Question 1:

A)
```
for(int i=0; i<n; ++i)              n+1
{   for(int j=0; j<n; ++j)          n+1
    { sum[i] += entry[i][j];        nxn
    }
}
```

Time Complexity:

↳ **Dry Run.**

Passes

i=0:

j=0 = sum[0] += entry[0][0];

j=1 = sum[0] += entry[0][1];

j=2 = sum[0] += entry[0][1];

⋮

j=n-1 = sum[0] += entry[0][j-1];

i=1:

j=0 = sum[1] += entry[1][0];

⋮

j=n-1 = sum[1] += entry[1][j-1]

⋮

i=n-2:

j=0 = sum[i] += entry[i][0];

⋮

j=n-1 = sum[n-2] += entry[n-2][n-1];

i=3;

j=0: Sum[3] += entry[3][0];

j=1: Sum[3] += entry[3][1];

⋮

j=n-1: Sum[3] += entry[3][n-1];

i=4:

j=0: Sum[4] += entry[4][0]

⋮

j=n-1: Sum[4] += entry[4][n-1].

Time Complexity:

$[(n+1)(n+1)] + n^2$

$[n^2+n+n+1] + n^2 \Rightarrow 2n^2+2n+1 = f(n)$

$O(f(n)) = \underline{O(n^2)}$ Ans worst case time.

B)
```
for(int i=0; i<n; i++)        n+1
{ for(int j=0; j<i; ++j)
    { m+=j; }
}
```

Passes

- i=0 = j=0 -
- i=1 : j=0 = m+=0
- i=1 : j=1 ——
- i=2 : j=0 = m+=0;
  - j=1 = m+=1
- i=3 : j=0 = m+=0
  - j=1 = m+=1
  - j=2 = m+=2

i=4 : j=0 = m+=0
- j=1 = m+=1
- j=2 = m+=2
- j=3 = m+=3

Time Complexity: $n(n+1)/2$

$f(n) = \cancel{[(n+1)(n-1)] + n^2}$

$O(f(n)) = O(n^2)$ Ans:

$O\left(\frac{n^2+n}{2}\right) = \underline{O(n^2)}$ Ans

C)

i=1

while(i<n){ tot+=i;
 i=i*2; }

Passes:

- i=1 = tot+=1
  1=1*2;
- i=2 = tot+=2
  = 2=2*2
- i=4 = tot+=4
  4=4*2
- i=8 = tot+=8
  8=8*2
- i=16= tot+=16
  16=16*2

Time Complexity

$O(f(n)) = O(\log n)$

as in the each pass i is multiplied by 2 so the complexity will be in log Bases.

D) i=n

while (i>0){ tot+=i
 i=i/2. }

- i= 32 tot+=32
- i=16 tot+=16
- i= 8 tot+=8
- i=4 tot+=4
- i=2 tot+=2

Time Complexity

$O(f(n)) = O(\log n)$

Because in each pass value of i is being halfed by 2. So it will be represented by log bases.

F) for(int i=0; i<n; ++i) $n+1$
{ for(j=0; j<n; ++j) $n+1$

E) for(int i=0; i<n; ++i) $n+1$
{ for(int j=0; j<n; ++j) $n+1$
{ Sum[i]+= entry[i][j][0]} $n*n$
}

Process

1. i=0 = j=0 = Sum[0]+= entry[0][0][0];
:
j=n-1 = Sum[0]+= entry[0][n-1][0];
2. i=1 : j=0 = Sum[1]+= entry[1][0][0];
:
j=n-1 = Sum[1]+= entry[1][n-1][0];
3. i=2 j=0 = Sum[2]+= entry[2][0][0];
:
j=n-1 = Sum[2]+= entry[2][n-1][0];
4. i=3 j=0 = Sum[3]+= entry[3][n-1][0];
:
j=n-1 = Sum[3]+= entry[3][n-1][0];
5. i=4 j=0 = Sum[4]+= entry[4][0][0];
:
j=n-1 = Sum[4]+= entry[4][n-1][0];

Time Complexity:-

$\because f(n) = 2n^2+2n+1$

$O(f(n)) = O(n^2)$

7) for(int i=0; i<n; ++i) $n+1$
{ for(j=0; j<n; ++j) $n+1$
{for(k=0; k<n; ++k) $n+1$
{sum[i][j]+= entry[i][j][k]} $n*n*n$
}
}

