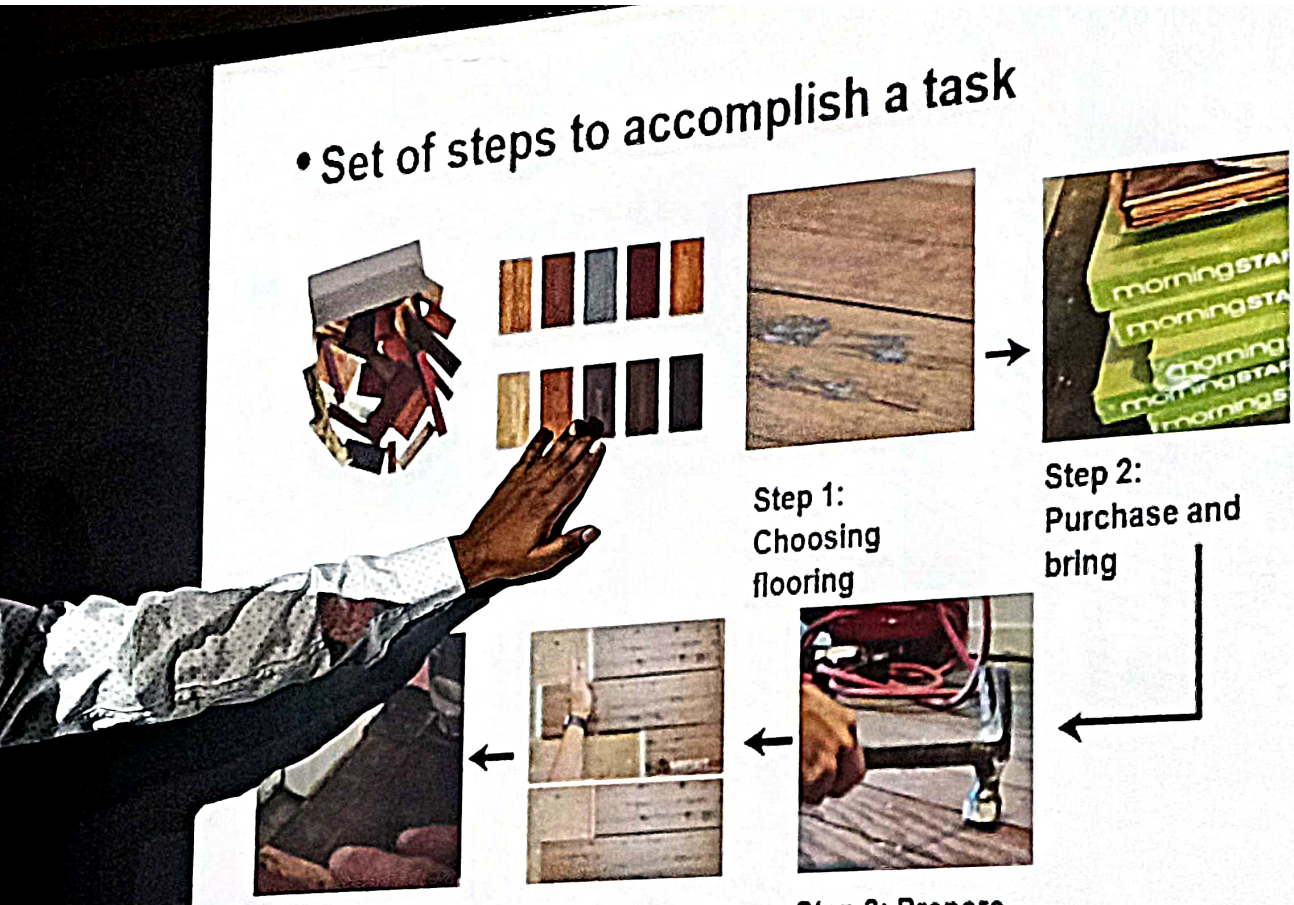
- Set of steps to accomplish a task



Step 1: Choosing flooring

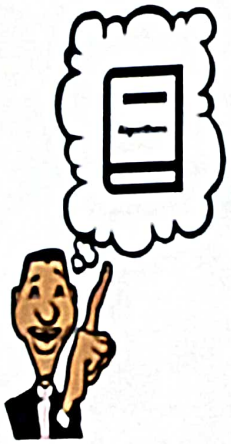
Step 2: Purchase and bring

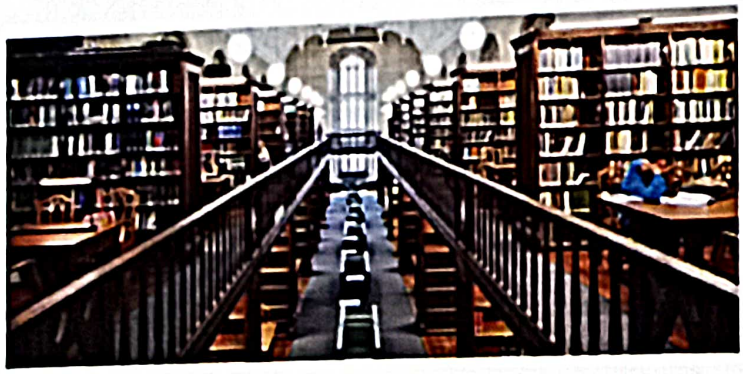
Step 3: Prepare sub flooring

Step 4: Determine the layout

Step 5: Trim door casing

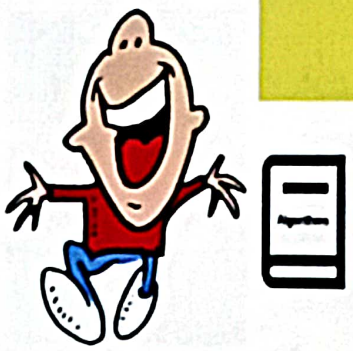
Why are Data Structures and Algorithms important?



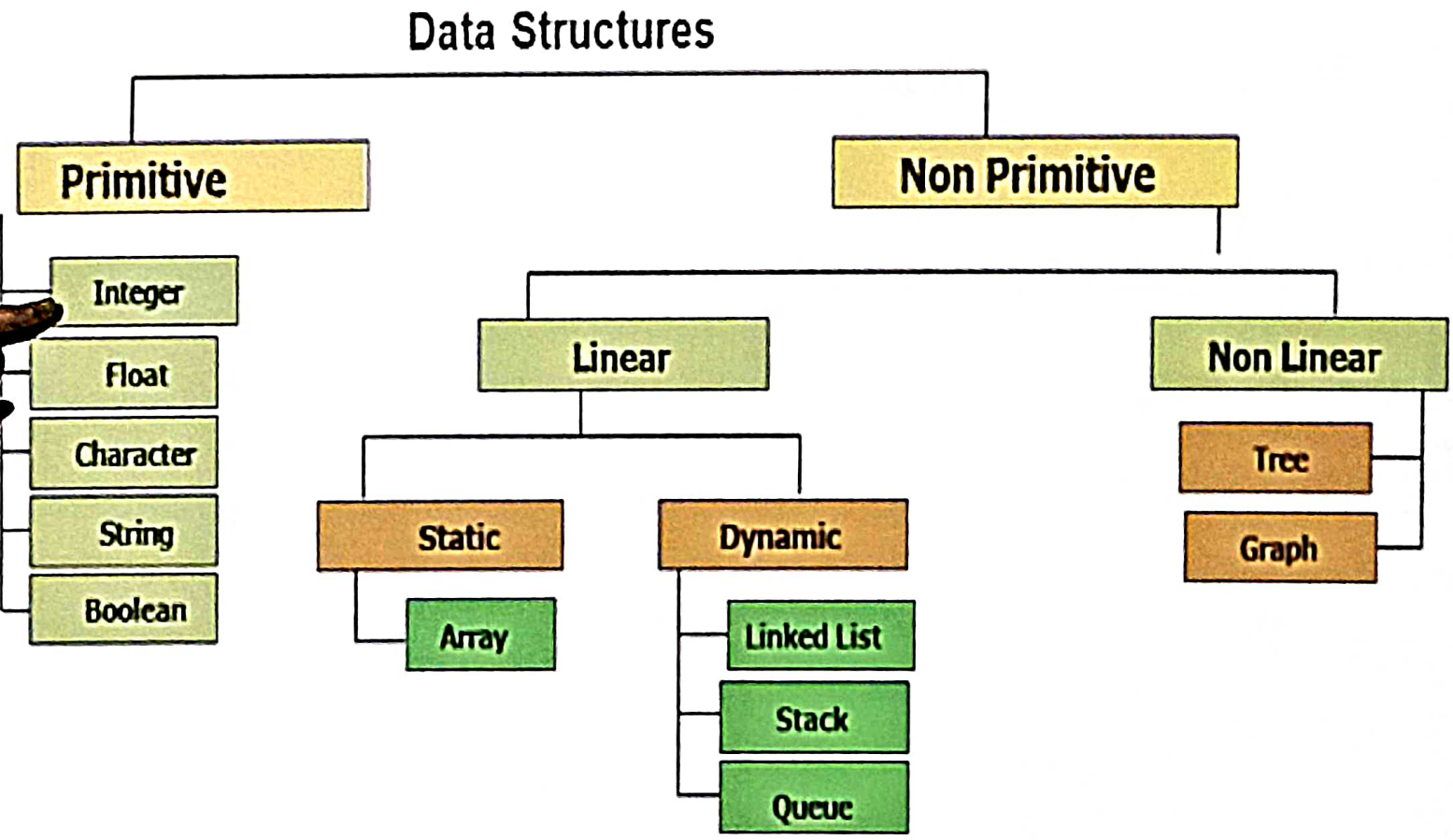


COMPUTER SCIENCE

ALGORITHMS



Types of Data Structures



Primitive Data Structures

DATA STRUCTURE	Description	Example
INTEGER	Numbers without decimal point	1, 2, 3, 4, 5,1000
FLOAT	Numbers with decimal point	3.5, 6.7, 6.987, 20.2
CHARACTER	Single Character	A, B, C, F
STRING	Text	Hello, Data Structure
BOOLEAN	Logical values true or false	TRUE, FALSE

