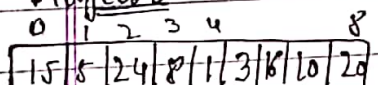
1 3 4 7 10 8

1 3 4 7 8 10 (sorted | uns)

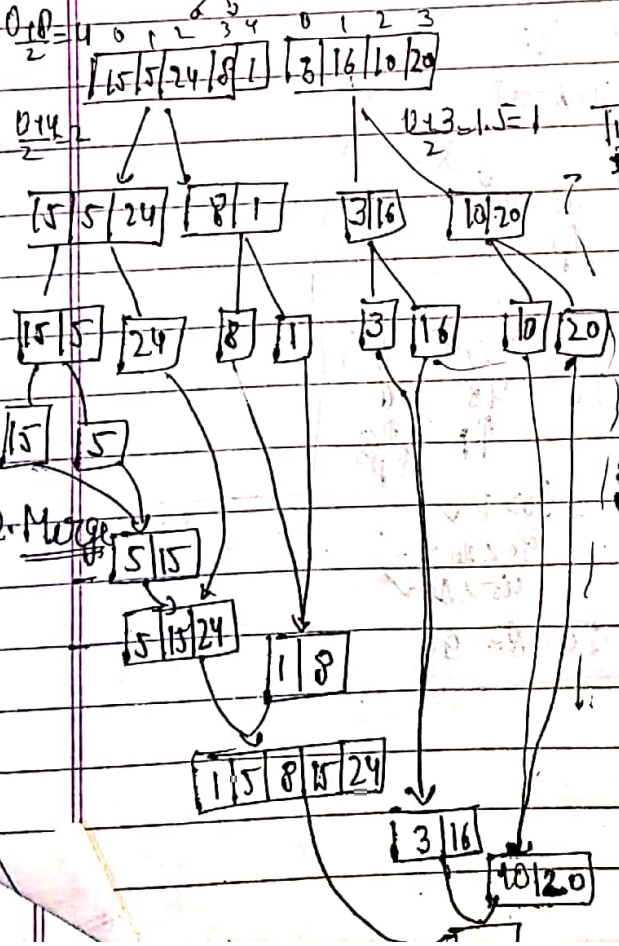
```
// loop for passes
// for find min
for(i=0; i<n-1; i++)
{ int min = i;
  for(j=i+1; j<n; j++)
  { if(a[j]<a[i])
    { min = j; }
    if(min != i)
    { swap(a[i], a[j]); }
  }
```

$O(n^2)$

Merge Sort :-



1. Divide




```
Merge sort(A, lb, ub)
{ if(lb<ub) → atleast two elements
  { mid = (lb+ub)/2;
    mergesort(A, lb, mid);
    mergesort(A, mid+1, ub);
    merge(A, lb, mid, ub); }
}
Merge(A, lb, mid, ub)
{ i = lb
  j = mid+1;
  k = lb;
  while(i<=mid && j<=ub)
  { if(A[i] <= A[j])
    { b[k] = a[i]
      i++, k++; }
    else { b[k] = a[j];
      j++, k++; }
```

Heap Sort

15	20	7	9	30

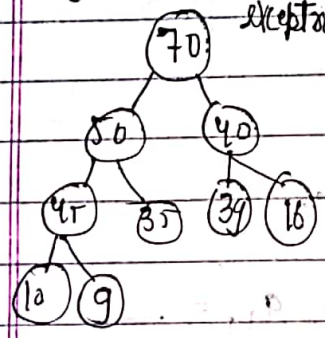
What is Heap?

