SESSION - 7

(SORTING)

DATE
PAGE

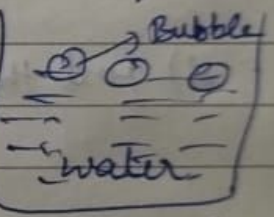
- Arranging in some orders
  - increasing / decreasing
    ascending / descending
  - Non-increasing / non increasing — 10, 10, 7, 5, 3
  - 5, 7, 10, 10, 20, 40

- Algos:
  - Bubble Sort
  - Selection Sort
  - Insertion "
  - Merge "
  - Count "
  - Quick sort
  - heap "
  - Tim "
  - Radix "

* Bubble Sort –

- Highest element will reach the end first.

eg: 7, 9, 6, 12, 10, 20, 29, 5, 3, 2



1st pass → 7, 6, 9

→ 7 6 9 10 12 20 5 3 2 | 29 → Highest element

2nd pass → 7 6 9 10 12 5 3 2 | 20 29 → Highest in 2nd pass

3rd pass → | 12 20 29

⋮

→

```
void bubbleSort(int ar[], int N)
{
  for(int i=0; i<N; i++){
  for(int j=0; j<(N-i-1); j++){
    if(ar[j] > ar[j+1]){
      swap ar[j] to ar[j+1].
    }
  }
}
```

Worst Case $(N^2, 1)$

→
```
for(int i=0; i<N; i++){
    bool flag=false;
    for(int j=0, j<(N-i-1); j++){
        if(ar[i]> ar[j+1]){
            //swap
            bool flag=true;
        }
        if(!flag) break;
    }
}
```

Best case (N, 1)

(*) **Selection Sort:** 1) Find the smallest element & place it at 0^{th} position in 1^{st} pass. 1^{st} position in 2^{nd} pass
2^{nd} " " 3^{rd} pass - - - and so on

OR

2) Find the largest element & place it at $(n-1)^{th}$
- - - - $(n-2)^{th}$
- - - $(n-3)^{th}$

eg. 20, 24, 7, 16, 5, 2, 31, 23

swap

2 | 24 7 16 5 20 31 23

2 5 | 7 16 24 20 31 23

```
void selectionSort(int ar[], int N)
{
    for(int i=0; i<N; i++)
    {
        int element=ar[i], index=i;
        for(int j=0; j<N; j++)
        {
            if(ar[j]<element)
            {
                element=ar[j];
                index=j;
            }
        }
        swap(ar[i], ar[index]);
    }
}
```

$(N^2, 1)$

(*) **Insertion Sort:-** - Shifting things (insert at right position)
- UNO in cards

eg.

20 3 12 17 5 16

Assume 1st card is always sorted

ele=3

20 20 12 17 5 16

3 | 20 12 17 5 16

```
void insertionSort(int ar[], int N)
{
    for(int i=1; i<N; i++) — N
    {
        int ele=ar[i]; j=i-1;
        while(j>=0 && ar[j]>ele) — N
        {
            ar[j+1]=ar[j];
            j--;
        }
        ar[j+1]=ele; — after first right place
    }
}
```

$(N^2, 1)$

- (N, 1) in Best case.

(*) Sorted array- 5, 20, 25, 40, 70

30 → new element to insert

→ insertion sort (if already sorted & wanted to insert any no).

(we can add this at last index then do insertion sort :)

Elections – 10 Candidates

1 2 3 4 5 6 7 8 9 10

people

votes: 2 3 9 3 7 1 1 7 3 4 3

1 → 2 votes | 4 – 1 vote
2 → 1 vote | 5 – 0 "
3 → 4 votes | 6 – 0 "

} winner → 3rd Candidate

– print winner as o/p.

Solutions

1) (10×N, 1) → BF
N×M (candidate) ↳ close to N^2

2) (N+10, 11)

```
void winner( )
{
  int winner=1, votes=0;
  for(int i=1; i<=10; i++)        // 10
  {
    int count=0;
    for(int j=0; j<N; j++)        // N
    {
      if(ar[j]==i)
      {
        votes = count
        count++;
      }
    }
    if(count > votes)
    {
      votes= count
      winner= i;
    }
  }
  return winner;
}
```

(10×N, 1)