Big O Notations

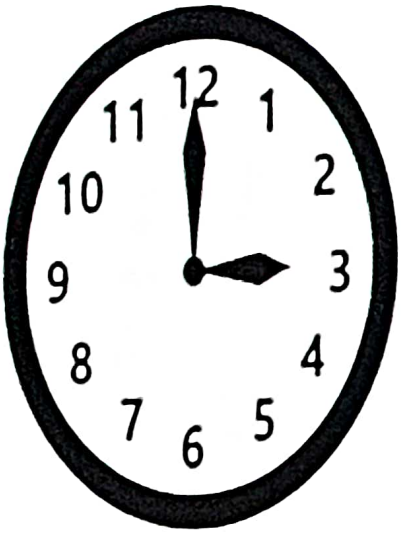
Size : 1TB

Big O is the language and metric we use to describe the efficiency of algorithms

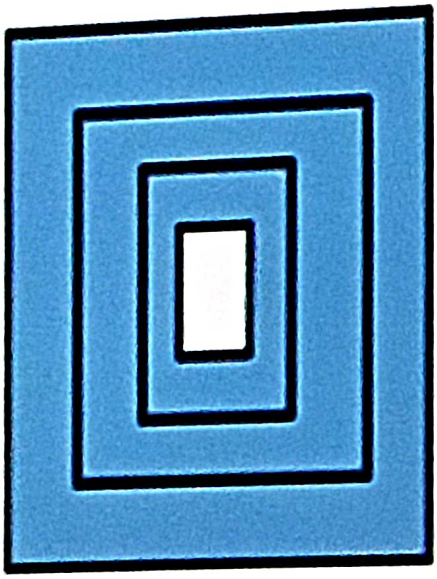
Big O Notations

$O(N)$, $O(N^2)$, $O(2^N)$

Space complexity : O(wh)