Almost complete binary tree

← MaxHeap

for every node i, the value is less than or equal to its parent node value

$A[Parent[i]] \geq A[i]$ except root

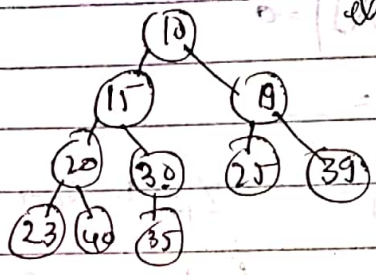


70	50	40	45	35	39	16	10	9

MinHeap

for every node, the value of is greater than or equal to its parent value

$A[Parent[i]] \leq A[i]$ except root



10	15	19	20	30	25	39	23	40	35

Insertion in Heap

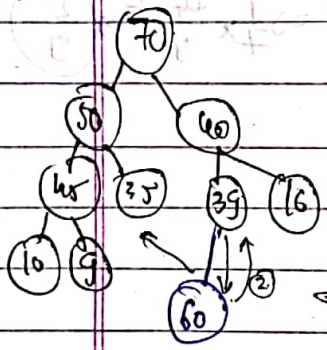
(last from)

```
insertheap(A, n, val)
{
  n = n+1;
  A[n] = val;
  i = n;
  while(i > 1)
  { parent = [i/2];
    if(A[i] > A[parent])
    {
      swap(A[p], A[i])
      i = parent;
    }
    else { return; } → means heap is OK
  }
```

Enter data = 60

should follow prop of complete binary tree

left child fill first



70	50	40	45	35	39	16	10	9	60

i = 10

Now check heap max property

(2) $60 \leq 39$ No, so swap

parent of $i = \frac{10}{2} = 5$

go to index 5

70	50	40	45	60	39	16	10	9	35

Check again with parent

$60 \leq 50$ No swap

$5/2 = 2$

A[2] = 50

70	60	40	45	50	39	16	10	9	35

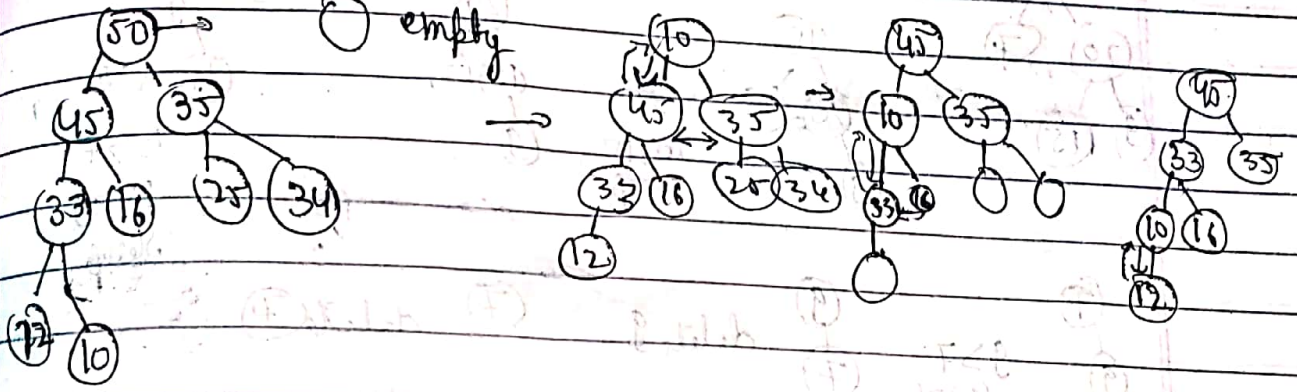
Max comparison

insertion = $O(\log n)$ → only height of tree

Delete Data from heap

Date : / /
Page No.

Ans:- only delete the root node



50	45	35	33	16	25	34	12	10

50 delete

10	45	35	33	16	25	34	12
1	2	3	4	5	6	7	8

find greater of child then compare that with parent & do swap

Sol left child at = $2i = 2 \times 1 = 45$.