<u>Passesr.</u>

1) <u>i=0</u>: j=0
k=0 : sum[0][0]+= entry[0][0][0]
⋮
k=n-1: sum[0][0]+=entry[0][0][n-1].
⋮
j=n-1
k=0 : sum[0][n-1]+=entry[0][n-1][0]
⋮
k=n-1 : sum[0][n-1]+=entry[0][n-1][n-1]

2) <u>i=1</u> : j=0=
k=0= sum[1][0]+=entry[1][0][0];
⋮
k=n-1= sum[1][0]+=entry[1][0][n-1];

j=n-1 =
k=0 = sum[1][n-1]+=entry[1][n-1][0]
⋮
k=n-1= sum[1][n-1]+=entry[1][n-1][n-1].

F) Time Complexity:

• $i=2$

$j=0$: $k=0 = Sum[2][0] += entry[2][0][0]$.

⋮

$k=n-1 = Sum[2][0] += entry[2][0][n-1]$.

⋮

$j=n-1$ $k=0 = Sum[2][n-1] += entry[2][n-1][0]$

⋮

$k=n-1 = Sum[2][n-1] += entry[2][n-1][n-1]$

• $i=3$

$j=0$: $k=0 = Sum[3][0] += entry[3][0][0]$

⋮

$k=n-1 = Sum[3][0] += entry[3][0][n-1]$

⋮

$j=n-1$: $k=0 = Sum[3][n-1] += entry[3][n-1][0]$

⋮

$k=n-1 = Sum[3][n-1] += entry[3][n-1][n-1]$

• $i=4$.

$j=0$: $k=0 = Sum[4][0] += entry[4][0][0]$

⋮

$k=n-1 = Sum[4][n-1] += entry[4][0][n-1]$

⋮

$j=n-1$ = $k=0 = Sum[4][n-1][0] += entry[4][n-1][0]$

⋮

$k=n-1 = Sum[4][n-1] += entry[4][n-1][n-1]$

F) time Complexity:

$O(F(n)) = O(n^3)$.

Where as $F(n) = \{(n+1)(n+1)(n+1)\} + n^3$

So $O(F(n)) = O(n^3)$. a.

G) for(int i=0; i<n; ++i) n+1
{ for (k=0; k<n; ++k) n+1
{ sum[i] += entry[i][0][k]; n*n
}
}

⇒ i=0   k=0 = sum[0] += entry[0][0][0];

:

k=n-1 = sum[0] += entry[0][0][n-1];

i=1   k=0 = sum[1] += entry[1][0][0]

:

k=n-1 = sum[1] += entry[1][0][n-1]

i=2   k=0 = sum[2] += entry[2][0][0]

:

k=n-1 = sum[2] += entry[2][0][n-1]

i=3   k=0 = sum[3] += entry[3][0][0]

:

k=n+1 = sum[3] += entry[3][0][n-1]

i=4   k=0 = sum[4] += entry[4][0][0]

:

k=n-1 = sum[4] += entry[4][0][n-1]

Time Complexity

$O(f(n)) = O(n^2)$.

4) for(i=0; i<n ; ++i)   (n+1)
{ for (j=0; j<sqrt(n) ; ++j)   sqrt(n)+1
{ m+=j; }   n * sqrt(n).
}

i=0 : j=0 = m+=0;
:
j = sqrt(n)-1 = m+= (sqrt(n) -1);

i=1 : j=0 = m+=0
:
j = sqrt(n)-1 = m+= (sqrt(n)-1)

i=2 : j=0 = m+=0
:
j = sqrt(n)-1 = m+= sqrt(n)-1

i=3 : j=0 = m+=0
:
j = sqrt(n)-1 = m+= sqrt(n)-1

i=4 : j=0 = m+=0
:
j = sqrt(n) - 1 = m+= sqrt(n) -1.

Time Complexity:

$O(f(n)) = O(n * Sqrt(n))$ as the outer loop will be going on for n times but the inner loop will be going on for Sqrt(n) thus $O(f(n)) = O(n * Sqrt(n))$ Ans

(I) for(i=0; i<n; ++i) (n+1)

{ ~~for(j~~

m+=j

m+=j 31 * n

m+=j

:

31 times }

Time Complexity:

$O(f(n)) = O(n)$

as the outer loop will be going for n times only and the instruction will be executing with C; constant = 31 Ans

i=0 = (m+=0) * 31

i=1 = (m+=1) * 31

i=2 = (m+=2) * 31

i=3 = (m+=3) * 31

i=4 = (m+=4) * 31

(J) int total(n)
for(int i=0; i<n; ++i)
{ subtotal+=i;}

main(){ for (int i=0; i<n; ++i)
{ tot+=total(i);}

Passes

i=0 => total(0) ↳ i=0 = subtotal+=0;
tot+=

i= ~~10~~ = ~~subtotal~~;

i=1= total(1)= i=0 = subtotal+=0;
tot+=
i=1 - ———

=2 tot+=total(2)= i=0= subtotal+=0
i=1 = subtotal+=1

=3 tot+=total(3)= i=0 = subtotal+=0.
i=1= subtotal+=1
i=2= subtotal+=2

=4 tot+=total(4)=

i=0 = subtotal += 0

i=1= subtotal+=1

i=2= subtotal+=2

i=3= subtotal+=3

Time Complexity

$O(f(n)) = O(n^2)$

the function will be going on for (n) times so does the loop in main thus $O(f(n)) = O(n^2)$

Ola

(K) for(i=0; i<n; ++i) n+1
{ subtotal=0;
for (j=0; j<i; ++j) n+1
{ subtotal+=j; }
total+= subtotal; n
}

Passes

$i=0$; $j=0$ —

$i=1$; $j=0$ subtotal+=j. - total= subtotal

$i=2$, $j=0$ subtotal+= j. - ~~tot += subtotal~~

$j=1$ subtotal+=j.

tot= subtotal

$i=3$ $j=0$ = subtotal +=j.

$j=1$ = subtotal+=j.

$j=2$: subtotal+=j

tot+=subtotal.

$i=4$ $j=0$ subtotal+=j

$j=1$ subtotal+=j

$j=2$ subtotal+=j

$j=3$ subtotal+=j.

tot+=subtotal.

Time complexity:

$O(f(n)) = O(n^2)$
as the outer function will be going on for like n times while the inner loop will be going on for i time = n so
$O(f(n)) = O(n^2)$.

(Q no 2)

A)

```
int Sum=0
for (i=1; i<=n; ++i)        n+1
 { for (j=1; j<=i; ++j)     n+1
   { Sum+=1;                n*n
   }
 }
```

Time Complexity:

The outer will be iterating for n times meanwhile inner loop will be going on n*n times so.

$f(n) = [(n+1)(n+1)] + n^2$

$n^2+n+n+1+n^2$

$f(n) = \boxed{2n^2+2n+1}$

$O(f(n)) = O(n^2)$.

Thus time Complexity for the given code is $O(n^2)$ Ans

B) $O(n^2)$

```
int sum = 0;
for(i = 0, i<n; ++i)            -(n+1)
{ for(j=0; j<=2*i; ++j){        2*(n+1) 2*i = 2*(n+1)
      sum += 1;                 n*n
  }
}
```

Time Complexity

$f(n) = \{(n+1)(2^{*}i+1)\} + n^2$

$\Rightarrow 2ni + n + 2i + 1 + n^2 \quad \because i = n$

$= 2n^2 + 2n + 1$

$O(f(n)) = O(n^2)$

Thus The time complexity is $O(n^2)$ Ans

C)

```
sum=0;
for(i=1; i<=n; i*=2)     logn
{ for(j=0; j<i; ++j)     n+1
  { sum+=1}              n*logn
}
```

Time Complexity

Outer loop will be iterating for $\log_2(n)$ as $i^{*}=2$ becomes after each iteration. Mean while inner loop will be going on for $n^{*}\log n$ thus Time Complexity will be

$O(f(n)) = O(n \cdot \log n)$ Ans

(Q3)

A) Sum=0 i=1

```
while(i<=n){            <- logn
   for( j=0 ; j<i; ++j) n+1
   { Sum+=1;  n* logn
   }
i*=2}
```

Time Complexity

$f(n) = (\log n * n+1) + n \log n$

$O(f(n)) = n \log n.$

Outerloop will be iterating for $\log_2 n$ times as i becomes i*=2 after each iteration. Whereas, innerloop will be going on for n*logn times as of outerloop's iteration along with innerloop.

$\boxed{O(f(n)) = n \log n}$ Ans

B)

```
Sum=0
for(i=0; i<n; ++i) n+1
{ for(j=0; j<n; ++j) (n+1)
   { for(k=j; k<n ; k+=j) {
      Sum+=1;          logn
   }                n3* logn
}  }
```

Time Complexity.