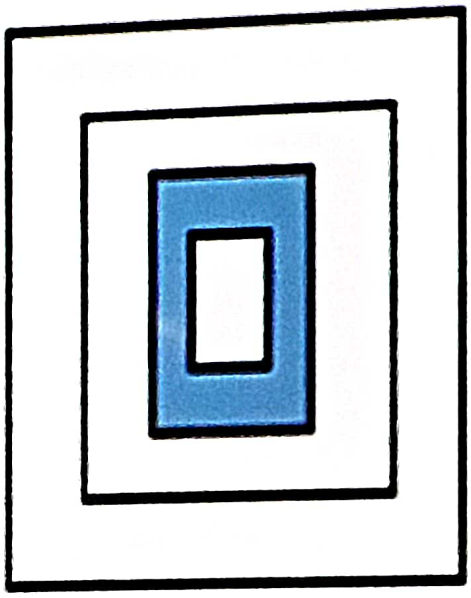


Code 1: 30 Sec



More memory

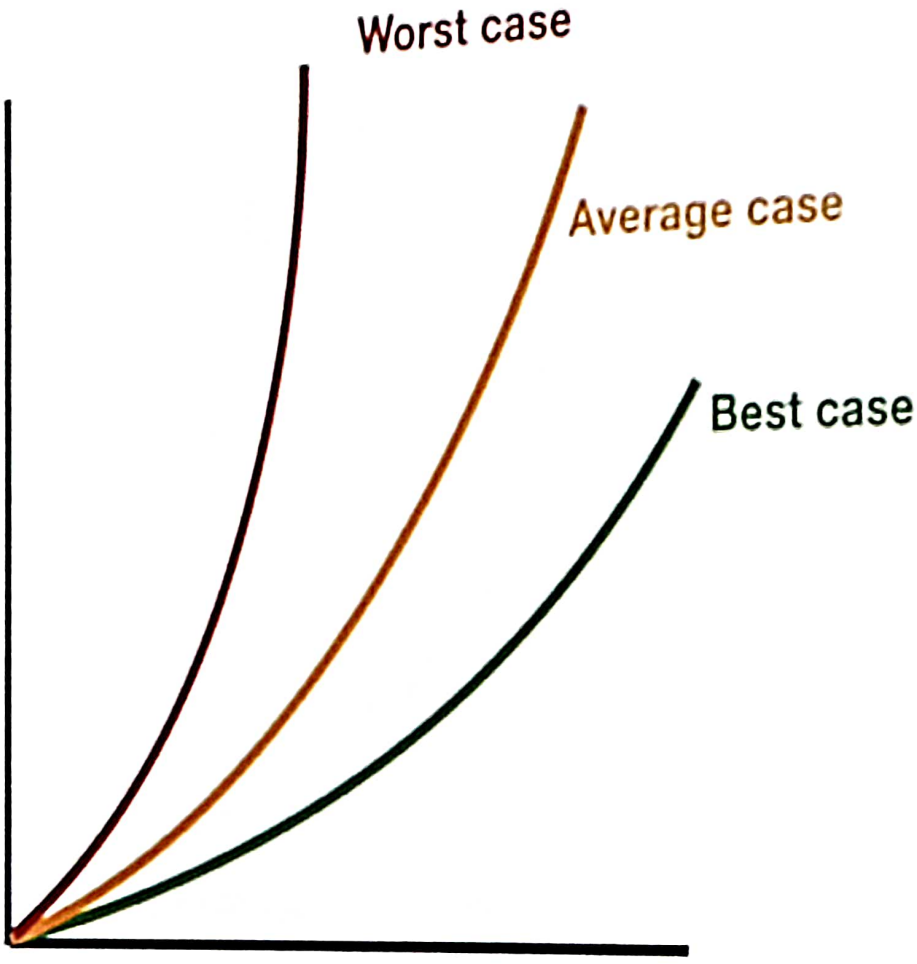
Code 2: 60 Sec



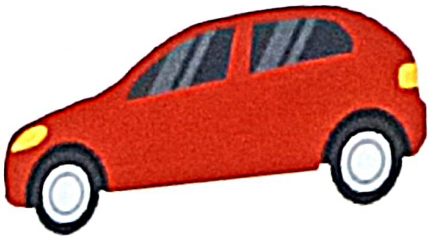
Less memory

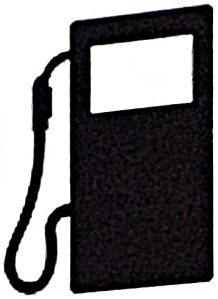
Big O notations

- Best case
- Average case
- Worst case



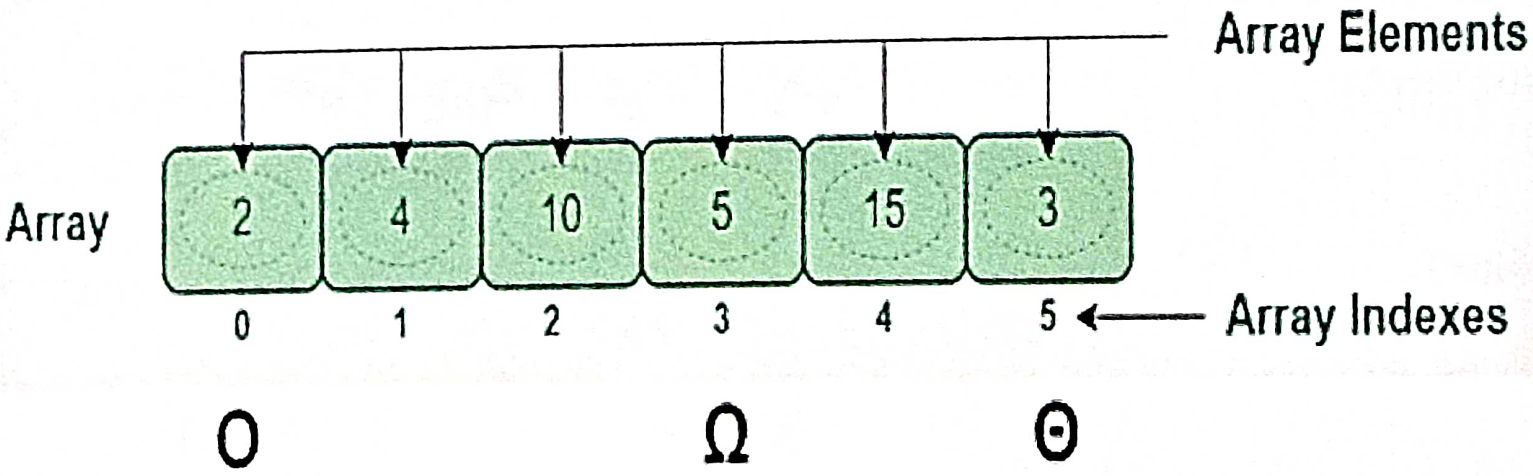
Big O notations





- City traffic - 20 liters
- Highway - 10 liters
- Mixed condition - 15 liters

Big O(Omicron), Big Theta and Big Omega



Big O, Big Theta and Big Omega

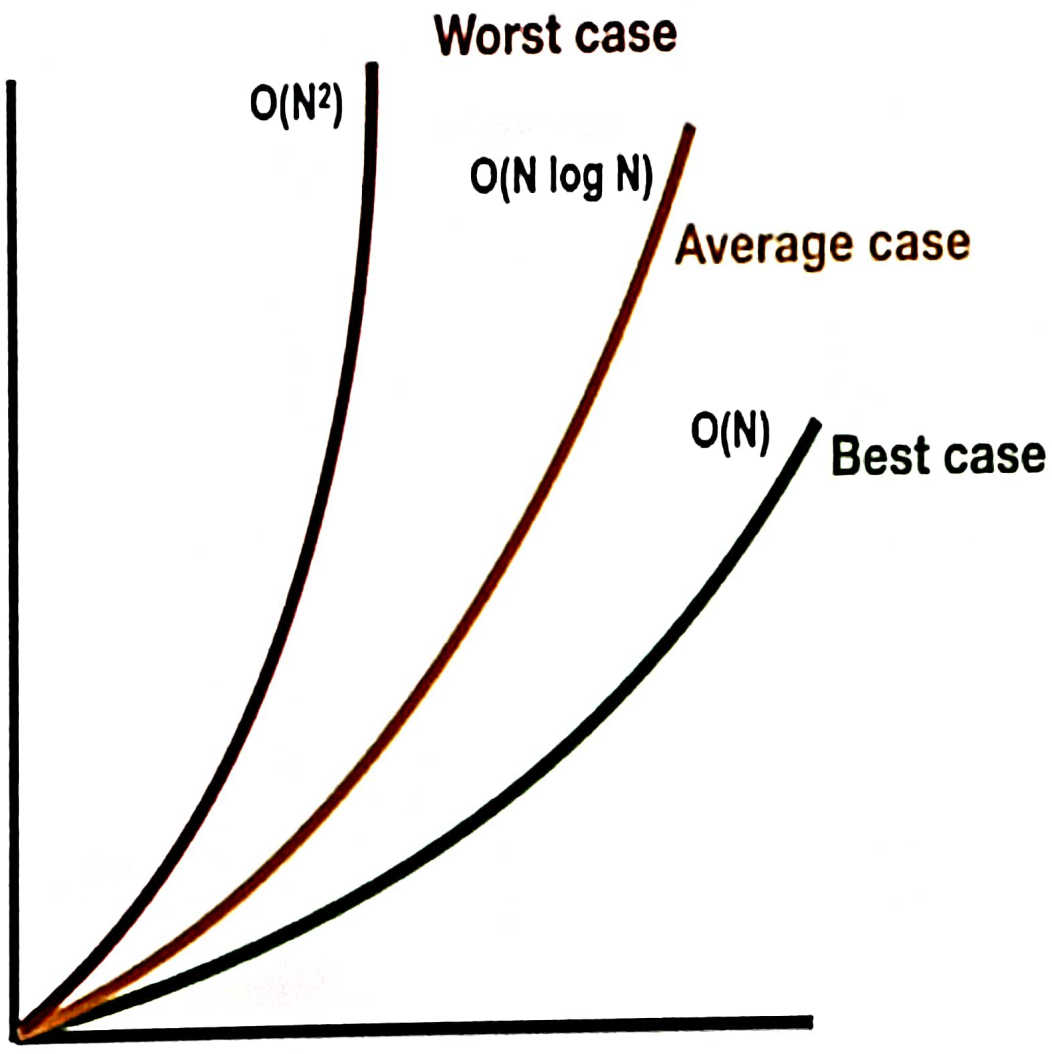
- **Big O** : It is a complexity that is going to be less or equal to the worst case.
- **Big - Ω (Big-Omega)** : It is a complexity that is going to be at least more than the best case.
- **Big Theta (Big - Θ)** : It is a complexity that is within bounds of the worst and the best cases.

How to measure the codes using Big O

5 Rules

No	Description	Complexity
Rule 1	Any assignment statements and if statements that are executed once regardless of the size of the problem	O(1)
Rule 2	A simple "for" loop from 0 to n (with no internal loops)	O(n)
Rule 3	A nested loop of the same type takes quadratic time complexity	$O(n^2)$
Rule 4	A loop, in which the controlling parameter is divided by two at each step	O(log n)
Rule 5	When dealing with multiple statements, just add them up	

Big O notations



Complexity	Name	Sample
O(1)	Constant	Accessing a specific element in array
O(N)	Linear	Loop through array elements
O(LogN)	Logarithmic	Find an element in sorted array
$O(N^2)$	Quadratic	Looking ar a every index in the array twice
$O(2^N)$	Exponential	Double recursion in Fibonacci

O(1) - Constant time

```
array = [1, 2, 3, 4, 5]

array[0] // It takes constant time to
access first element
```

Complexity	Name	Sample
O(1)	Constant	Accessing a specific element in array
O(N)	Linear	Loop through array elements
O(LogN)	Logarithmic	Find an element in sorted array
$O(N^2)$	Quadratic	Looking ar a every index in the array twice
$O(2^N)$	Exponential	Double recursion in Fibonacci

O(N) - Linear time

```
array = [1, 2, 3, 4, 5]

for element in array:
    print(element)
//linear time since it is visiting every element
of array
```

- O(LogN) -

```
array = [1, 2, 3, 4, 5]
for index in range(0,len(array),3):
    print(array[index])

//logarithmic time since it is visiting only some
elements
```

O(1)		Find an element
O(LogN)	Logarithmic	Looking ar a every index in the array twice
$O(N^2)$	Quadratic	Double recursion in Fibonacci
$O(2^N)$	Exponential	

O(LogN) - Logarithmic time Binary search

```
search 9 within [1,5,8,9,11,13,15,19,21]
compare 9 to 11 → smaller
search 9 within [1,5,8,9]
compare 9 to 8 → bigger
search 9 within [9]
compare 9 to 9 return
```

```
N = 16
N = 8 /* divide by 2 */
N = 4 /* divide by 2 */
N = 2 /* divide by 2 */
N = 1 /* divide by 2 */
```

$2^k = N \rightarrow \log_2 N = k$

O(LogN)	Logarithmic	
$O(N^2)$	Quadratic	Looking ar a every index in the array twice
$O(2^N)$	Exponential	Double recursion in Fibonacci

$O(N^2)$ - Quadratic time

```
array = [1, 2, 3, 4, 5]

    for x in array:
        for y in array:
            print(x,y)
```

O(N)		Find an el...
O(LogN)	Logarithmic	Looking ar a every index in the array twice
$O(N^2)$	Quadratic	Double recursion in Fibonacci
$O(2^N)$	Exponential	