right child at = $2i+1 = 3 = 35$

45 > 35

45 > 10

swap

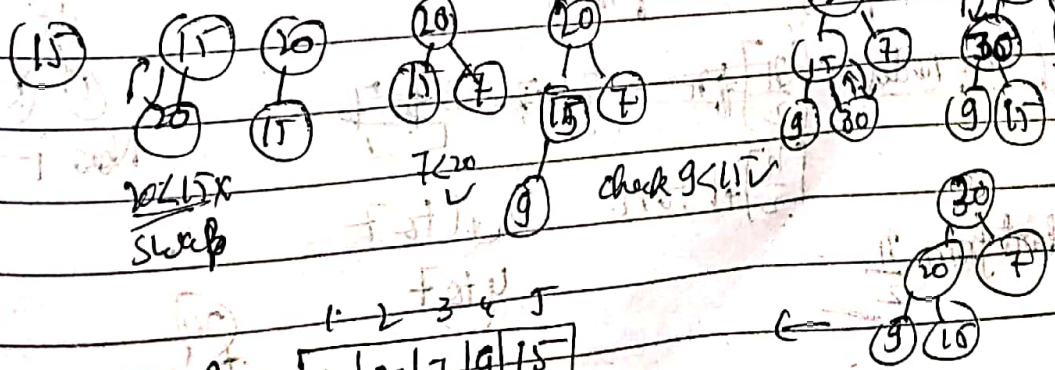
$i = 2$ → 45

33 > 10

swap

Heap Sort → Insert in heap then delete from heap

	1	2	3	4	5
A	15	20	7	9	30

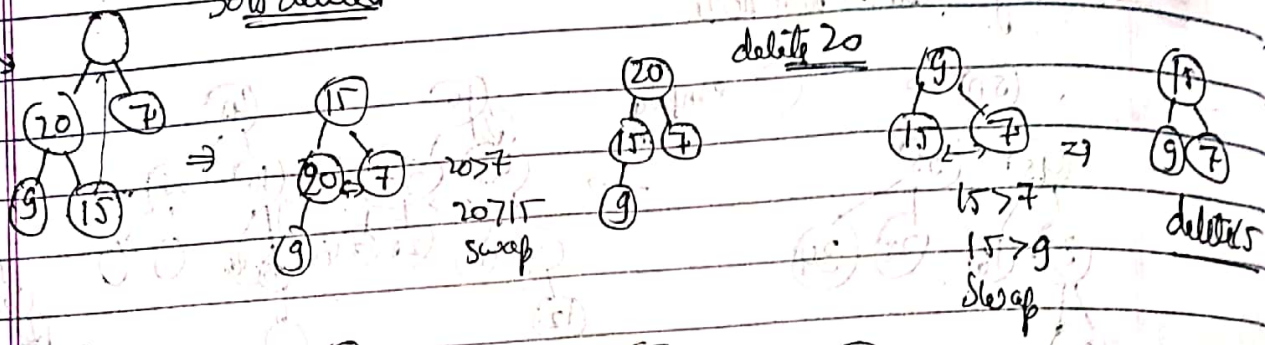


	1	2	3	4	5
A =	30	20	7	9	15

PTO.

Date : / /
Page No.

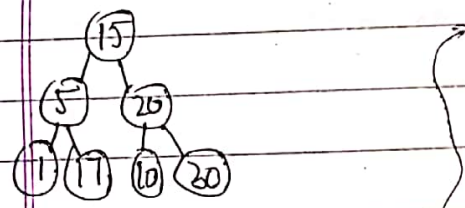
Now delete data from it (root is only deleted)



A = | 7 | 9 | 15 | 20 | 30 |

Time complexity → insertion ⇒ O(nlogn) — can be ⇒ O(n) by heapify method
deletion ⇒ O(nlogn)
O(2nlogn)
↳ O(nlogn)