Count array: 0 1 2 3 4 5 6 7 8 9 10
0 0 0 0 0 0 0 0 0 0 0

votes: 2 3 9 3 7 1 1 7 3 4 3

– n^{th} cell in count array → stores the value (votes) of n^{th} people

max = 5 votes
index = 3

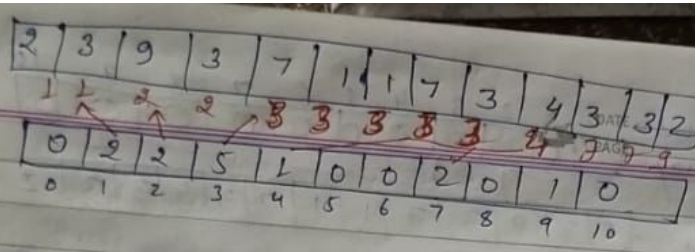
② take count array & update the vote count and then find maximum.

```
→ void winner( )
{
  int count[11] = {0}   → initialise all element 0
  for(int i=0; i<N; i++)
  {
    count[ar[i]]++;
  }
  int max=0; winner=1;
  for(int i=0; i<=10; i++)
  {
    if(max < count[i])
    { max = count[i];
      winner = i;
    }
  }
  return winner;
}
```

(N+10, 11)
for m candidates
↓
(N+M, M+1)

→ Using count array can I sort votes array?

o/p → 1 1 2 2 3 3 3 4 7 7 9



→ int K=0;

10 ← for (int i=1; i<=10; i++)
{
n cells we are filling { for (int j=0; j< cnt[i]; j++)
{
ar[K++] = i;
}
}
↳ inserting the increasing

(N+10) → Count sort
↳ Since its having linear TC so best algorithm theoratically for sorting

TC ⇒ (N+M) } M is no of Candidate
SC ⇒ M

(*) B.S.I → (N^2, 1)
CS → (N+M, M)
↳ linear

1 → Negative Numbers
2 → large Numbers
3 → Range is too large
} Problems with this

1 → (-3) 2 7 6 1 -3 2 2 4
don't have -ve index in count array

$-3 \le ar[i] \le 7$
↳ maximum
So → 0th index → -3
1st " → -2
2nd " → -1
3rd " → 0
4th index → 1
5th " → 2
!
10th " → 7
min

$a \le ar[i] \le b$

cnt [ar[i]-a]
0

$-a = (-a) = 0$

→ Negative problem resolve by this.

2 → Large No. $10^9 \le ar[i] \le 10^9+100$
a b

10^9 → 0th index
10^9+1 → 1st index
!
!

[a, b]
= b-a+1
$= 10^9+100-10^9+1$
size of Count array = 101

— this problem also solve.

(*) $0 \le ar[i] \le 10^{10}$
cnt [$10^{10}+1$] · Range is too large
↳ MLE (10^{6-7} for int)
If range is too large, Count array fails.

(*) Points 1) Range R → (b-a+1), [a, b]
2) cnt[R]
3) cnt [ar[i]-a] ++
4) find result for O/P

(*) $0 \le ar[i] \le 10^5$

| -2 | 2 | 0 | 1 | 3 | 99999 |

with BS → $N^2 = 6^2 = 36$
CS → cnt[99999+1] — No, because Memory will be wasted.

X N << R

(*) A_N : 3 5 9 16 21 40
B_M : 7 12 20 21 50

Given 2 sorted array. Print the all data in sorted order.

1) Brute Force: $C_{N+m} = \underbrace{(N+M)}_{\text{copy A \& B to C array (new array)}} + \underbrace{(N+M)^2}_{\text{sorting any of B/S/I}} + \underbrace{(N+M)}_{\text{print array}}, \underbrace{(N+M)}_{\text{space for additional C array.}}$

2) C_{N+m} : $N + (N*M) + (N+M), (N+M)$
- N → copy A in C array
- $(N*M)$ → Insertion step
- $(N+M)$ → print C array
- $(N+M)$ → SC for array

3) A_N : 3 5 9 16 21 40
B_M : 7 12 20 21 50

P_1 on 3, P_2 on 7

$A[P_1] < B[P_2]$ → Print minimum & (P_1+1) move pointer

2 Pointer Technique

Code:-
```
void printArray(int A[], int N, int B[], int M)
{
  int P1=0, P2=0;
  while(P1<N && P2<M)
  {
    if(A[P1] < B[P2])
    {
      print(A[P1++]);
    }
    else { print → B[P2++];
    }
  }
```

```
  while(P1<N)
  {
    print → A[P1++];
  }
  while(P2<M)
  {
    print → B[P2++];
  }
}
```
printing remaining elements of A or B after any 1 pointer reaches end.

TC → $(N+M)$ steps overall
SC → 1

(*) A_N : 2 4 12 17 20 - - - - -
B_M : 5 13 16 21 40

Merge A & B (merge B array into A array in such a way that A will remain sorted).