$O(2^N)$ - Exponential time

```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

an array of size n

$$a = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}$$

O(n)

an array of size n*n

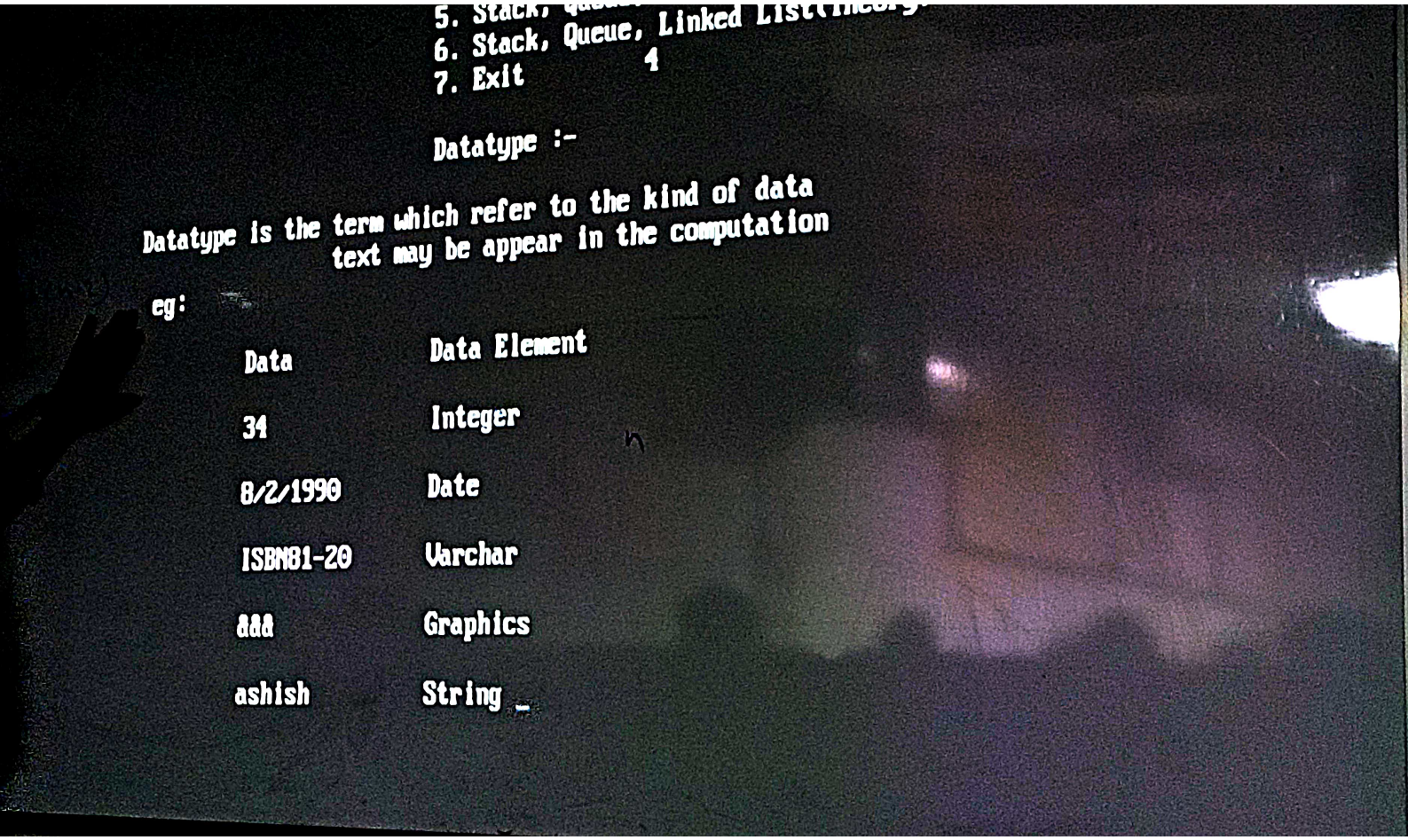
$$a = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix}$$

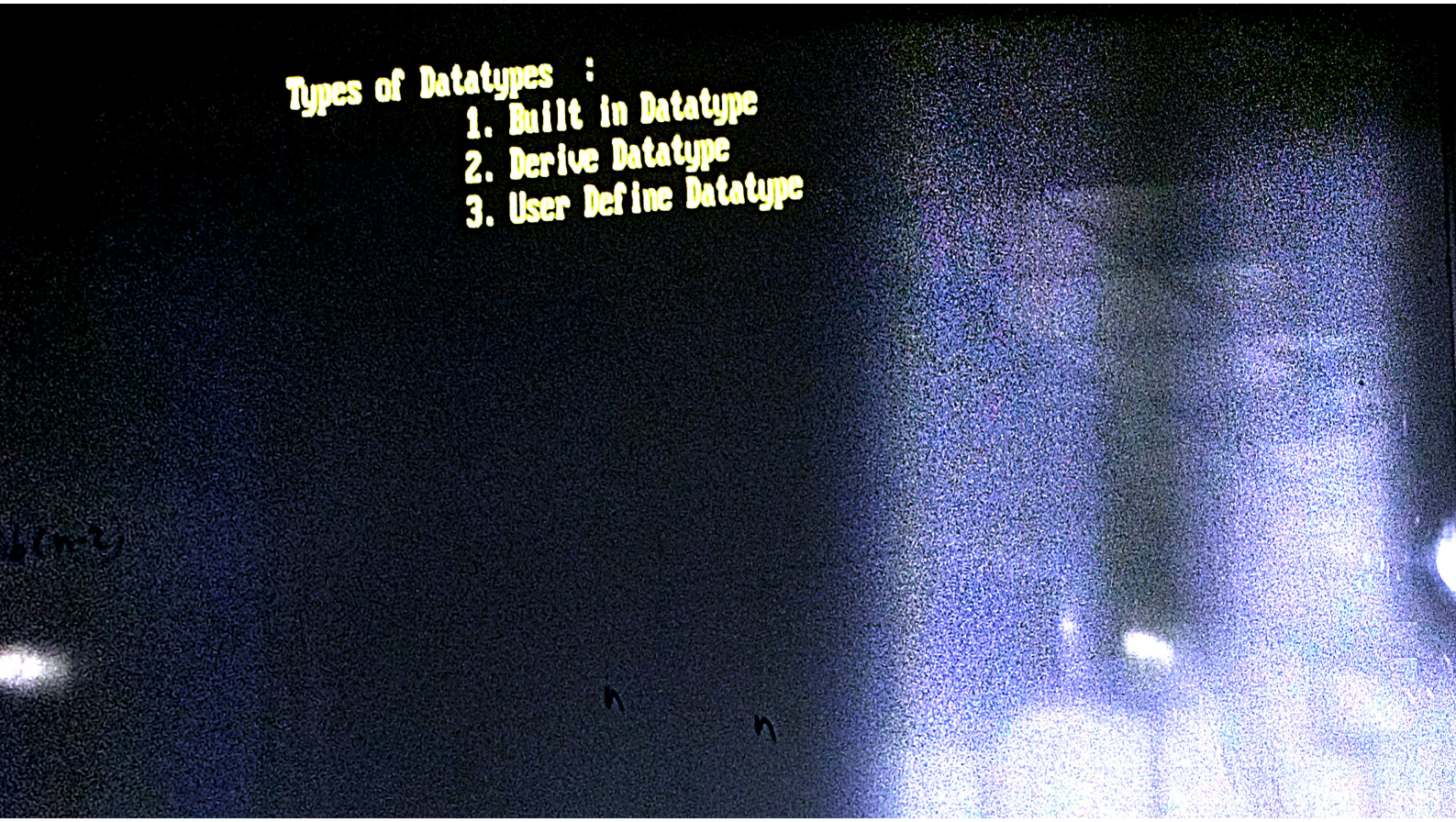
$O(n^2)$

```
def sum(n):
  if n <= 0:
    return 0
  else:
    return n + sum(n-1)
```

Space complexity : O(n)

```
1  sum(3)
2    → sum(2)
3       → sum(1)
4          → sum(0)
```


5. Stack, Queue, Linked List(Theory)
6. Stack, Queue, Linked List(Program)
7. Exit 4
Datatype :-
Datatype is the term which refer to the kind of data
text may be appear in the computation
eg:
Data Data Element
34 Integer
8/2/1990 Date
ISBN81-20 Varchar
aaa Graphics
ashish String


Types of Datatypes :
1. Built in Datatype
2. Derive Datatype
3. User Define Datatype

Types of Datatypes :

1. Built in Datatype
2. Derive Datatype
3. User Define Datatype

1. Built in Datatype

A datatype which is already define inside the library of C & C++,
called built in datatype

eg : int, char, float, double, short, long

2. Derive Datatype

A datatype which is to be derive on the basis of existing datatype,
called derive datatype

eg : struct, union, enum

3. User Define Datatype

A datatype which is define by user,
called user define datatype

eg : class _

```
Select Your Option      1. Sorting
                        2. Searching
                        3. Array
                        4. Datatype
                        5. Stack, Queue, Linked List(Graphical)
                        6. Stack, Queue, Linked List(Theory)
                        7. Exit         3

                        Array :-

An array is a finite, order & collection of homogenous data element
If Array is
        * FINITE - The array must have its size it means that array is static l
inear data structure