Heapify method :-



A = | 15 | 5 | 20 | 1 | 17 | 10 | 30 |
1 2 3 4 5 6 7

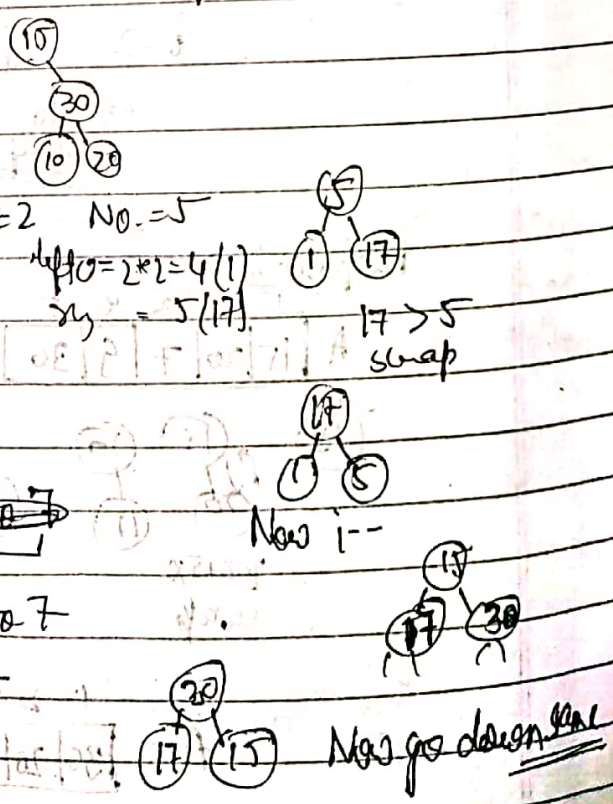
30 is child → leaf → leaf
same for 10, 17, 1

Now 20 — Apply only on non leaf nodes

No. of leaf nodes ⇒ $\lceil \frac{n}{2} \rceil$ to n ~~$\lceil \frac{7}{2} \rceil$ to 7~~

index $\lceil \frac{n}{2} \rceil + 1$ to n 7/2 + 1 to 7

we start from $\frac{n}{2}$ 4 to 7



Date : / /
Page No.

```
MaxHeapify(A, n, i)
{
    int largest = i;
    int l = (2*i);
    int r = (2*i)+1;
while(l<=n && A[l] > A[largest])   -> should be less than overflow index
    {
        largest = l;
    }
while(r<=n && A[r] > A[largest])
    {
        largest = r; }
if (i != largest)
    {
        swap(A[larg], A[i])
        heapify(A, n, largest);
    }
}

heapsort(A, n)
{
    for(i = n/2, i >= 1, i--)        -> build
    {
        Maxheapify(A, n, i);
    }
    for(i = n; i >= 1; i--)
    {
        swap(A[1], A[i]);
        Maxheapify(A, n, 1);
    }
}
```

Radix Sort → base (bucket sort)
→ no comparison

jenny > jiya
for no from

321 → least significant digit
most significant digit

15, 1, 321, 10, 802, 2, 123, 90, 109, 11

find max
find digit in max-d
Make all digit d digit no.

015, 001, 321, 010, 802, 002, 123, 090, 109, 011

Sort through least sig digit

Pass 1

0:	010, 090
1:	001, 321, 011
2:	802, 002
3:	123
4:	
5:	015
6:	
7:	
8:	
9:	109

Now at 2nd place

After Pass One:- 010, 090, 001, 321, 011, 802, 002, 123, 015, 109

Pass Second:-

0:	001, 802, 002, 109
1:	010, 011, 015
2:	321, 123
3:	
4:	
5:	
6:	
7:	
8:	
9:	090

001, 802, 002, 109, 010, 011, 015, 321, 123, 090

0:	001, 002, 010, 011, 015,
1:	109, 123,
2:	
3:	321
4:	
5:	
6:	
7:	
0:	802
9:	

Pass three:- 001, 002, 010, 011, 015, 090, 109, 123, 321, 802

No. of passes :- length of max digit

Time Complexity → $O(d*(n+b))$
d → digit, (n+b) → total, b → base

$= O(3(10+b)) \approx O(n)$

Shell Sort

Date : / /
Page No.

C in insertion sort

23 29 15 19 31 7 9 5 2

Sorted — Unsorted

7 needs comparisons
but if directly compared with 23
then no more comparisons
that is Shell sort

23 29 15 19 31 7 9 5 2

gap

gap = [N/2] or any other value

gap2 = gap2/2 = N/4

N/8

0	1	2	3	4	5	6	7	8
23	29	15	19	31	7	9	5	2
i						j		

gap = 9/2 = 4

23 > 31 No swap

29 < 7 swap

23 7 — 1 29

do more

After this → only in end

Compare i with i-4

and swap

j = n-1 now stop pass 1

gap = 4/2 = 2

Pass 2

2 7 9 5 23 29 15 19 31

i j

2 5 9 7 23 29 15 19 31

i j

2 5 9 7 15 29 23 19 31

i → i j → j

15 with 9 no swap

Main thing

gap = 5

gap ↓

if gap = 1 → insertion sort

So in last step it is same as it

2 5 9 7 15 19 23 29 31

i → i j → j

Pass 2 — 7, 19 no swap

Now j = n-1

gap = 2/2 = 1

Now normal insertion sort

Pass 3 —

gap = 1/2 ⇒ 0 gap ≥ 0 stop.

```
for(gap = n/2 ; gap>0 ; gap/=2)
{
  for(j=gap ; j<n ; j++)
  {
    for(i=j-gap ; i>=0 ; i-=gap)
    {
      if(a[i+gap] > a[i])
        { break; }
      else { swap(a[i+gap], a[i]) }
    }
  }
}
```

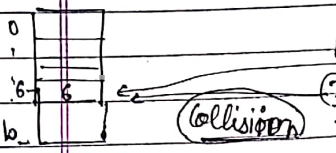
Date : / /
Page No.

Hashing → used to search (constant time not dependon n)

Types of Hashing :-

Key = 2, 26 $h(k_i) = k_i \% m$

6 % 10 = 6 store at 6



To resolve this

Types

① Open Hashing (Closed Addressing)

② Closed Hashing (Open Addressing)

- Chaining
- linear probing
- quadratic pb.
- Double hashing

Counting Sort :-

1 0 2 1 0 1 1 5 6 7 5 4 2 2 0 0 1

n = 17

0 - 7 ⇒ K = 7 max Range ⇒ 0 - K

① Case $0 \le a[i] \le K$

② Case ⇒ $a[i] \in I$ (No. should be +ve)

a

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

n = 17
K = 9

Count size = K+1

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

count of 2

as Simple a that

```
for(int i=0; i<n; i++)
{
    ++ Count[a[i]];
}
```

* Start | 0 | 0 | 0 | 0 | 0 | ---

0	0	0	0	0

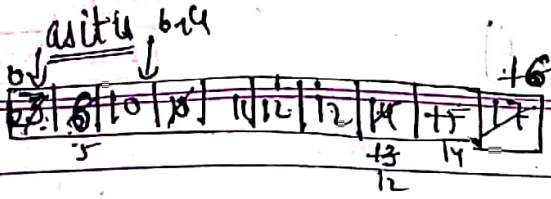
0	1	1	---

0	2	1	---

This type

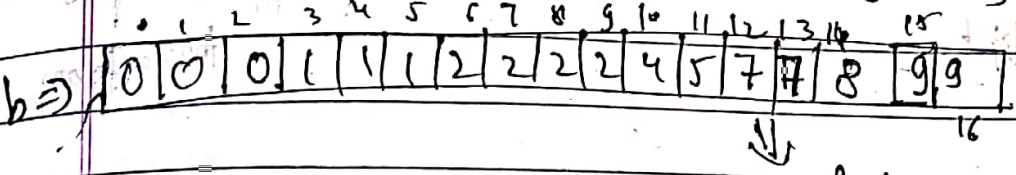
Date: / /
Page No:

update countarray




```
for(int i=1; i<=k; i++)
{ count[i] = count[i] + count[i-1];
```

to find → Measure of Stability




```
for(i=n-1; i>=0; i--)
{
  b[--count[a[i]]] = a[i];
}
```

to copy to original

```
for(i=0; i<n; i++)
{ a[i] = b[i];
}
```

Steps:-

① Make countarray of main list → all are 0 in start
② Update to list [6|4|5] → [6|10|15]
③ New array
④ Copy

Time Complexity

↳ n+k+n+n
⇒ 3n+k
↳ O(n+k)

drawback: k should be feasible
if array size = 100
k = 10000 &
n
↳ no value float

Next Radix Sort program

↳ trey