1. The task is to maintain a database of bits, to be precise, an array of bits X[1 ... n], that supports the following operations.

// X is global.

( i ) **Init(n): Initialization of X to an n-bit array. All bits of X[1...n] are initialized to 0.**

**Init(n):**

for i in 1…n:

X[ i ] = 0

*Time Complexity - Time to traverse all n element and assign a value - O( N ) where n is size of array provided.*

( ii ) **Set(i) where i <= n-1: Set X[i]=1. It is guaranteed that i <= n-1, so the rightmost bit of X[] is never set.**

**Set( i ):**

X[ i ] = 1

*Time Complexity - Time complexity to assign value to a particular index of array X- O( 1 )*

( iii ) **IsSet(i): return X[i]**

**IsSet(i)**:

return X[i]

*Time Complexity - Time complexity to access value to particular index of array X- O( 1 )*

( iv ) **NextUnset(i): return the smallest j >= i s.t. X[j]=0. This will always return some index.**

**NextUnset(i):**

for j in i…n-1:

if X[ j ] == 0:

return j

return n

*Time Complexity - Time to traverse all n elements in the worst case - O( N ), where N is the size of the array provided.*

Note- rank of bit is index of the bit in the bit sequence.

**B ) Implementation of the above problem using dis-joint set.**

Approach: there are n dis-joint sets containing one element each . Each element is represented by a number defining the bit's rank in the bit sequence and contains a field named value representing the bit's value. Leader of each set is itself.

// FIND SET WILL RETURN NULL IF THE SET DOES NOT EXIST

( i ) Init(n)- Initialization of n-bits of the bit sequence. The default value assigned to a newly created bit is 0.

**Init(n):**

for i in 1…n:

create\_obj= obj( )

create\_obj.value = 0

create\_obj.index = i

makeset(create\_obj )

( ii ) Set(i)- setting value 1 to an nth bit of the bit sequence.

**Set( i ):**

create\_obj=obj()

create\_obj.value=0

create\_obj.index=i

found\_obj\_0 = find(create\_obj)

found\_obj\_0

if ( found\_obj\_0 == NULL):

return

else:

found\_obj\_0.value=1

return

( iii ) IsSet(i): return the bits value of ith bit.

**IsSet(i):**

create\_obj\_0=obj( )

create\_obj\_0.value=0

create\_obj\_0.index=i

create\_obj\_1=obj( )

create\_obj\_1.value=1

create\_obj\_1.index=i

found\_obj\_0 = find(create\_obj\_0)

found\_obj\_1 = find(create\_obj\_1)

if (found\_obj\_0 != NULL):

return found\_obj\_0.value

else:

return found\_obj\_1.value

( iv ) NextUnset(i): return the smallest j >= i s.t. X[j]=0.

**NextUnset(i):**

for i in i…n:

create\_obj\_0=obj( )

create\_obj\_0.value=0

create\_obj\_0.index=i

found\_obj\_0 = find(create\_obj\_0)

if (found\_obj\_0 != NULL):

return found\_obj\_0.index

C )

Init(9) – As shown above, it will create a 9-bit sequence, values initialized with zero

IsSet(3) – return the value of bit rank 3 in bit sequence [ return 0 ].

Set(4) – set the object field of bit rank 4 in bit sequence to 1.

NextUnset(4) – return the bit smallest rank between 4 to 9, which is 0 [ return 5 ].

Set(5) – set the object field of bit rank 5 in bit sequence to 1.

NextUnset(4) – return the bit smallest rank between 4 to 9, which is 0 [ return 6 ].

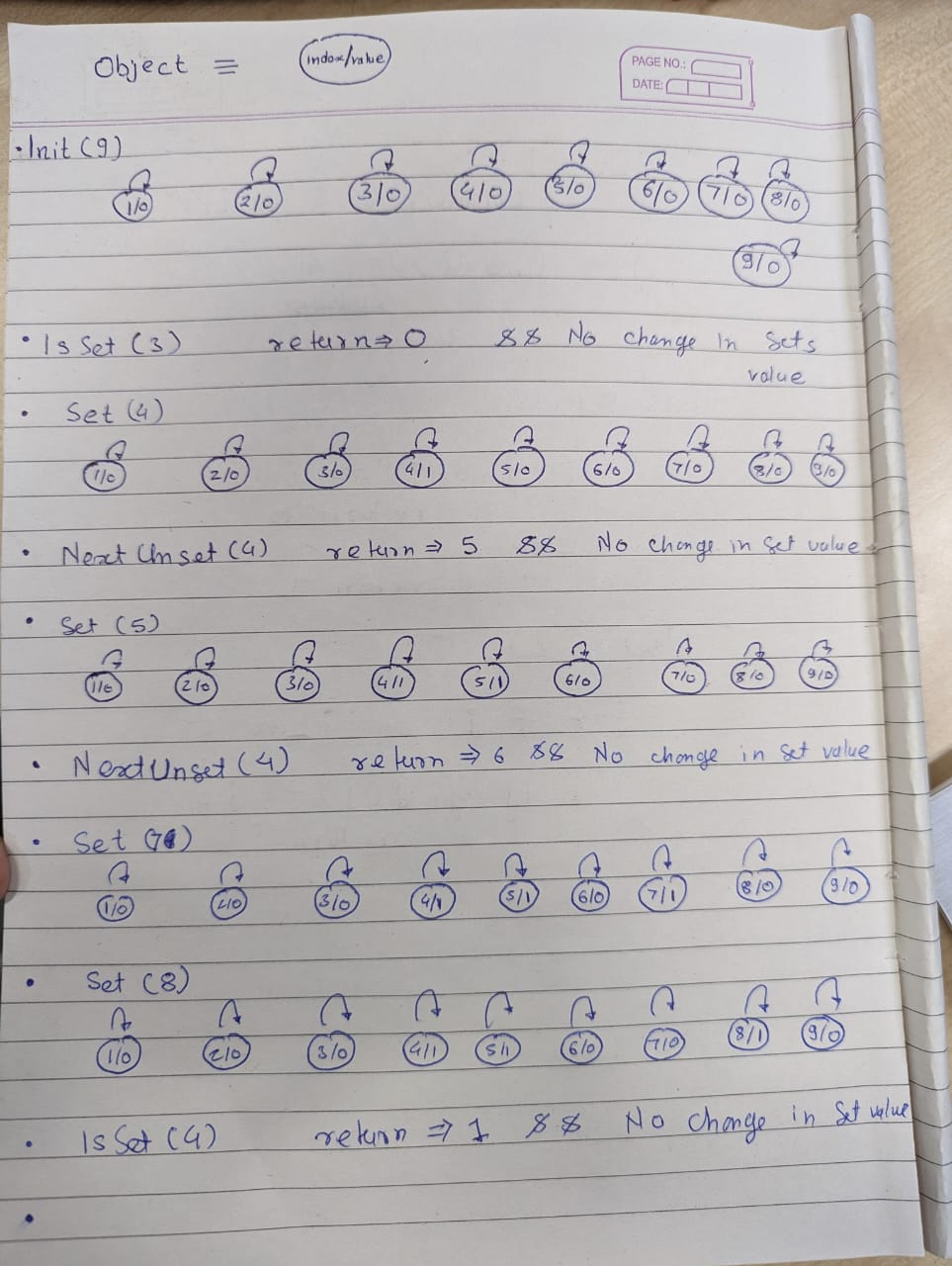
Set(7) – set the object field of bit rank 7 in bit sequence to 1.

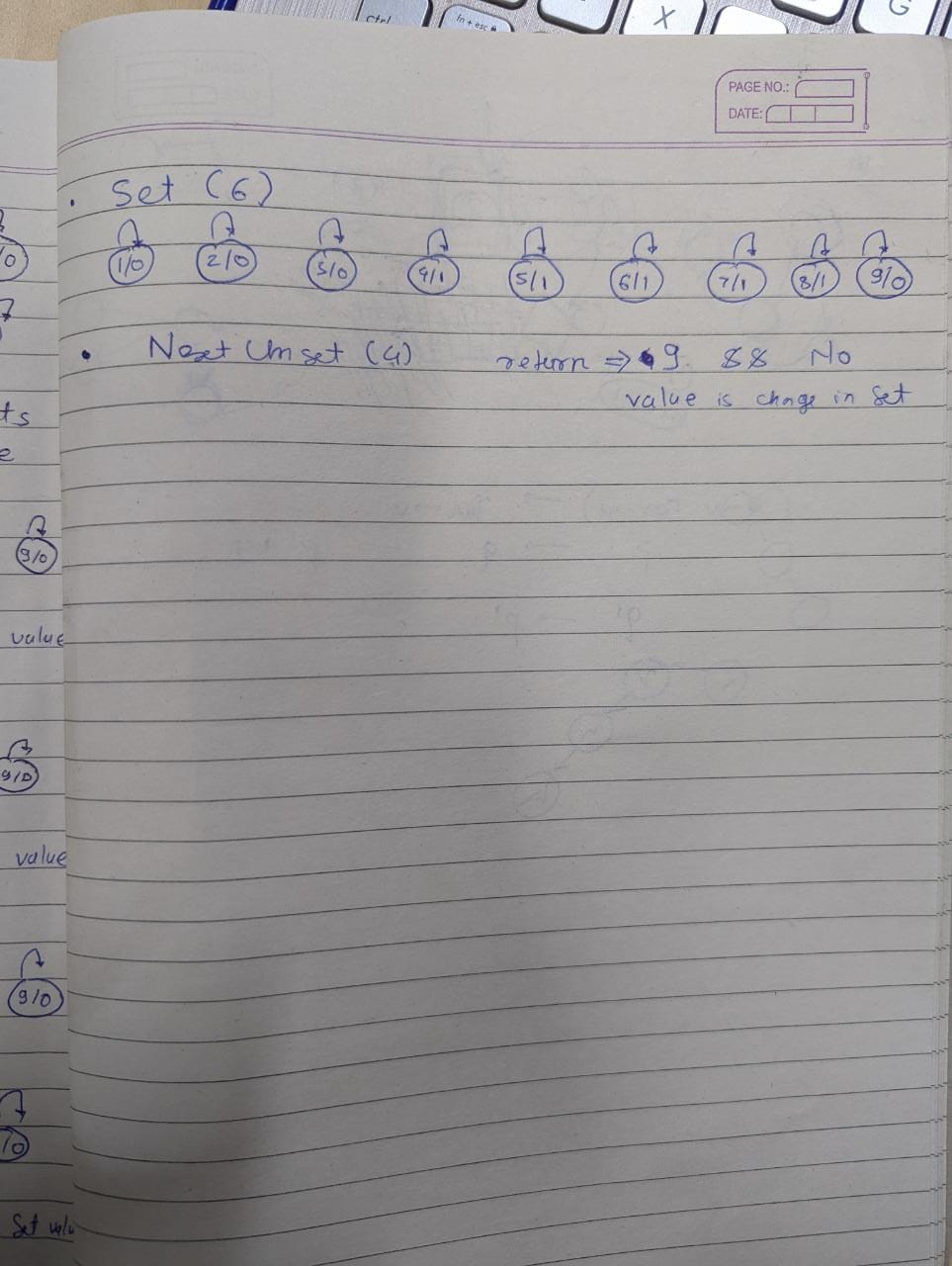
Set(8) – set the object field of bit rank 8 in bit sequence to 1.

IsSet(4) – return the bit smallest rank between 4 to 9, which is 0 [ return 6 ].

Set(6) – set the object field of bit rank 6 in bit sequence to 1.

NextUnset(4) – return the bit smallest rank between 4 to 9, which is 0 [ return 9 ].





D)

|  | Init(n) | IsSet( i ) | Set( i ) | NextUnset( i ) |
| --- | --- | --- | --- | --- |
| Direct (as in (a)) | O( n ) | O( 1 ) | O( 1 ) | O( n ) |
| Reversed tree | O( n ) | O( n ) | O( n ) | O( n2 ) |
| Reversed tree + Union by depth | O( n ) | O( logn ) | O( logn ) | O( n\*logn ) |
| Shallow tree + threading | O( n ) | O( 1 ) | O( 1 ) | O( n ) |
| Shallow + threading + union by size | O( n ) | O( 1 ) | O( 1 ) | O( n ) |