        * ORDERED - Array can store the data element in continues memory locati
on

        * HOMOGENOUs - Array can store the same type of data element
      Operations  :
                1. Insertion
                2. Deletion
                3. Go To Back
```

2. Deletion
3. Go To Back 1

INSERTION

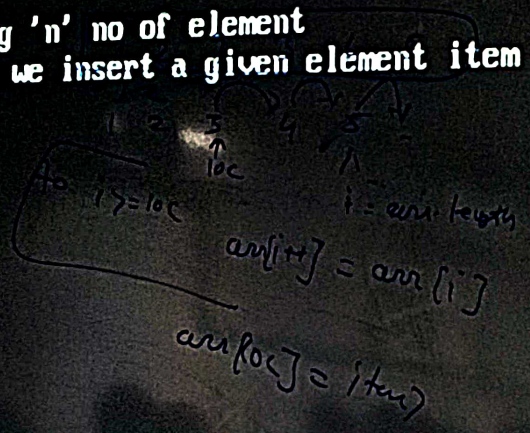
For Theroy Press 1
For Practical Press 2
For Go to Back 3 1

Algorithm

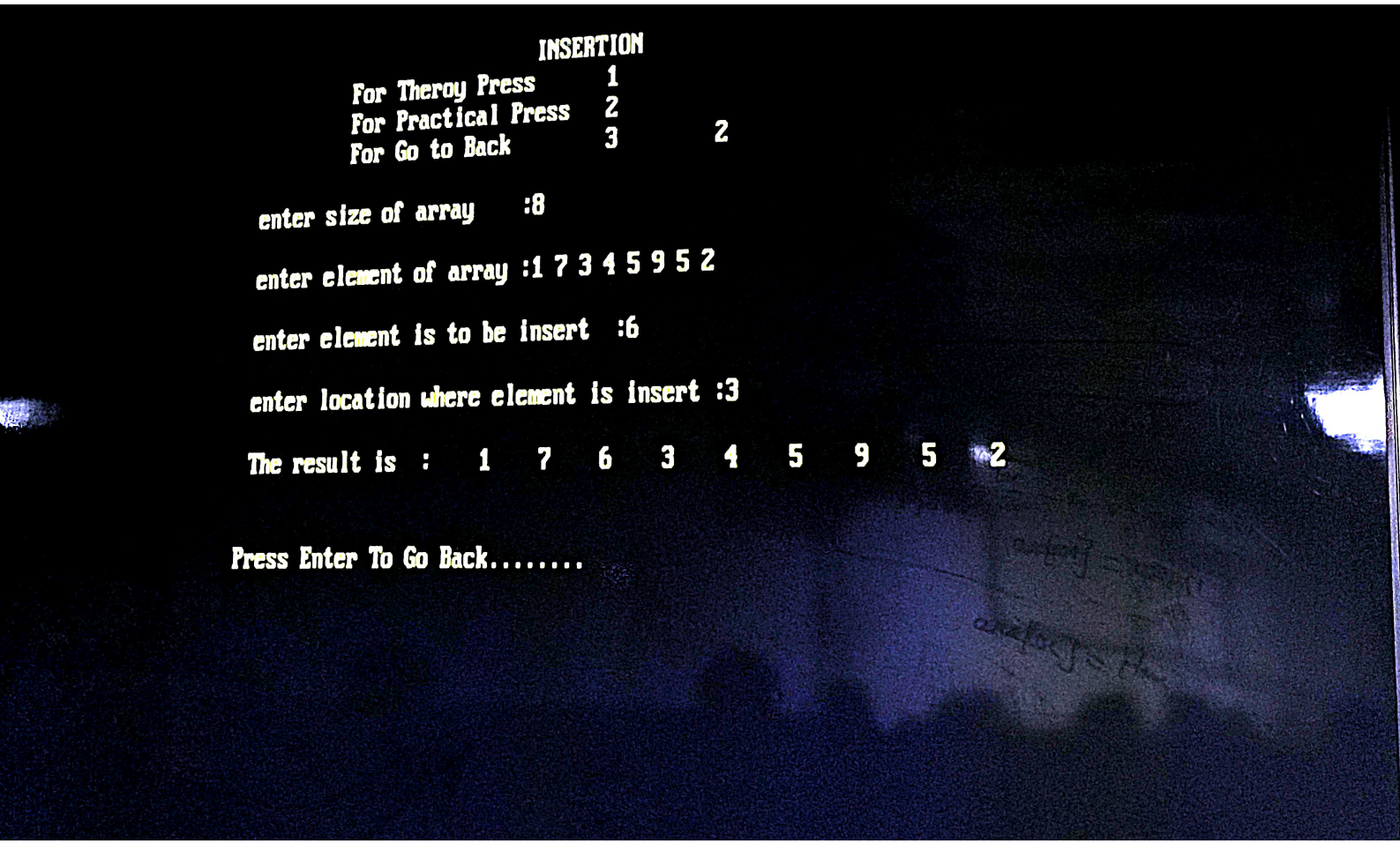
INSERT_LA(a,n,item,loc)

here a is linear array containing 'n' no of element
By using this algorithm we insert a given element item at loc

1. Set j=n
2. Repeat while(j>=loc) do
   a[j+1] = a[j]
   j = j-1
3. End while loop
4. a[loc] = item
5. Stop.



Go To Back Press Enter


INSERTION
For Theroy Press 1
For Practical Press 2
For Go to Back 3
2
enter size of array :8
enter element of array :1 7 3 4 5 9 5 2
enter element is to be insert :6
enter location where element is insert :3
The result is : 1 7 6 3 4 5 9 5 2
Press Enter To Go Back.....

INSERTION

For Theroy Press 1
For Practical Press 2
For Go to Back 3 1

Algorithm

INSERT_LA(a,n,item,loc)

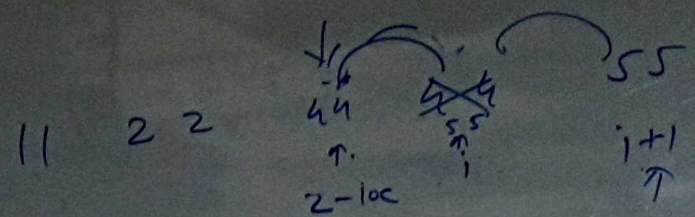
here a is linear array containing 'n' no of element
By using this algorithm we insert a given element item

at loc

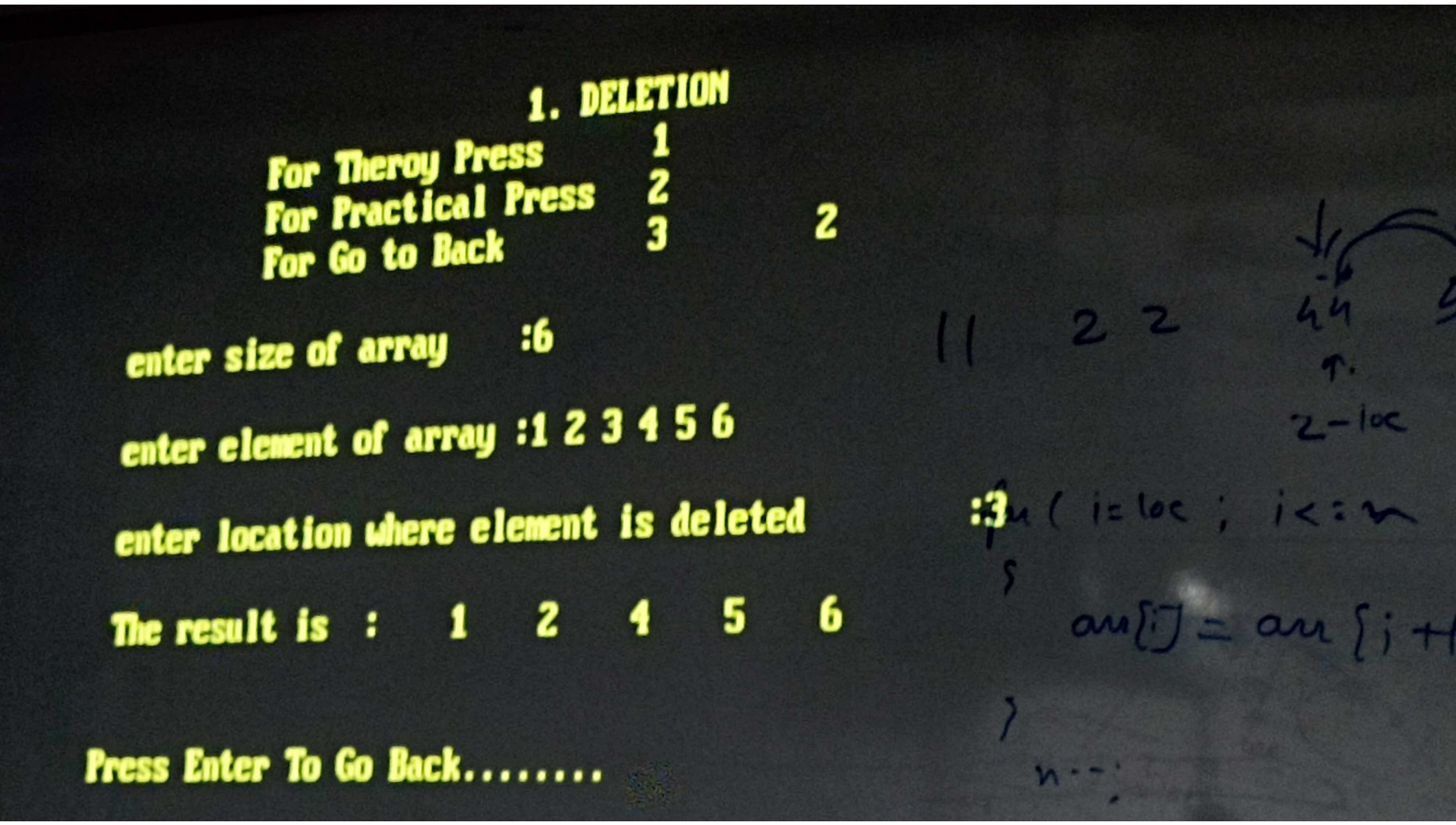
1. Set j=n
2. Repeat while(j>=loc) do
   a[j+1] = a[j]
   j = j-1
3. End while loop
4. a[loc] = item
5. Stop.

Go To Back Press Enter_

```
                    1. DELETION
          For Theroy Press      1
          For Practical Press   2
          For Go to Back        3        2

enter size of array    :6

enter element of array :1 5 3 78 8 3 6

enter location where element is deleted
The result is :   1   5   3   78   8

Press Enter To Go Back.....
```




```
:for ( i=loc ; i<=n
 {
    arr[i] = arr[i+1]
 }
 n--;
```


1. DELETION
For Theroy Press 1
For Practical Press 2
For Go to Back 3
2
enter size of array :6
enter element of array :1 2 3 4 5 6
enter location where element is deleted :3
The result is : 1 2 4 5 6
Press Enter To Go Back.....
for (i=loc ; i<=n
arr[i] = arr[i+
n--;

1. DELETION

For Theroy Press 1
For Practical Press 2
For Go to Back 3 1

Algorithm
DELETE_LA(a,n,item,loc)
here a is linear array containing 'n' no of element
By using this algorithm we delete a given item at loc

1. Set j=loc
2. Repeat while (j < loc) do
    1. a[j] = a[j+1]
    2. j = j+1
3. End while
4. Stop.

Go To Back Press Enter_

5. Stack, Queue, Linked List(Graphical)
6. Stack, Queue, Linked List(Theory)
7. Exit 1

Sorting :-

Sorting is a technique by using it we can sort the given element in the form of ascending or descending order

Sorting means arrengement of variable data in the form of ascending or descending order

Let us suppose that we have an array containing n number of element i.e A[1],A[2],A[3].....A[n].

Ascending Order :- If element is ascending order, if it is in following form A[0]>A[1]>A[2]>A[3].....>A[n-2]>A[n-1].

Descending Order :- If element is descending order, if it is in following form A[0]<A[1]<A[2]<A[3].....<A[n-2]<A[n-1].

Sorting Techniques are four types

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Merge Sort
5. Heap Sort
6. Go To Back

Select Your Best Choice. _

```
BUBBLE_SORT(a,n)

a is an array containing n numbers of elements
ving stes for I=1 To n-1
=1
While (j<=n-i)
  If(a[j]>a[j+1])
     temp=a[j]
     a[j]=a[j+1]
     a[j+1]=temp
  End If
  j=j+1
le loop
```

temp (9)

arr[ ]: { 8, 9, 3, 7, 5, 7 }

(swap) (temp)