$f(n) = [(n+1)(n+1) * \cancel{(\log n)}^{(n+1)}] + \cancel{n \log n}^{n3}$

$O(f(n)) = \cancel{O(n^2 \log(n^2))}^{O(n^3)}$ Ans

As the outerloops and middle will be going for n^2 time meanwhile innermost will be going for ~~logn~~ (n+1)*n3 time thus

$\boxed{O(\cancel{n^3 \log n})}$ Ans

$O(n^3)$ Ans

Analy

(A) A=[9,1,2,3,4,5,6,7,8].

(B)

Insertion Sort:

Case A:

as insertion Sort works best on small list like in(A). and it works on comparing and swapping on with only element thus Insertion Sort is the best Choice.

Why Not Bubble and Selection?

Bubble Sort works on Comparing one element with other until all the array is Sorted thus the time Complexity of it $O(n^2)$ and the A is partially Sorted.

Where as, Selection Sort Sorts the array by taking Smaller elements on the left Side of larger element. In the given Case all the elements will be Swapped by $O(n)$ times thus its only taking time extra.

Case B:

Insertion Sort:

Insertion Sort is Suitable for Case 'B'. Since list is kind of Short from "20-30" based on Last Playback time. Insertion Sort will sort it by each song Playback time and swap it to the appropriate Position.

(A) name: Haider

a) Why not Selection and Bubble Sort:

Selection Sort and Bubble Sort both are not suitable for this

Scenario. In Bubble Sort worst case will be it will compare the values unnecessary which will be just waste of time. Time Complexity of Bubble Sort is $O(n^2)$

Where as, Selection Sort Compares the values and places it on either left or right. In this Case it will be just waste of time.

c) Time Complexities

1) Insertion Sort:

Best case: $O(n)$

Average case: $O(n^2)$

Worst case: $O(n^2)$

2) Bubble Sort:

Best case: $O(n)$

Average Case: $O(n^2)$

Worst Case: $O(n^2)$

3) Selection Sort:

Best Case: $O(n^2)$

Worst Case: $O(n^2)$

Average Case: $O(n^2)$

(Qno5)

(A) name: Haider

A =

H	A	I	D	E	R

b) Apply Bubble Sort:

H	A	I	D	E	R

A	H	I	D	E	R

1st pass.

A	H	I	D	E	R

2nd pass

A	H	D	I	E	R

3rd Pass

A	H	D	E	I	R

4th pass

↳ No Pass

A	H	D	E	I	R

No Pass

A	H	D	E	I	R

5th pass

A	D	H	E	I	R

6th pass

A	D	E	H	I	R

No Pass

Ans 5

B)

1) Inplace Sorting.

because

mory.

② Apply Selection Sort:

A =

H	A	I	D	E	R

A	H	I	D	E	R

A	D	I	H	E	R

A	D	E	H	I	R

← Sorted

③ Apply Insertion Sort:

A =

H	A	I	D	E	R

1st pass

A	H	I	D	E	R

2nd pass

A	I	H	D	E	R

3rd pass

A	D	I	H	E	R

4rd. pass

A	D	E	I	H	R

No pass.

Ans 5
B)

1) **Inplace Sorting:-**

- **Bubble Sort** holds the property of In-place, because it only requires a constant amount of additional memory.

- **Insertion Sort:-**

  Yes, Insertion Sort holds property of Inplace as it only requires and operates only on input array directly, and it does not need any extra array.

- **Selection Sort:-**

  Yes, Selection Sort holds the property of in-place sorting as it works on the constant or given array to find minimum and max and does the swapping.

2) **Online Sorting:-**

- **Bubble Sort:-**

  No, Bubble Sort is not suitable for online sorting as it only performs sorting on an constant array and it can't handle the elements as they are coming.

- **Insertion Sort:-**

  No, Insertion Sort is not suitable as ~~they~~ it need to have a given array to get sorting.

- **Selection Sort:-**

  No Selection Sort is also not suitable as it also need to have a constant array like scenario for Bubble and insertion.

Stable Sorting

Bubble Sort

Yes, Bubble Sort is Stable. It prevents the same element to be exchanged like if two number said to be equal then it will not get swapped.

Insertion Sort

Yes, its also Stable like Bubble Sort - If two numbers said to be equal then it will not get swapped to prevent the position.

Selection Sort

No its not Stable as it may change the position of elements during the process. - If two numbers are equal then it may get swapped.