O/P → A → 2, 4, 5, 12, 13, 16, 17, 20, 21, 40

Solutions- 1) $(M + N^2, 1)$
- M → copy from B to A
- N^2 → perform sort

2) $(M*N, 1)$ or $(M*(N-M), 1)$
- $(M*N)$ → Insertion step

3) C_N :
A_N :
B_M :
- 2 pointer technique -

$(N+N, N)$ → copying C to A

2 pointer on A & B to move to C

4) Reverse Pointer & match greater element and place at the end.

$(M + (N-M), 1)$

↳ $(N, 1)$ → 2 Pointer app

Code:

```
int P1 = N-M-1, P2 = M-1, P3 = N-1;
while (P1 >= 0 && P2 >= 0)
{
  if (A[P1] > B[P2])
  {
    A[P3] = A[P1];
    P3--;
    P1--;
  }
  else
  {
    A[P3] = B[P2];
    P2--;
    P3--;
  }
}
```

Here, no element will be left to compare. In 'A' element might left to compare but that is already sorted.

⊛ Ar_N :

0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	1	0	1	0	1	0	1	0	0	1	1

Only 0's & 1's are given.

O/P → 0 0 0 0 0 0 1 1 1 1 1 1 1

2) 0's = 5, 1's = 7

0^{th} index = 5

1^{th} " = 7

Count Sort

$(N + (N+2), 2)$

Soln

1) $N^2, 1$ (BS/SS/IS)

2) 0's = 5, 1's = 7 (outer loop to fill 0's & 1's)

$(N + (N+2), 2)$ ↳ Count Sort

Fill Count array

CS 3) $(N, 1)$ 2P

[SESSION-8]

3) 2 Pointer → Never jump multiple step at a time.
- One step at a time.

```
int P1 = 0, P2 = N-1;
while (P1 < P2) {
  if (ar[P1] == 0) {
    P1++;
  }
  else if (ar[P2] == 1) {
    P1--;
  }
  else if (ar[P1] > ar[P2]) {
    // swap(ar[P1], ar[P2]);
    P1++; P2++;
  }
}
```

$(N, 1)$

⊛ ar_N : 3 10 12 5 17 18 5 30 25

→ # pairs 1) i < j
2) ar[i] > ar[j]

O/p - e.g (3, -5)
i < j
ar[i] > ar[j]

3 → 1 (5 → 0)
10 → 2 (17 → 1, 18 → 1, 30 → 1, 25 → 0)
12 → 2

total 8 pairs O/P

1) BF

```
int ans = 0;
for (int i = 0; i < N; i++)
{
  for (int j = i+1; j < N; j++)
  {
    if (ar[i] > ar[j])
    {
      ans++;
    }
  }
}
return ans;
```

Soln

1) $(N^2, 1)$

2) Merge Sort $(N \log_2 N, N)$

```
void merge(                    );
{
(same code as merge sort)

}
```

Inversion Count -

min <= pairs <= max ← Sortedness we can decide whether asc or desc

0 <= pairs <= N(N-1)/2

ascending
5 12 20 40 → 0

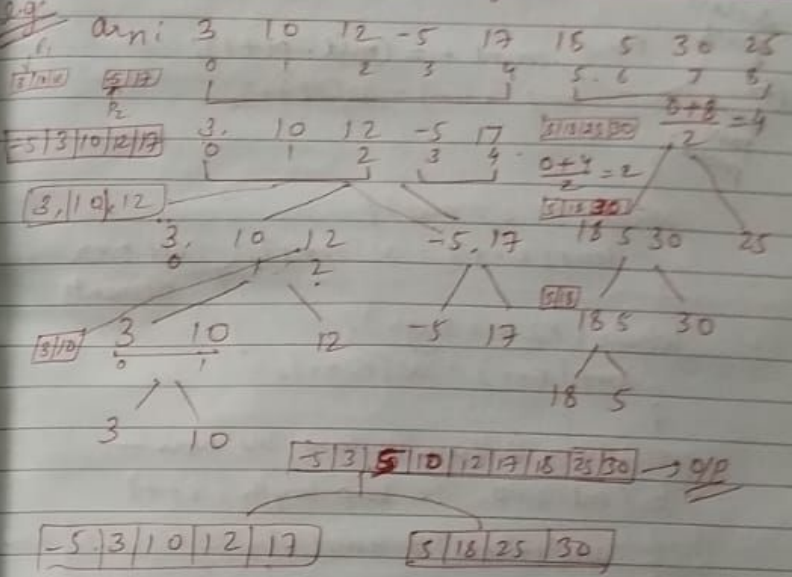
descending
40 20 12 5 → 3+2+1

Merge Sort - * Divide & Conquer Technique

→ Divide the array into sub arrays until its possible & then merge it back.

e.g.

arr: 3 10 12 -5 17 15 5 30 25




```
void MS(int ar[], int low, int high)
{
    if(low==high) return;
    int mid=(low+high)/2;
    MS(ar, low, mid);      // left
    MS(ar, mid+1, high);   // right
    merge(ar, low, mid, high);
}
```

T(N): T(N/2) + T(N/2) + N

```
void merge(int ar[], int low, int mid,
           int high)
{
    int P1=low, P2=mid+1;
    int temp[high-low+1];
    int k=0;
    while(P1<=mid && P2<=high)
    {
```

```
if(ar[p1] <= ar[p2])
{
    temp[k++] = ar[p1++];
}
else {
    count += (mid - p1 + 1);
    temp[k++] = ar[p2++];
}
}
```

Declare global (for count pair problem)

```
while (p1 <= mid) {
    temp[k++] = ar[p1++];
}
while (p2 <= high) {
    temp[k++] = ar[p2++];
}
```

left over elements

```
// everything is there in temp by now —
for (int i=0; i< temp.length; i++)
{
    ar[i+low] = temp[i];
}
}
```

→ $T(N) = 2T(N/2) + N = N \log N$

$(N \log_2 N, N) =$ MergeSort

* Master Theorem (simplified) →

$T(N) = aT(N/b) + N^c$ where $t = \log_b a$, $a>0, b>1$

1) $t < c \rightarrow O(N^c)$
2) $t = c \rightarrow O(N^t \log N)$ or $O(N^c \log N)$
3) $t > c \rightarrow O(N^t)$

Eg: $T(N) = 2T(N/2) + N$

$a=2, b=2, N^c = N \Rightarrow c=1$

$t = \log_b a = \log_2 2 = 1$

$\therefore t=c \Rightarrow N^t \log N = N \log N$

⊛ ar_N: | -5 | 12 | 6 | 9 | 20 | 19 | 30 |

1) K=15, 2) K=25, $[i \neq j]$, $[ar[i] + ar[j] = K]$

O/P => true / false

↳ 6+19 = 25

-5+30 = 25

Sol^ns

1) $(N^2, 1) \rightarrow$ BF
2) $(N \log N + N, N)$ → MS
   Sorted using MS → 2P

→ 1)

```
for(int i=0; i<N; i++)
{ for (int j= i+1; j<N; j++)
  {
    if(ar[i]+ar[j]==K)
      return true;
  }
}
return false;
```

2) Sorted using Merge –

p1 ↑ , p2 ↓

| -5 | 6 | 9 | 12 | 19 | 20 | 30 |

-5+30 = 25 > 15 (Move P2)

-5+20 = 15 = 15 (Found)

For K=31 -5+30 = 25 < 31 (Move P1)

ar[P1] + ar[P2] < K ↳ Move P1

ar[P1] + ar[P2] > K ↳ Move P2

ar[P1] + ar[P2] == K ↳ Found

→ Code:

```
// sort the array using MS
int P1=0; P2= N-1;
while (P1<P2) {
  if(ar[P1]==ar[P2])
    return true;
  else if (ar[P1] + ar[P2] < K) {
      P1++;
  }
  else if (ar[P1] + ar[P2] > K) {
      P2--;
  }
return false;
```

(*) a-b=k | -5 16 20 29 30

P1 k=9 P2

16-(-5)=21>K

Not able to judge which to move both side decreasing diff.

(*) a+b+c=K

1) BF - $(N^3, 1)$

2) a+b=K-C → Outer loop to fix C & inside loop use 2 pointer for C+a+b=K

$(N \log_2 N + N^2, N)$

MS

(*) Linear Search -

20 40 -60 7 25 41

K=7 Found

TC → (N, 1)

(*) Binary Search - search space [ordered elements]

eg -5 40 20 36 29 15

K=29 ↓ sorting

-5 15 20 29 36 40

0 1 2 3 4 5 $\frac{0+5}{2}=2$

mid

ar[mid] < K → right
ar[mid] > K → left
ar[mid] == K → true

// iterative Approach

```
bool binarySearch(int ar[], int K, int N)
{
  int low=0; high=N-1;
  while(low<=high){
    int mid=(low+high)/2;
    if(ar[mid]==K) return true;
    else if(ar[mid]>K) high=mid-1;
    else low=mid+1;
  }
  return false;
}
```

(*) // Recursive Approach

```
bool searchBinary(int ar[], int N, int low, int K, int high)
{
  if(low>high) return false;
  int mid=(low+high)/2;
  if(ar[mid]==K)
    return true;
  else if(ar[mid]>K)
    return searchBinary(ar, N, low, mid-1)
  else if(ar[mid]<K)
    return searchBinary(ar, N, mid+1, high);
}
```

→ $T(N) = T(N/2) + T(N/2) + 1$

$= \log_2 N$ → Binary Search

(*) Quick Sort

Todo