```
for ( i=0 to n
    for( j=1 ; j<=n; j++ )
        if ( arr[j] > arr[j+1] )
            temp = arr[j]
            arr[j] = arr[j+1]
            arr[j+1] = temp
```

BUBBLRE SORT :-

Let us suppose that we have an array A containing n number of
i.e A[1],A[2],A[3].....A[n].

In Bubble Sort first we count number of element in the given array
If the Number of element is n the total passes is n-1

Algorithm

BUBBLE_SORT(a,n)

Where

a is an array containing n numbers of element

1. Repeat following stes for i=1 To n-1
    1. Set j=1
    2. Repeat While (j<=n-i)
        1. If(a[j]>a[j+1])
            temp=a[j]
            a[j]=a[j+1]
            a[j+1]=temp
        2. End If
        3. j=j+1
    3. End While loop
for loop
.op
. Press Enter

temp { 9 }

arr[ ]: { 8, 9, 3, 7, 5, 7 }

(swap) temp

for (i=0 to n

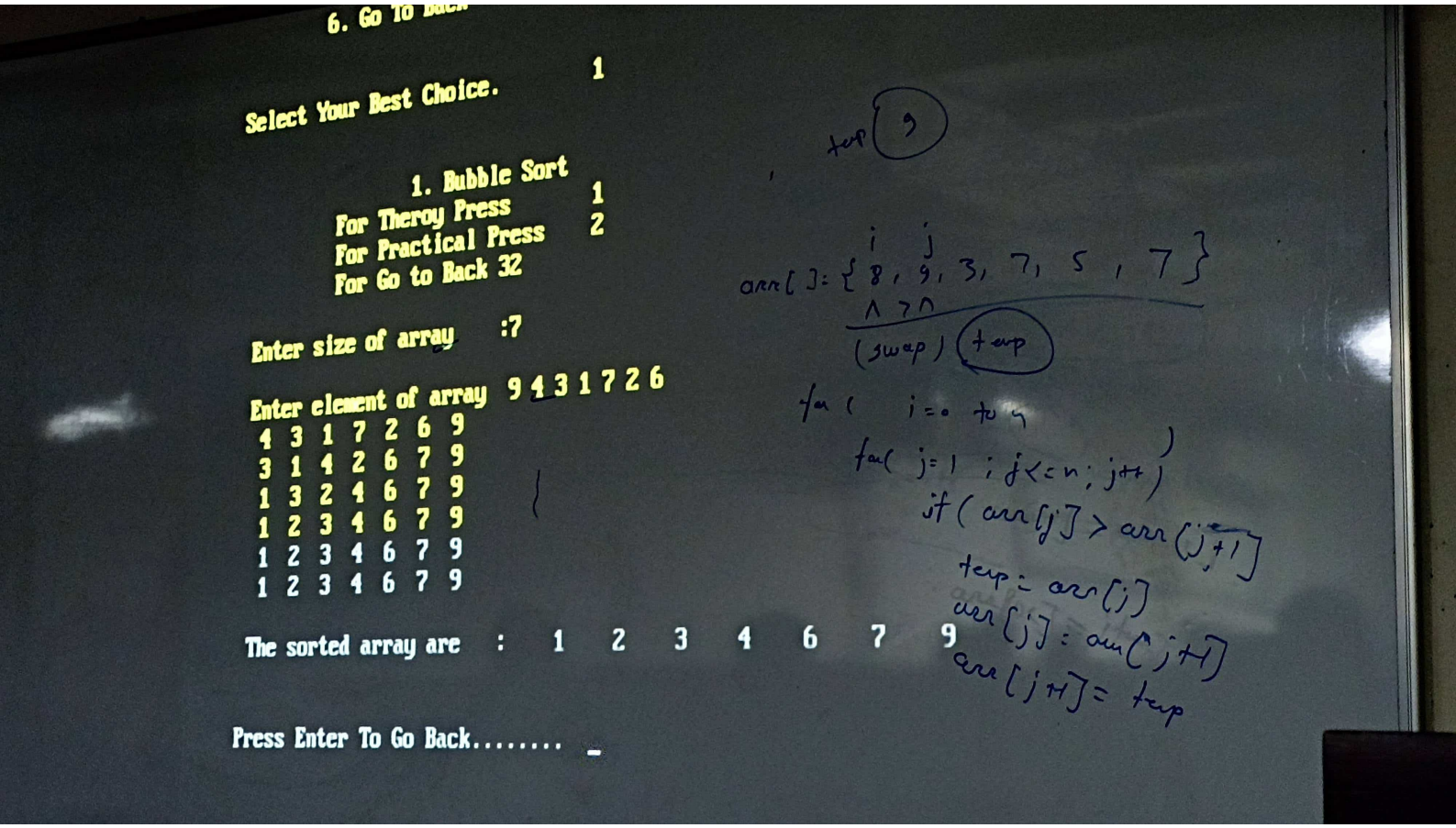
for(j=1 ; j<=n ; j++)

if (arr[j] > arr[j+1])

temp = arr[j]

arr[j] = arr[j+1]

arr[j+1] = temp


6. Go To Back
Select Your Best Choice. 1
1. Bubble Sort
For Theroy Press 1
For Practical Press 2
For Go to Back 32
Enter size of array :7
Enter element of array 9 4 3 1 7 2 6
4 3 1 7 2 6 9
3 1 4 2 6 7 9
1 3 2 4 6 7 9
1 2 3 4 6 7 9
1 2 3 4 6 7 9
1 2 3 4 6 7 9
The sorted array are : 1 2 3 4 6 7 9
Press Enter To Go Back..... _
temp
arr[ ]: { 8, 9, 3, 7, 5, 7 }
(swap)
temp
for(i=0 to n
for(j=1 ; j<=n; j++)
if(arr[j] > arr[j+1]
temp = arr[j]
arr[j] = arr[j+1]
arr[j+1] = temp

```
//insertion sort: ascending order
import java.util.Arrays;
// class InsertionSort{
//      public static void main(String[] args) {
//              int arr[]={5,17,3,9,15,1};
//              int loc,temp;
//              for(int i=1;i<arr.length;i++)
//              {
//               temp=arr[i];
//               loc=i-1;
//               while(loc>=0 && arr[loc]>temp)
//               {
//                arr[loc+1]=arr[loc];
//                loc--;
//               }
//                arr[loc+1]=temp;
//              }

//System.out.println(Arrays.toString(arr));
```

temp (9)

pos = i-1

(i-1) > (i)

temp

0 (1) (2)

arr[ ] = { 8, 9, 3, 7, 5, 7 }

∧ > ∧

(swap) (temp)

for (i = 0 to n

for (j = 1 ; j <= n ; j++)

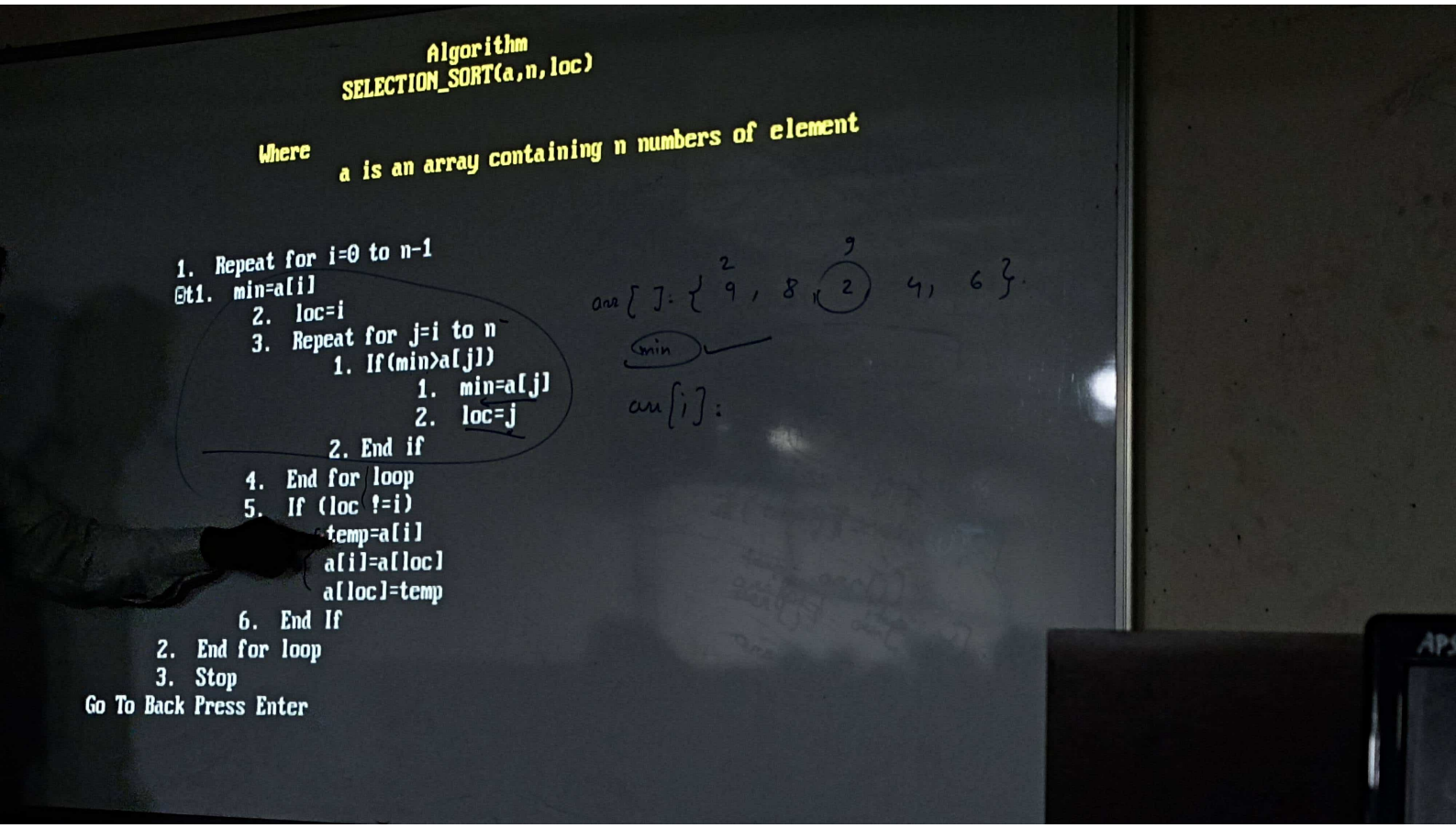
if (arr[j] > arr[j+1]

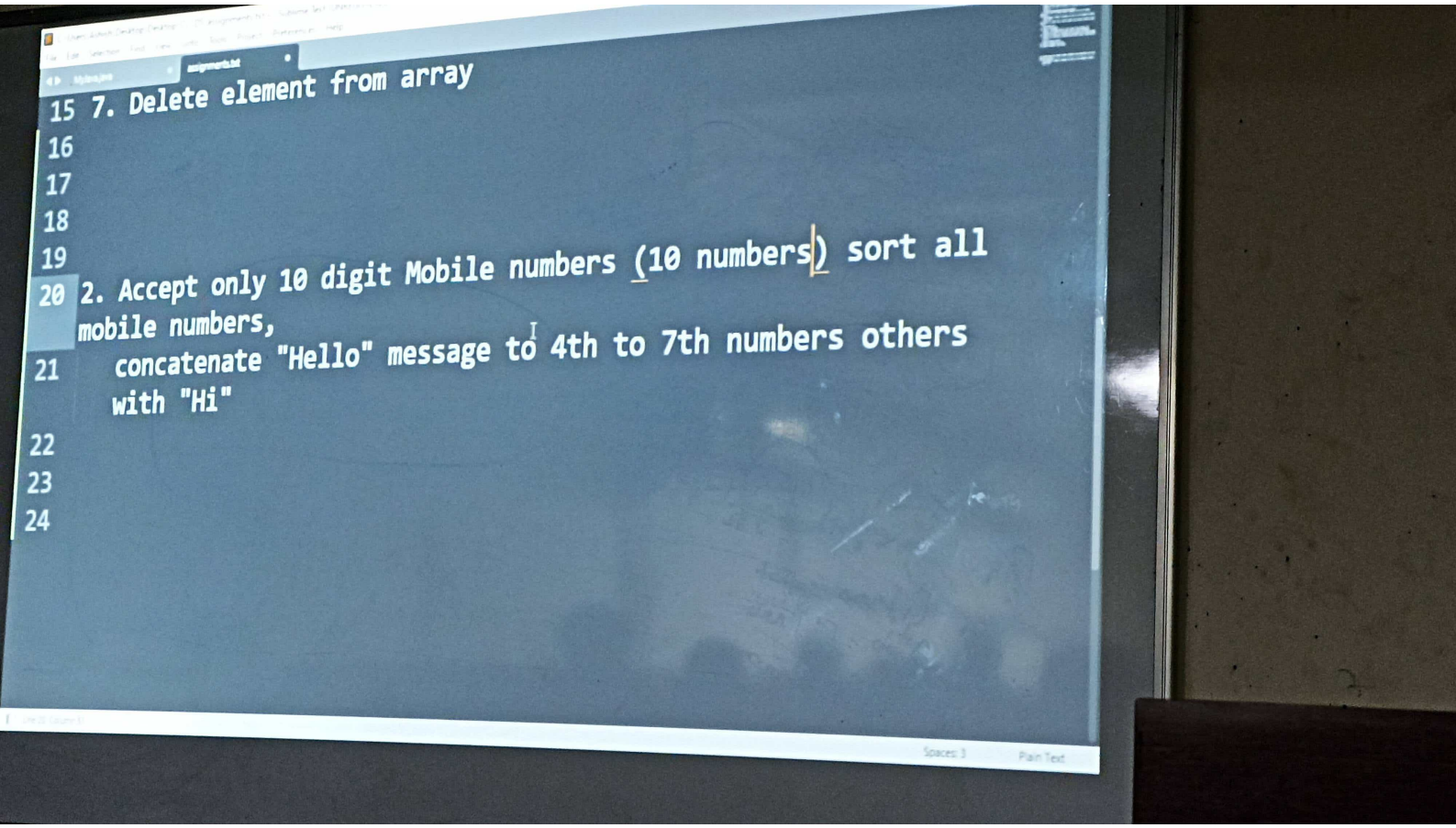
temp = arr[j]

arr[j] = arr[j+1]

arr[j+1] = temp

Ln 10, Col 13 100% Windows (CRLF) UTF-8

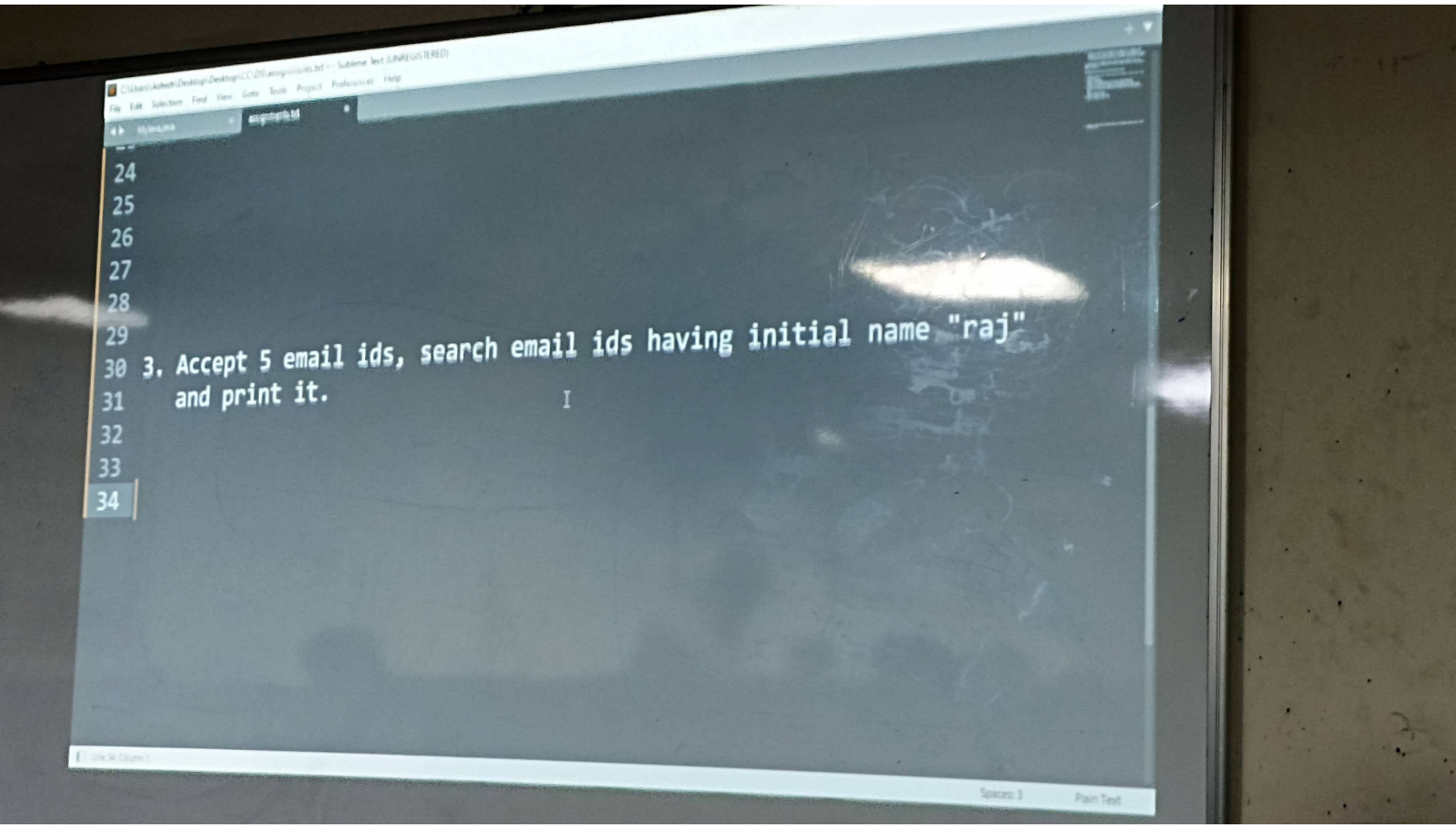

Algorithm
SELECTION_SORT(a,n,loc)
Where
a is an array containing n numbers of element
1. Repeat for i=0 to n-1
@t1. min=a[i]
2. loc=i
3. Repeat for j=i to n
1. If(min>a[j])
1. min=a[j]
2. loc=j
2. End if
4. End for loop
5. If (loc !=i)
temp=a[i]
a[i]=a[loc]
a[loc]=temp
6. End If
2. End for loop
3. Stop
Go To Back Press Enter
arr[]: { 9, 8, 2, 4, 6 }
min
arr[i]:

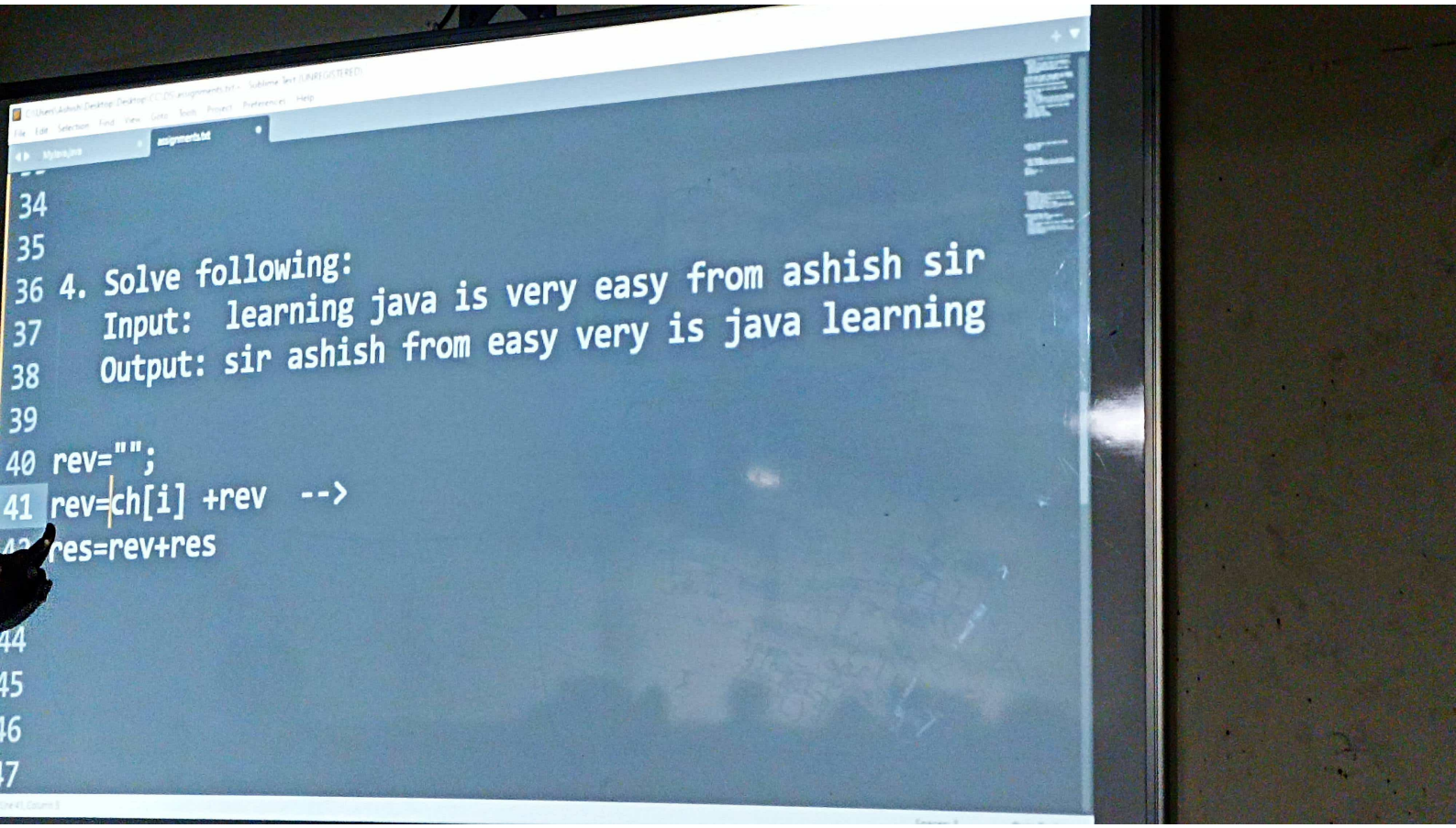

15 7. Delete element from array
16
17
18
19
20 2. Accept only 10 digit Mobile numbers (10 numbers) sort all
mobile numbers,
21 concatenate "Hello" message to 4th to 7th numbers others
with "Hi"
22
23
24

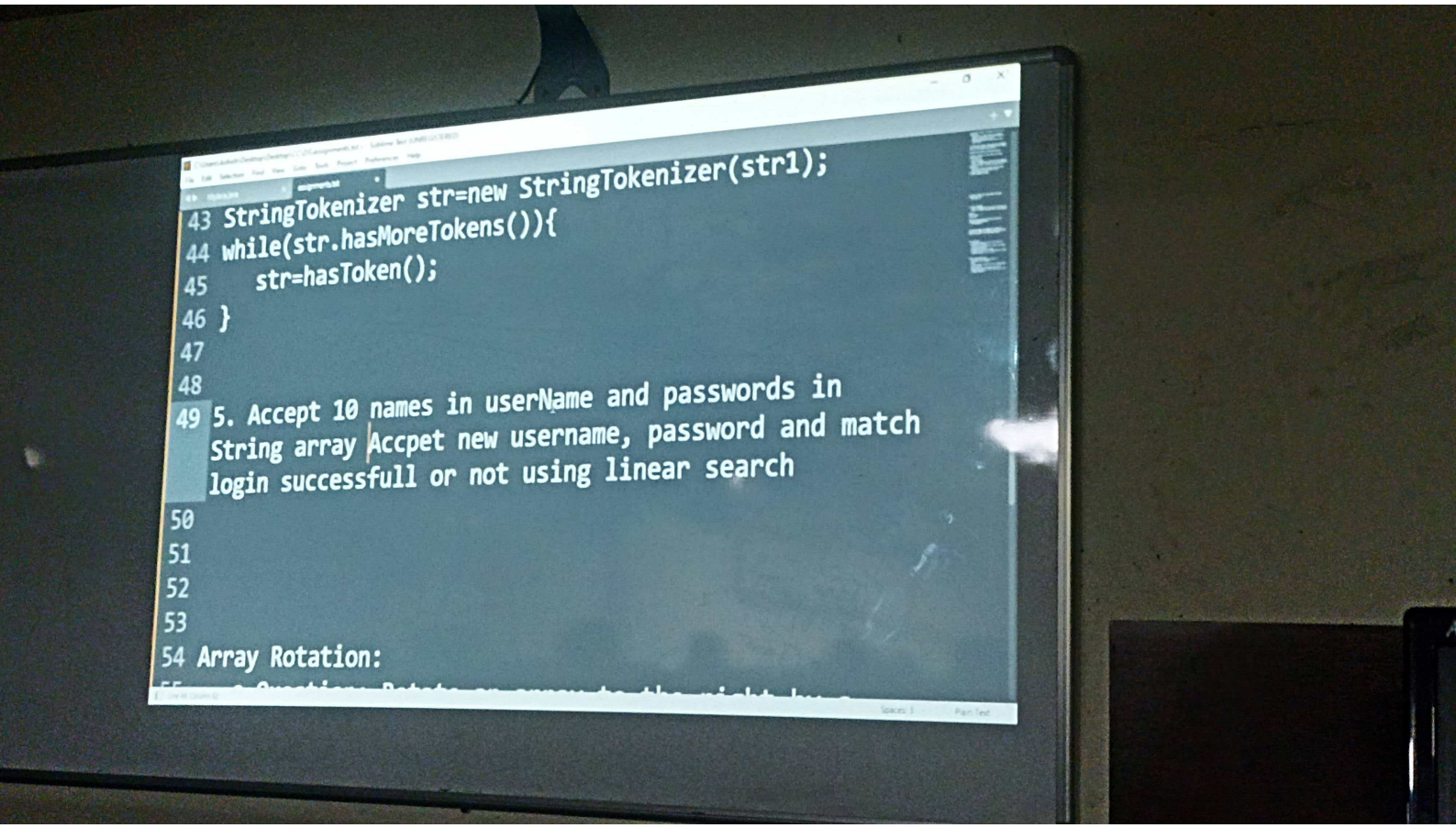
```
        1. Linear Search
For Theroy Press      1
For Practical Press   2
For Go to Back        3        1

                    Algorithm
                Linear_search(a,n,item,loc)
                here a is linear array containing 'n' no of element
                 item is element contain location loc
1.  Insert given item at the end of array
        a[n+1]=item
2.  [initialized]
        loc=1
3.  Repeat while( a[loc]! = item) do
        set loc=loc+1
4.  End while
5.  If( loc == n+1 ) then
        printitem not found
    else
        printitem found at,  loc
6.  End if
7.  Stop.
Go To Back Press Enter_
```

```
2.  Repeat while( item!=la[mid] ) and ( beg<end )  do
        1.  If(item<a[mid]) then
                End=mid-1
        2.  Else
                Beg=mid+1
        3.  End if
        4.  mid=int((beg+end)/2)
•.  End while
•.  If ( item==a[mid] ) then
        loc=mid
        print 'item found at, loc
5.  Else
        loc=null
        print 'item not found'
6. End If
7.  Stop.

Go To Back Press Enter_
```

```
26
27
28
29  1. Accept 10 students percentage and check who is topper using
30     selection sort. check topper's percentage is available in
       result[] if yes print location of matched percentage else print
       no
31     result[]={70,60,90,88,155,96,76,72,81} using binary search
32
```


24
25
26
27
28
29
30 3. Accept 5 email ids, search email ids having initial name "raj"
31 and print it.
32
33
34


34
35
36 4. Solve following:
37 Input: learning java is very easy from ashish sir
38 Output: sir ashish from easy very is java learning
39
40 rev="";
41 rev=ch[i] +rev -->
res=rev+res
44
45
46
47


StringTokenizer str=new StringTokenizer(str1);
while(str.hasMoreTokens()){
str=hasToken();
}
5. Accept 10 names in userName and passwords in String array Accpet new username, password and match login successfull or not using linear search
Array Rotation: