Page:
Date:

Java

Datatype

Primitive [8 types]

Byte	8 byte
short	
char	2 byte
Boolean	1 byte
int	- 4 byte
long	8 byte
float	- 4 byte
double	- 8 byte

Non primitive

- String
- Array
- class
- Object
- Interface

Priority operator

$* \; / \; \% > + \; -$

greater priority

Object creation with new Keyword.

```
int n = //user
for (i=0 ; i<n ; i++)
{
    n = n*i
}
    sout (n)
```

08.10.23

①

```
        c=5
      ------>
 r  | *  *  *  *  *
 =  | *  *  *  *  *
 4  | *  *  *  *  *
    v *  *  *  *  *
```

```
for (i=0 ; i<=4 ; i++) -
{ for (j=0 ; j<=9? ; j++)
  {
        s.o.P ('*');
  }
        s.o.P ("");
}
```

②

```
*   *   *   *   *
*               *      // boundary par * hai
*               *      cond: i=1 j=1 or
*   *   *   *   *            i=n j=m
```

```
for (i=0 ; i<=n ; i++)
{
    for (j=1 ; j<=m ; j++)
    {
        if (i==1 || j==1 || i==n || j==m)
        {
            SO.P(*)
        }
            SO.P ("");
    }
    SO.P ();
}
```

(3)

```
*
*  *
*  *  *
*  *  *  *
```

```
for (i=1 ; i<=n ; i++)
{ for (j=1 ; j<=1 ; i++)
  {
     S.O.P(*);
  }
  S.O.P();
}
```

(4)

```
*  *  *  *
*  *  *
*  *
*
```

```
for (i=n ; i>=1 ; i--)
{
   for (j=1 ; j<=i ; j++)
   {
      S.O.P( * );
   }
   S.O.P ();
```

(5)

```
_ _ _ *        3 space + 1*
_ _ * *        2 space + 2*
_ * * *        1 space + 3*
* * * *        0 sp + 4*
```

```
for (i=0 ; i<=n ; i++)
{
    for (j=1 ; j<=n-i ; j++)
    {
       S.O.P(" ");
    }
    for (j=1 ; j<=i ; j++)
    {
       S.O.P(*);
    }
}
```

(6)

```
1
0  1
1  0  1
0  1  0  1
1  0  1  0  1
```

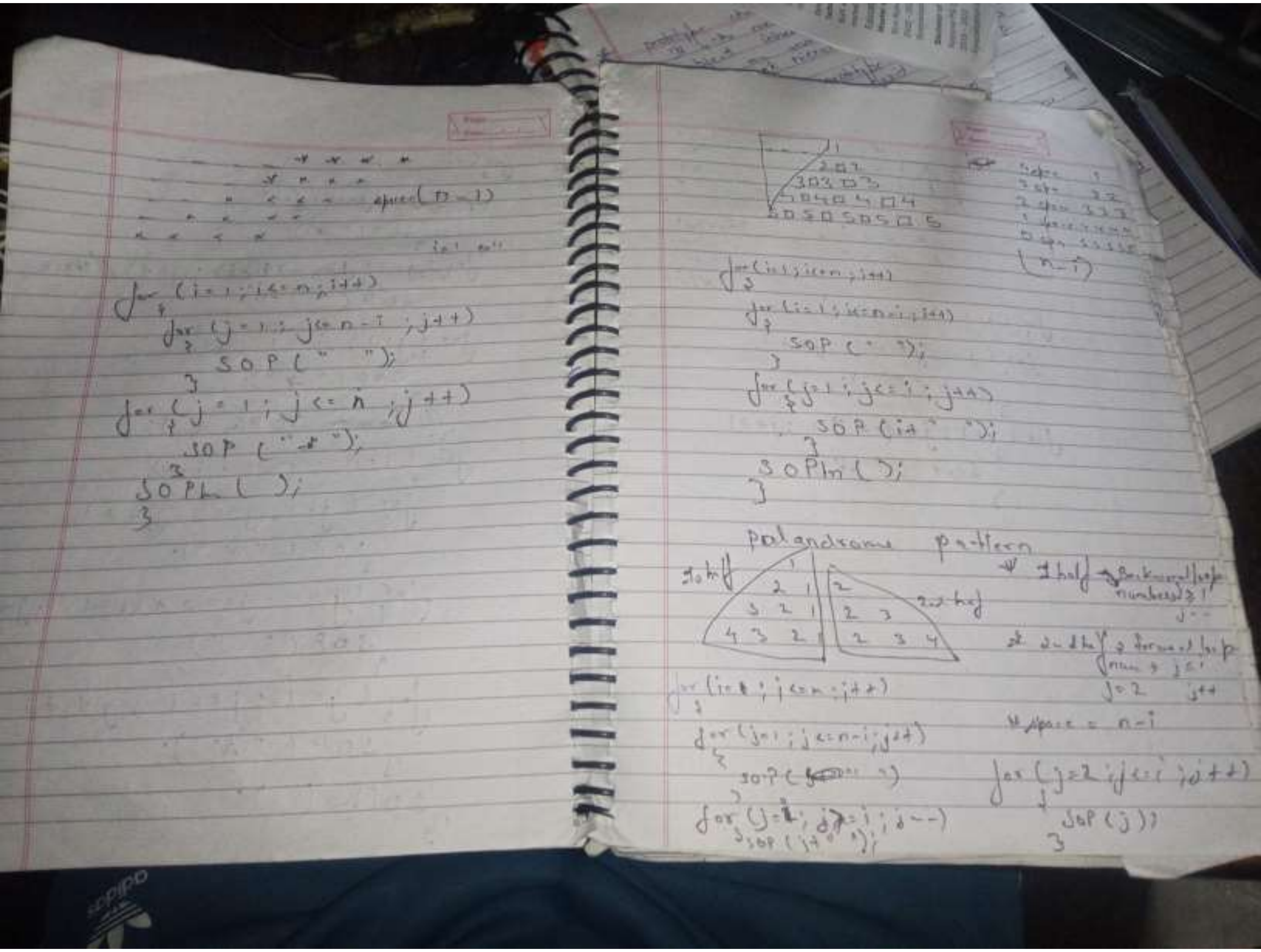
for (even sum = i+j) = 1
for odd = 0

```
for (i=1 ; i<=n ; i++)
{
    for (j=1 ; j<=i ; j++)
    {
       ~~S.O.P~~ int s = i+j ;
       if ( s%2==0)
       {
          S.O.P( 1);}
       else { S.O.P (0);
       }
    S.O.Pnl ()}
```

```
*          **        for(i=1;
**        ***        space = 2*(n-i)
***      ****
****      **         for(i=n; i>=1; i--)
**         *

for(i=1; i<=n; i++)
{ for(j=1; j<=i ;j++){ SOP("*")
  for( ~~for~~ j=1; j<=2*(n-i) ;j++)
  {
    SOUT(" ");
  }
  for(j=1 ; j<=i; j++)
  {
    SOT(" * ");
  }
  SOUT( );
//lower
for(j=n ; i>=i ; i--)
{
  for( j=i;j<2*(n-i) ;j++)
  {
    SOT("   "))
  }
  for( j=1 ; j<=i; j++)
  {
    SOT(" * ");
  }
  SOUT( );
}
}
```

```
for(i=1; i<=n; i++)
{
  for(j=1; j<=i ; j++)
  {
    SOP(" *");
  }
  for(j=1; j<= 2*(n-i) ; j++) //space
  {
    SOP("   ");
  }
  for(j=1; j<= i ; j++)
  {
    SOP(" * ");
  }
  // ~~[illegible]~~ SOPln ( );
}
for (i=n ; i>=1 ; i--)
{
  for (j=1; j<=i ; j++)
  {
    SOP(" * ");
  }
  for (j=1 ; j<=2*(n-i) ;j
  {
    SOP("   "),
  }
  for (j=1 ; j<=i ; j++)
  {
    SOP(" * ");
  }
  SOPln( );
}
```



```
- - - - * * * *
- - - * * * *
- - * * * *      space(n - 1)
- * * * *
* * * *
                 i=1 n=1

for (i=1; i<=n; i++)
{
   for (j=1; j<=n-i ; j++)
   {
       SOP("  ");
   }
for (j=1; j<=n ;j++)
{
     SOP ("*");
}
SOPln ( );
}
```

```
- - - - 1
- - - 2 □ 2
- - 3 □ 3 □ 3
- 4 □ 4 □ 4 □ 4
5 □ 5 □ 5 □ 5 □ 5
```

```
i space  1
2 sp  2 2
3 space  3 3 3
4 space  4 4 4 4
5 spa  5 5 5 5 5
(n-i)
```

```
for (i=1; i<=n; i++)
{
  for (j=1; j<=n-i; j++)
  {
     SOP (" ");
  }
  for (j=1; j<=i; j++)
  {
     SOP (i+" ");
  }
  SOPln ( );
}
```

Palandrome pattern

```
1st half          1
              2   1 | 2
          3   2   1 | 2   3           2nd half
      4   3   2   1 | 2   3   4
```

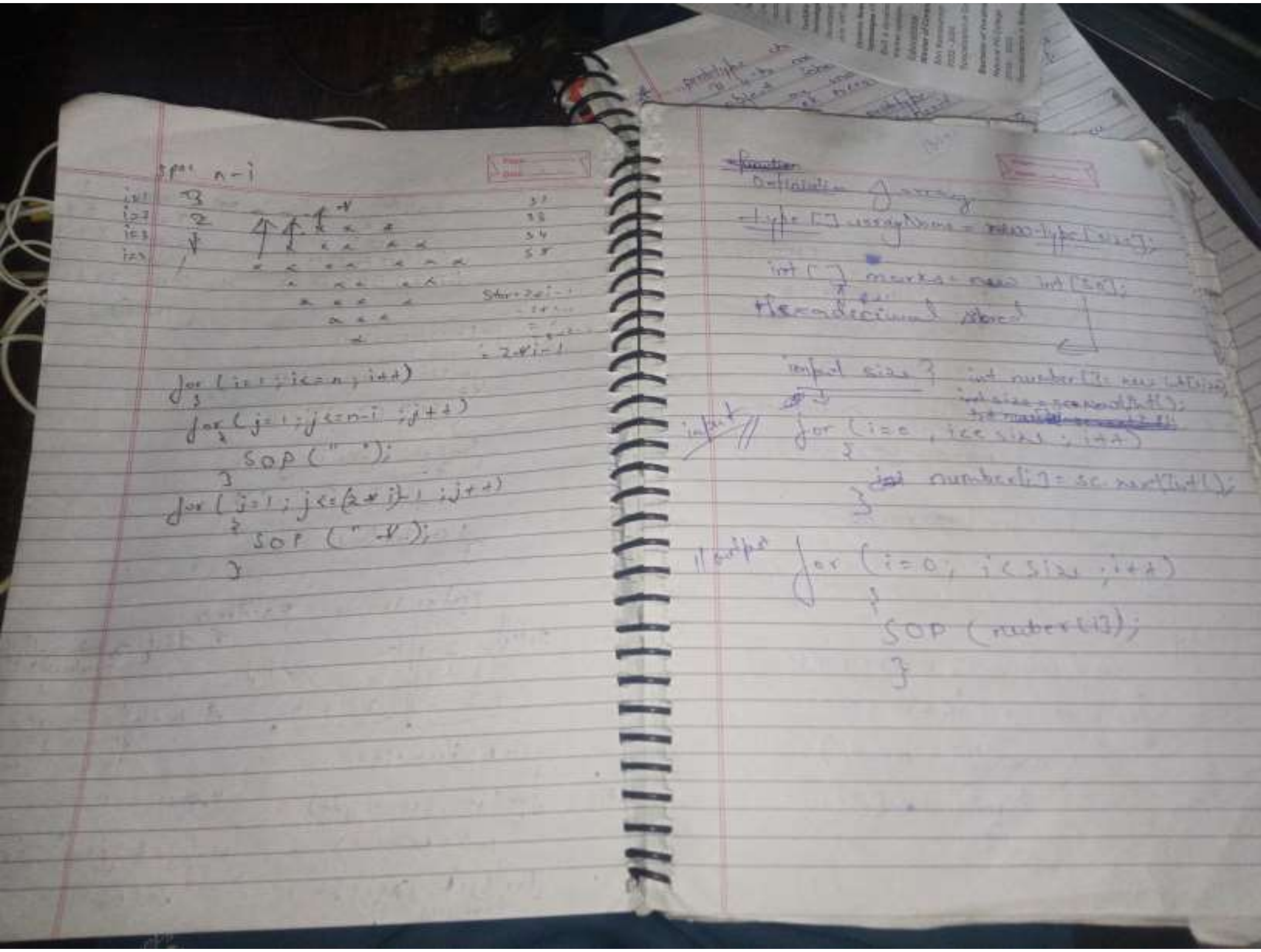
* 1 half → Backward loop
  numbers ≥ 1
  j--

* 2nd half → forward loop
  num → j ≤ i
  j = 2   j++

* space = n-i

```
for (i=1; i<=n; i++)
{
  for (j=1; j<=n-i; j++)
  {
     SOP(" ");
  }
  for (j=i; j>=1; j--)
  {
     SOP (j+" ");
  }
```

```
for (j=2; j<=i; j++)
{
   SOP (j);
}
```



space n-i

```
i=1  3
i=2  2
i=3  1
i=4

      *
    * * *
  * * * * *
* * * * * * *
  * * * * *
    * * *
      *
```

Star = 2*i-1

= 2*i-1

```
for (i=1; i<=n; i++)
{
  for (j=1; j<=n-i ;j++)
  {
     SOP (" ");
  }
for ( j=1 ; j<=(2*i)-1 ;j++)
  {
     SOP ("*");
  }
```

~~function~~

Definition of array

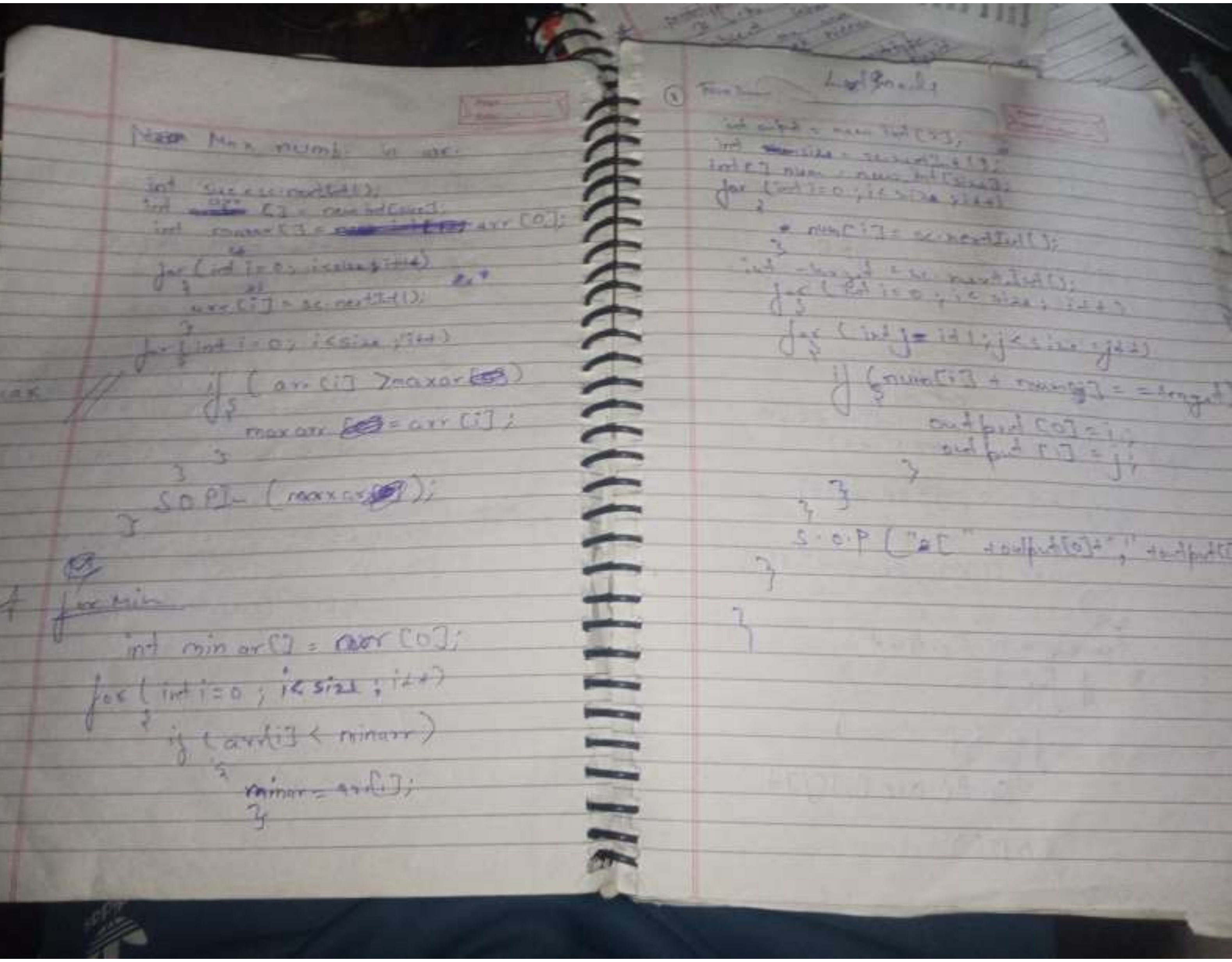
type [] arrayName = new type[size];

int [] marks = new int[30];

hexadecimal stored

```
input size }    int number[]= new int[size];
                ~~int size = sc.nextInt();~~
//input   for (i=0 , i<size ; i++)
          {
             ~~int~~ number[i]= sc.nextInt();
          }

//output  for (i=0; i<size ;i++)
          {
            SOP (number[i]);
          }
```



Max numb in arr.

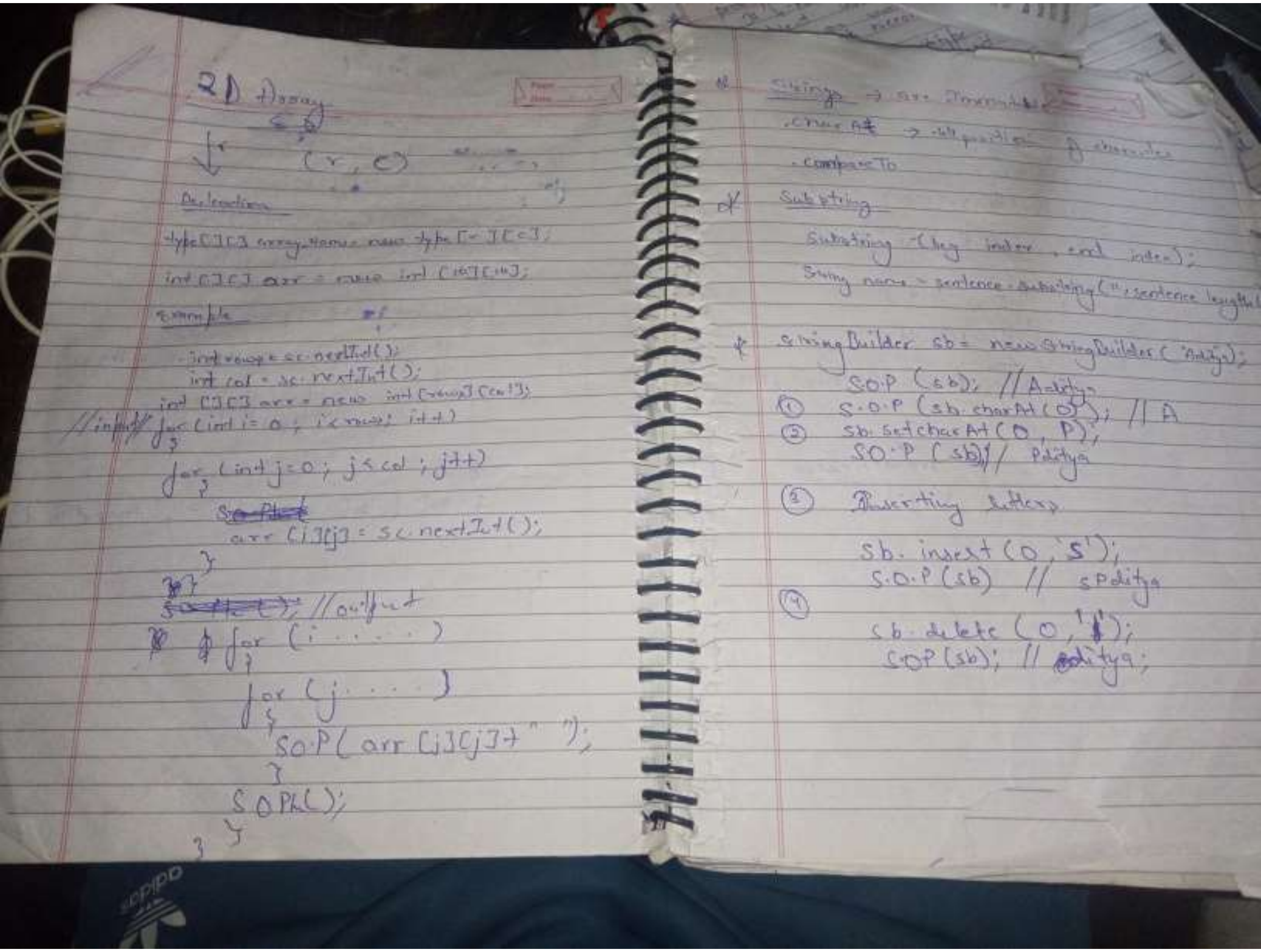
```
int size = sc.nextInt();
int arr [] = new int[size];
int maxarr[] = arr[0];
for(int i=0; i<size; i++)
{
    arr[i] = sc.nextInt();
}
for(int i=0; i<size; i++)
{
MAX //
    if (arr[i] >maxarr)
    {
        maxarr = arr[i];
    }
}
SOPln (maxarr);
}
```

for Min

```
int minarr[] = arr[0];
for(int i=0; i<size; i++)
{
    if(arr[i] < minarr)
    {
        minarr = arr[i];
    }
```

① Two Sum   Leetcode

```
int output = new int[2];
int size = sc.nextInt();
int[] num = new int[size];
for (int i=0; i<size; i++)
{
    num[i] = sc.nextInt();
}
int target = sc.nextInt();
for (int i=0; i<size; i++)
{
    for (int j= i+1; j<size; j++)
    {
        if (num[i] + num[j] = = target)
        {
            output[0] = i;
            output[1] = j;
        }
    }
}
S.O.P ("a[" +output[0]+"," +output[
}
}
```



2D Array

(r, c)

Declaration

```
type[][] arrayName = new type[r][c];
int[][] arr = new int[10][10];
```

Example

```
   int rows = sc.nextInt();
   int col = sc.nextInt();
  int [][] arr = new int[rows][col];
//input for (int i = 0; i< rows; i++)
  {
   for (int j=0; j<col; j++)
   {
      ~~S.o.P~~
      arr[i][j] = sc.nextInt();
   }
  }
  ~~S.o.P( );~~ //output
  for (i . . . . . )
  {
     for (j . . . . )
     {
       S.O.P( arr[i][j]+" ");
     }
    S.O.Pln();
  }
 }
```

Strings → are Immutable

.charAt → [illegible] position of character

.compareTo

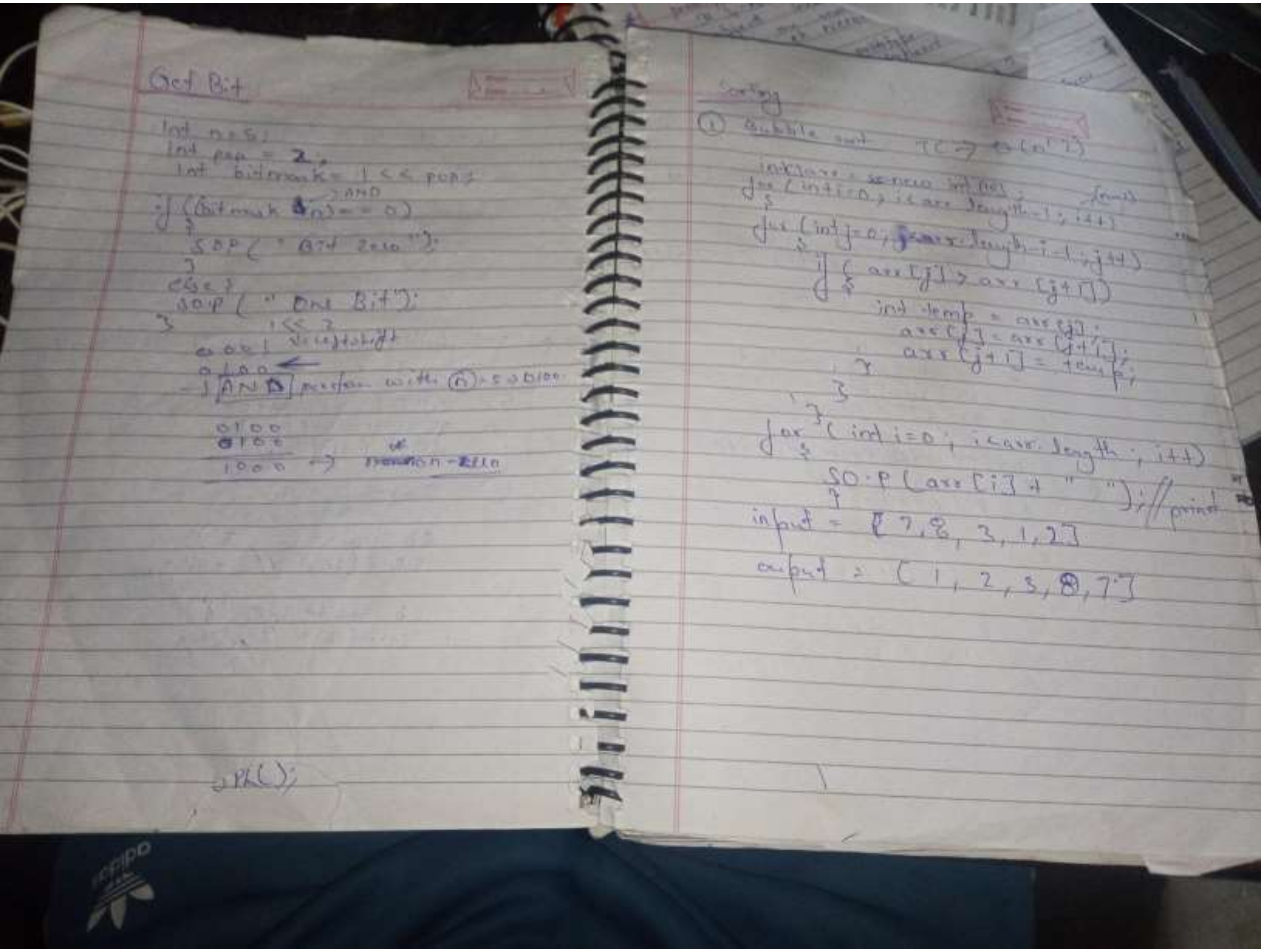
Substring

```
Substring (beg index, end index);
String name = sentence.Substring(" ", sentence.length());
```

```
StringBuilder sb = new StringBuilder ("Aditya");
     S.O.P (sb); // Aditya
①  S.O.P (sb.charAt(0)); // A
②  sb.setcharAt(0, 'P');
     S.O.P (sb)// Paditya

③  Inserting letters

     sb.insert(0, 'S');
     S.O.P(sb) // SPaditya
④
     sb.delete(0, '1');
     S.O.P(sb); // aditya;
```



Get Bit

```
int n=5;
int pos = 2;
 int bitmask = 1 << pos;
                 →AND
if (bitmask &n) == 0)
{
  SOP (" Bit zero");
}
else {
 SOP (" One Bit");
}
          1<< 2
   0001  [illegible]
   0100 ←
 → AND operation with (n) = 5 ⇒ 0100

   0100
   0100
   ----
   1000 → non-zero
```

=PK();

Sorting

① Bubble sort   TC → O(n^2)

```
int[] arr = new int[10];
for (int i=0; i<arr.length-1; i++)
{
 for (int j=0; j<arr.length-i-1; j++)
 {
    if ( arr[j] > arr[j+1])
    {
       int temp = arr[j];
         arr[j] = arr[j+1];
         arr[j+1] = temp;
    }
  }
}
for (int i=0; i<arr.length; i++)
{
    SO.P (arr[i] + "  ");//print
}
input = [7,8,3,1,2]
output = [1,2,3,8,7]
```

Selection (O) n^2

```
for (int i=0 ; i<arr.length-1 ; i++)
{
    int smallest = i;
    for (int j = i+1 ; j<arr.length ; j++)
    {
        = j;
        if (arr[smallest] > arr[j])
        {
            smallest = j;
        }
    }
    int temp = arr[smallest];
    arr[smallest] = arr[i];
    arr[i] = temp;
}
for (int i=0; i<arr.length; i++)
{
    SOP (arr[i]+"  ");
}
```

Insertion sort - n^2

```
for (int i=0 ; i<arr.length ; i++)
{
    int current = arr[i];
    int j = i-1;
    ~~while (j>=0 && current<arr[j])~~
while (j>=0 && current < arr[j])
{
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = current;
}
for (int i=0; i<arr.length; i++)
{
    SOP (arr[i]+"  ");
}
SOP();
}
```

Recursion → function that call itself

Iteration
Function

① Print No. from 5 to 1.

```
for (int i = 5 ; i>0 ; i++)
{
    Sout (i)
}
```

↓

to Recursive fun.

```
public static void printNum (int n)
{
    if (n==0) // Base case
        return ;
    syso (n);
    printNum (n-1); // recursion
}
p s v m ( )
{
    int n = sc.nextInt ( );
    printNum (n);
}
```

N.B. If base case is not defined the error occurr called stack overflow.

② Print sum of first n natural no.

```
p s v printnatural (int n, ~~int s~~, ~~[illegible]~~)   int i
{
    int sum = 0;
    int i = 1;
    if (i == n)
    {
        sum = sum + i;
        SOP (sum);
        return
    }          sum = sum + i;
    ~~print~~ natural (~~int s~~, n);
    } ~~[illegible]~~;

p s v m (
{
    print ~~print~~ natural ( 5);
}
```

because No empty or code below print natural.

(3) Print factorial of no : n..

1) Input - n=5
2) "Kaam" ? ⇒ n(n-1)!
   ⇒ n x (n-1)!
   return fact.
3) Base case. if n==1 or 1, return 1;

```
p s int calc_fact (int n )
{
  if (n==1 || n==0)
  { return 1;
  }
  int fact_nt = calc_fact (n-1);
  int fact_n = n * fact n1;
   return fact n;
}
p s v m ( )
{
  int n = sc. next In
  int ans = calc_fact(n);
  SOP (ans);
}
```

n=1
n=2
n=3
n=4
n=5
main n=5

2x1
3x2
3x2x4
3x2x4x5
120 in Multiplication

(4) Fibonacci series

(1) Input ⇒ 1st = 0
   2nd = 1
(2) "Kaam" ⇒ c = a+b
(3) Base case ⇒ 2nd last ...

```
p s v print fib( int n, int a, int b)
{                              ↳ 2 last term
  if (n==0)
  {
    return;
  }
  int c = a+b;
  SOP (c);
  print_fib ( b, c, n-1); //
}                  ↳ a=b, b=c
p s v m ( )
{
  int a = 0 , b = 1;
  SOP (a);
  SOP (b);
  int n = sc. nextInt ()
  print_fib (a, b, n-2);
}
```

already printed 0, 1

pf (5,8,0)
pf (3,5,1) = 8
pf (2,3,2) c = 5
pf (1,2,3) c = 3
pf (1,1,4) c = 2
pf (0,1,5) c = 1
Main n(7)

0,1,1,2,3,5,8

Tower of Hanoi

S1 :- n = 3

S2 :- String source, helper, destinatio

S3 :- Approach{
call TOH ((n-1) , src, destination)
to helper
leave last disk
(src, h, D) concept

SDH

S4 :- SOP ("transfer Disk"
+ n + " from " + src + " to " + destination);

S5 :- call TOH (n-1, helper, src, dest)

HSD

S6 :- Base case
```
if (n == 1)
{
  SOP (S4);
  return;
}
```

Boolean Sorted

```
public static boolean Sorted (int arr[], int index)
{
  if ( index == arr.length - 1)
  {
    return true;
  }
  ~~if (arr length~~
  if (arr[index] < arr[index+1])
    {
     return Sorted(arr, index+1);
    }
  else
    {
     return false;
    }
  }
psvm () {
  int arr[] = {1, 2, 3, 4, 5};
   SOPln (Sorted(arr, 0));
 }
}
```

Move all 'x' to the end

axbcxxd

```
public static void Movex (String str, int index, int count,
                          String newString)
{
    if (index == str.length())
    {
        SOP (newString);
        for (int i=0; i<count; i++)
        {  newString = newString + 'x';
        }
    }
    char currchar = str.charAt(index);
    if (currchar == 'x')
    {
        count ++;
    Movex (str, index+1, count, newString);
    }
else {
        newString = newString + currchar;
    Movex (str, index+1, count, newString);
    }
}
p s v m
{
    movex (~~str~~ str, 0, 0, "");

}
```

Remove duplicates

```
public static boolean[] map = new boolean[26];
p s v Remove (String str, int index, String
                                  newString)
{
    if (index == str.length())
    {
        SOP (newString);
        return;
    }
    char currchar = str.charAt(index);
    if (map[currchar - 'a'] == true)
    {

        Remove(str, idx+1, newString)
    }
    else
    {
    newString += currchar;
    map[currchar - 'a'] = true;
    remove (str, index+1, newString)
    }
}
p s v m ()
{  string str = "abbccda";
     remove (str, 0, "");
}
}
```

O(2^n)

Subsequences of char

```
p.s v. subsequence (String str, int idx
                    String newString)
{
  if (idx == str.length())
  {
   SOP( newString );
   return ;
  }
char currChar = str. charAt (idx);
// to be
subsequence (str, idx+1, newString+currChar)
// not to be
subsequence (str, idx+1, newString)
}
p.s.v.m ( )
```

uniqueSubsequences

we use hashset
HashSet

```
public static void unique (String str, int idx, String newString, HashSet<String> set)
  if (idx == str.length())
  {
   ~~SOP (newString);~~
   ~~return;~~
  }
  char currchar = str.charAt(idx);
   if (set.contains(newString))
   {
      return ;
      }
      else
      { SOP (newString
        set.add (newString);
        return;
      }
   }
  char currchar = str.charAt(idx)
  uniqueSeq (str, idx+1, newString+currchar, set );
  uniqueSeq (str, idx+1, newString , set );
}
psvm ()
 String str = "aaa" ;
  HashSet <String> set = new HashSet<>
  uniqueSeq (str, 0,
 }
```

$O(4^n)$

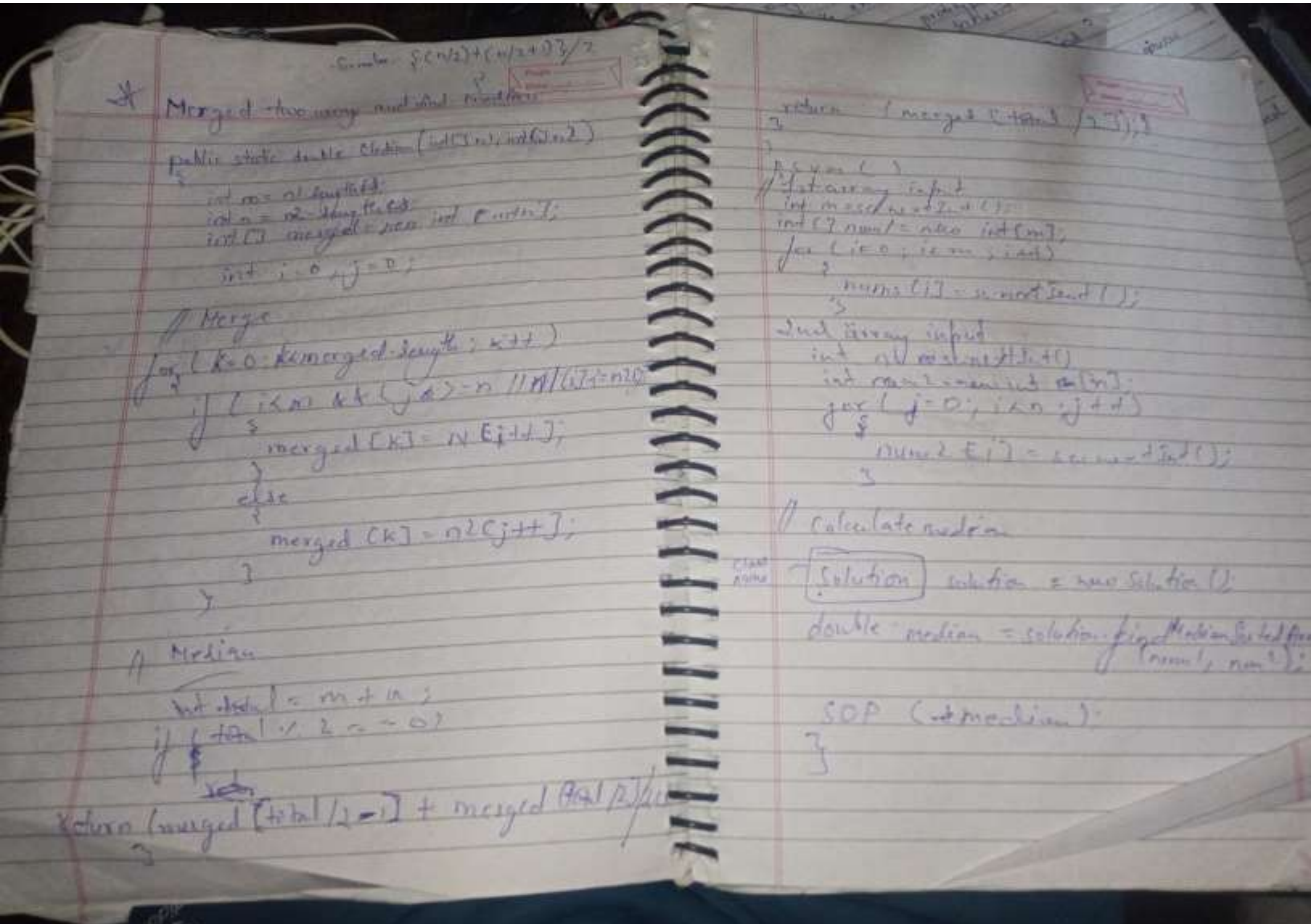
Mobile keypad

```
public static ~~keypad~~
        string[] keypad = {"", "", "abc", "def",
    "ghi", "jkl", "mno", "pqr", "stu", "vwx",
    "yz"};
ps v printcombo(string str, int idx, string comb)
    {
      if (idx == str.length())
      {
        sop(comb);
        return;
      }
  char currchar = str.charAt(idx);
  string mapping = keypad[currchar - '0'];
for (int i = 0; i < mapping.length(); i++)
{
   printcombo(str, idx+1, comb + mapping.charAt(i));
 }
}
psvm()
   string s = "2";
   printcombo(s, 0, "");
}
```

$O(n!)$

permutation of string

```
psv printpermutation(string str, string perm)
    if (str.length() == 0)
    {
        SOP(perm);
        return;
    }
  for (int i = 0; i < str.length(); i++)
  {
     char currchar = str.charAt(i);
     string newstr = str.substring(0, i) + str.substring(i+1);
     printpermutation(newstr, perm + currchar);
   }
}
```



Formula : {(n/2)+(n/2+1)}/2

* Merged two array and find median

```
public static double Median(int[] n1, int[] n2)
{
    int m = n1.length();
    int n = n2.length();
    int[] merged = new int [m+n];

    int i = 0, j = 0;

// Merge
for (k=0; k<merged.length; k++)
{
    if (i<m && (j>=n || n1[i]<=n2[j])
    {
        merged[k] = n1[i++];
    }
    else
    {
        merged [k] = n2[j++];
    }
}

// Median

    int total = m + n;
    if (total % 2 == 0)
    {
        ~~retn~~
return (merged [total/2-1] + merged [total/2])/
    }
```

```
return (merged [total/2]);}
}
}

psvm ( )
// 1st array input
int m = sc.nextInt();
int[] num1 = new int[m];
for (i=0; i<m; i++)
{
    nums [i] = sc.nextInt();
}
2nd array input
    int n = sc.nextInt()
    int num2 = new int [n];
    for ( j=0; i<n; j++)
    {
        num2 [i] = sc.nextInt();
    }

// Calculate median

class name
Solution solution = new Solution();

double median = solution.findMedianSortedArr
                        (num1, num2);

    SOP (median);
}
```

10 (121 /
10

21　　2

21
10

d = 10/121 % 10 / 2

d = 21

r = 0 % 10 + 1

x = x / 10
= 121 / 10
x = 12

r=1, d=12, x=0
d = 12 % 10 → 2

Palindrome

```
public boolean ispalindrome( int x)
{
  if (x<0)                    //negative
  {
    return false;
  }

  int original = x;
  int rev = 0;

  while ( x>0 )
  { int digit = x%10;
      rev = rev * 10 + digit;
        x = x / 10;
  }
    return original == reve

}
  p s v m ( )
  { int x = 121;
    if (isph (x))
    {
      SOP ( "it is pal")
    }
    else
    {
      SOP ( "it is notpal
    }
  }
}
```

Roman-to Integer

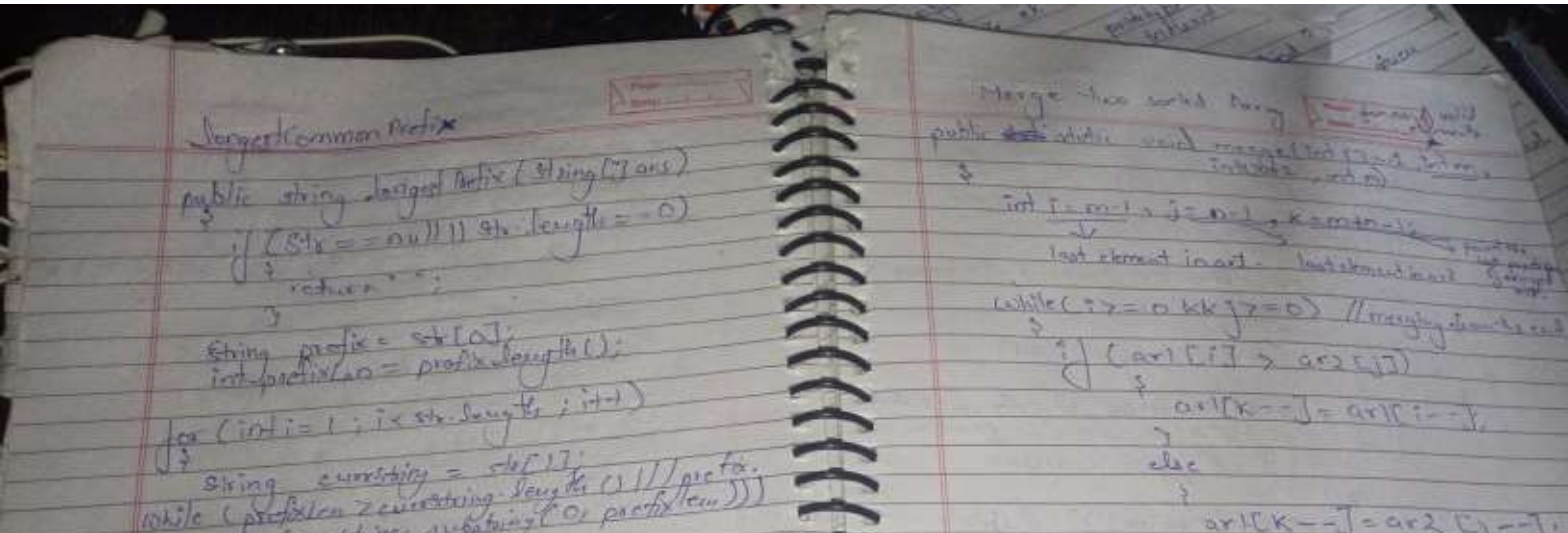
```
public static int AddSub(char c, char a1, char a2)
{
  return (c == a1 || c == a2) ? -1 : 1;
}
public int romanToInt(String s)
{ int result = 0;
for (int n = 0; n < s.length(); n++)
{
  char nextchar = (n+1) < s.length() ?
          s.charAt(n+1) : '\0';

switch(s.charAt(n))
{
  case 'M' : ~~result~~
       result += 1000; break;

  case 'D' :
       result+= 500; break;
  case 'C' :
       result += 100 * AddSub(nextchar,
                         'M', 'D');
       break
                       scalfactor = -1

case 'L' :
     result += 50;
          break;
case 'X'
     result += 10* ~~cal~~AddSub
               (nextchar 'C', 'L');
     break;

case 'V' :
       result += 5 ;
          break;
case 'I' :
       result += 1* AddSub(nextchar, 'X', 'V'
          break;                  nex
  }
return result;
}
```



Longest Common Prefix

```
public string longestPrefix (String[] ans)
{
    if (Str = = null) || str. length = = 0)
    {
        return " ";
    }
    String prefix = str[0];
    int prefixLen = prefix.length();

for (int i = 1 ; i < str. length ; i++)
{
    String currstring = str[i];
while (prefixLen > currstring. length () || !prefix.
equals (currstring. substring (0, prefixLen)))
{
        prefixLen - - ;
        if ( ~~prefix~~ prefixLen = = 0)
        {
            return " ";
        }
    prefix = prefix . substring (0, prefixLen);
  }
 }
} return prefix ;
}
```

LOK

Merge two sorted Array

```
public ~~void~~ static void merge(int [] ar1, int m,
                        int [] ar2 , int n)
{
    int i = m-1 , j = n-1 , k = m+n-1 ;
         last element in ar1.   last element in ar2

    while ( i >= 0 && j >= 0 ) // merging elements end
    {
        if (ar1 [i] > ar2 [j])
        {
            ar1[k--] = ar1[i--];
        }
        else
        {
            ar1[k--] = ar2 [j--];
        }
    }

    If element remain in ar2 the

    while (j >= 0)
    {
        ar1 [k--] = ar2 [j--];
    }
  }
}
```

Remove Duplicate from Array :

```
public static int remove(int[] nums)
{
    if (nums.length == 0)      // Base case
    {
       return 0;
    }
    int i = 0; //pointer to track the unique
                  element.

for (int j=0 ; j< nums.length ; j++)
{
    if ( nums[j] != nums[i] ) //found but
    {                             no action
         i++ ; // just increment
         nums[i] = num[j]; //update
                    array with unique
                      elements.
    }
}
return i+1;
}
}
```

Remove Element

```
public static int remove(int[] nums, int val)
{
     int  i=0;
     for (int j=0; j<nums.length ; j++)
     {
          if (nums[j] != nums[i])
          {
               ~~[illegible]~~
               num[i] = num[j];
                 i++;
          }
     }
     return i;
  }
}
```

1,2,3, []
+1

1,2,4

Index of first occurrence

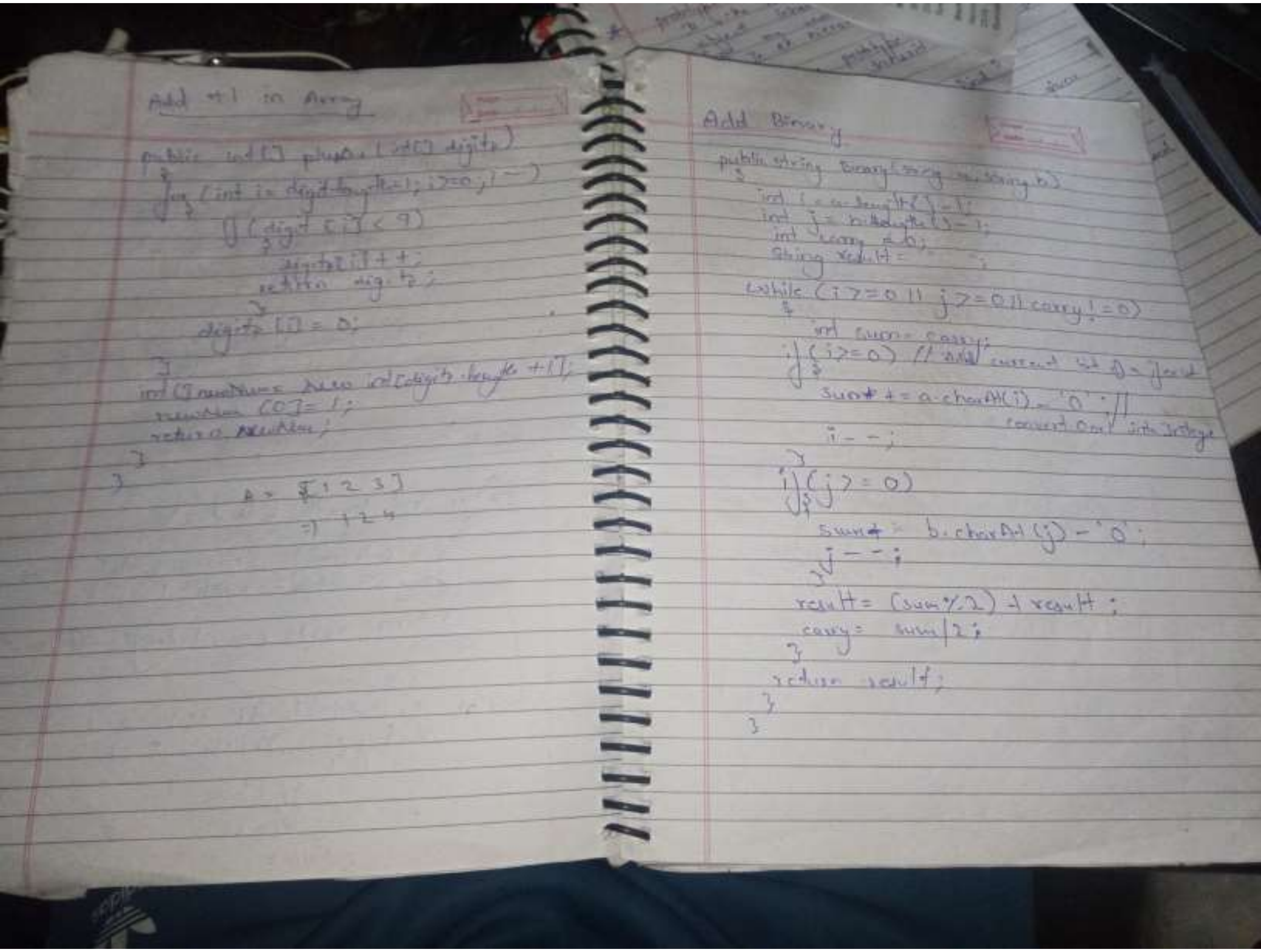
```
public ~~static~~ static int ~~string~~ occurrence
        (String str, String needle)
  {
    if (needle.isEmpty())
    {
        return 0;
    }
  for(int i=0; i<= str.length() - needle.length();
          i++)
     {
       if (str.substring(i, i+needle.length()).
             equal(needle)
         {
          return i;
         }
     }
   return -1;
  }
}
```

Search Insert Position

```
public static int Search(int[] num, int target)
  {
    for (int i=0; i<num.length; i++)
     {
        if (num[i] >= target)
          {
             return i;
          }
     }
    return num.length; //target should be inserted at end
  }
}
```

Length of last Word

```
public int length (String s)
  {
     s = s.trim(); // removing trail spaces
     String [] word = s.split(" ");
     return words[word.length - 1].len
  }
}
or    int c=0;
      for (int i=s.length()-1; i>=0; i--)
      {
        if (s.charAt(i)!=' ')
         {
           c=c+1;
           else
           break;
         }
      return c;
```



Add +1 in Array

```
public int[] plusOne (int[] digits)
{
  for (int i= digit.length-1; i>=0; i--)
  {
     if (digit [i] < 9)
     {
        digits[i] + +;
       return digits;
     }
   digits [i] = 0;

 }
int [] newNum= new int[digits.length +1];
newNum [0]= 1;
return newNum;
 }
}
```

A = [1 2 3]
=> 1 2 4

Add Binary

```
public string Binary(string a, String b)
{
  int i = a.length() - 1;
  int j = b.length() - 1;
  int carry = 0;
  String result = "  ";
 while (i>=0 || j>=0 || carry != 0)
 {
    int sum= carry;
   if (i>=0) // Add current bit of a if exist
   {
     sum += a.charAt(i) - '0'; // convert 0 or 1 into integer
     i - - ;
   }
  if (j>= 0)
  {
     sum += b.charAt (j) - '0';
     j - - ;
   }
  result= (sum%2) + result ;
  carry= sum/2;
 }
 return result;
 }
}
```

4 climbing stairs.

```
public in climb (int n)
{
    if (n <= 2)
    {
        return n;
    }
    int first = 1 , secound = 2;
for (int i = 3; i <= n; i++
  {
    int   current = first + second
          second = current;
  }
return second;
}

}
```

Sort - the peoples

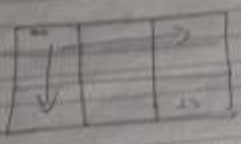
```
public string[] sortpeople(string[] names , int[] heights)
{
    int n = names. length();
  String[][] people = new String [n][2];
for (int i = 0, i < n ; i++)
  {
      people [i][0] = name [i];
      people [i] [1] = String .valueOf (height
  }
  bubbleSort (people);

  String [] sortedname = newString[n];
    for (int i = 0 ; i < n ; i ++)
     {
        sortedname [i] = people[i][0];
     }
    return sortedname;
  }
```

{1 , 2 , 3}

count total path

n = 3 , m = 4



```
public static int move (int i, int j, int n, int m)
{
    ~~int~~ if (i==n || j==m)
    {
        return 0;
    }
    if (i==n-1 || j==m-1)
    {
        return 1;
    }

    ~~int down path = move (i+1, j+1 n~~
int downpath = move (i+1, j, n, m);
int rightpath = move (i, j+1, n, m);
return downpath + rightpath;
}
```

Place tiles of size 1×m in floor of size n×m

```
public static int placetiles(int n, int m)
    if (n==m) ~~return~~
    {
        return (2) — ways
    }
    if (n<m)
    {
        return 1;
    }
    int vertPlacement = placetiles(n-m, m);
    int horizontPlacement = placetiles(n-1, m);
    return v+h;
}
```

Find the no. of ways in which invite n people to your par
or in pairs.

```
public static int callguest( int
{
    if (n <= 1)
    {
        return 1;
    }
    int way1 = call guest (n
    int way2 = (n-1) * callg
    return way1 + way
}
```

Count-total path

n=3 , m=4

```
public static int move (int i, int j, int n, int m)
{
    ~~int~~ if (i==n || j==m)
    {
        return 0;
    }
    if (i==n-1 || j==m-1)
    {
        return 1;
    }
~~int down path = move (i+1, j+ n~~
int downpath = move (i+1, j, n, m);
int rightpath = move (i, j+1, n, m);
return downpath + rightpath;
}
```

Place tiles of size 1xm in a floor of size nxm

```
public static int placetiles (int n, int m)
    if (n==m) ~~return~~
    {
        return 2 — way
    }
    if (n<m)
    {
        return 1;
    }
    int vertPlacement = placetiles(n-m, m);
    int horizontPlacement = placetiles(n-1, m);
    return v + h;
}
```

Find the no. of ways in which you can invite n people to your party, single or in pairs.

```
public static int callguest( int n)
{
    if (n <= 1)
    {
        return 1;
    }
    int way1 = callguest (n-1);
    int way2 = (n-1) * callguest (n-2);
    return way1 + way2;
}
```

* Print all the subset of a set of first n natural no.

```
public static void findsubset (int n, ArrayList<Integer> subset)
{
    subset.add(n);                 // add hoga
    findsubsets(n-1, subset);

    subset.remove(subset.size()-1);
    findsubsets(n-1, subset);      // add nahi hoga

    // Base case
    if(n==0)
    {
      printsubset(subset) // function
      return;
    }
p.s.v printsubset (Arraylist<Integer> subset)
{
  for (int i=0; i<subset.size(); i++)
  {
    SOP(subset.get(i)+"   ");
  }
  SOPln();
}

p.s.v.m ()
{
  n = 3;
Arraylist<Integer> subset = new Arraylist<>();
findsubset(n, subset);
}
```

Overloading

* Occurs within one class

* Name of method same but different parameter

* Return type can be same or diff.

* Ex:- compile-time polymorphism

Overriding

Occurs in inheritance (Subclass & Superclass)

* Name & Parameter both same.

Return type is always same.

* Ex:- Run time polymorphism.

```
class ABS
{
  public static int sum(int x, int y)
  {
    return x+y;
  }
  public static int sum(int x, int y, int z)
  {
    return x+y+z;
  }
p.s.v.m ()
{
  SOP(sum(2,3));
  SOP(sum(2,3,7));
}
```

```
class Animals
{
  public void eat()
  {
     print(Animal eating)
  }
}
class Dog extends Animals
{
  public void eat()
  { super.eat();
     print(Dog eating)
  }
}
Animal a1 = new Animal
   a1.eat();
Dog Animal a2 = new
   a2.eat();
```

runtime polymorphism

Polymorphism which exist at the time of execution of program is called runtime poly.
// method overriding we cannot perform overloading.

[illegible]
① [illegible]
② java Demo
③ java Demo

① call by value → calling method with parameter as value. // Here argument value is passed to the parameters.

```
int class Add
{
  print sum (int a, int b)
  {
    return a+b;
  }
ps vm ( )
{
  int x = 10 ; y = 20 ;
  int c = sum (x, y);
  SOPln ("sum" + c);
}
```

formal

actual param

② call by reference

[illegible] method [illegible] reference [illegible] original [illegible] the parameter.

class [illegible]

```
void swap (int a[])
{
  int t = a[0];
  a[0] = a[1];
  a[1] = t;
}
psvm ( )
{
  int a[] = {10, 20};
  swap (a);
  for (int i=0 ; i<=1 ; i++)
  {
    SOP (a[i]);
  }
}
}
```

pass by Reference	pass by value
means the actual object is not passed rather a reference is ~~pass~~ of object passed. ~~This~~ any changes made by external method, are also reflected in all place.	When a object is passed by value, mean the copy of an object is passed. effect of the changes are made to that object it doesn't affect the original value.

Backtracking

```
public static void printPermutation(String str, String perm,
                                                 int idx)
for (int i=0; i<str.length(); i++)
{
    char currChar = str.charAt(i);   [illegible]
    String newstr = str.substring(0, i) +
           str.substring(i+1);
    printPermutation(~~[illegible]~~ newstr, perm+currChar, idx+1);
    if (str.length == 0)
    {
        SOP(per);
    }  return;
}
```

A B ~~C~~

A C B

B A C

B C A

C A B

C B A

level

Single level inheritance

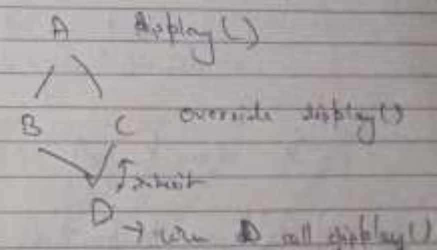
```
class shape
{
    public void area()
    {
        SOP("display area");
    }
}
class Triangle extend Shape
{
    P V area(int l, int h)
    {
        SOP( 1/2 * l * h);
    }
}
class Eq.Triangle extend Triangle
{
    p v area (int l, int h)
    {
    SOP(1/2 * l * h);
    }
```

Hierarchical I.V.

```
Base class Shape
class Triang extend Shape
{
}
class Circle extend Shape
{
}
```

Multiple Inheritance
↳ Issues Interface

↓ Do not support in Java it's due to avoiding the complexity of name ambiguity that cause the diamond problem.



→ when D call display() then compiler cannot decide whether to the method from B or C.

Java is not Pure OOPS?

1) Primitive DT
int, float, char, boolean
↓
not an object
as they are not.

So java introduced wrapper classes (Integer, Float) to treat primitive ~~class~~ as object when needed but primitive themselves are not object.

2) Static Method
Math.sqrt()
Integer.parseInt()
can be invoked without creating instance of a class.
→ called without creating a Math object.

3) Lack of Multiple Inheritance

Java import & package concept are structural and not based on objects.

Encapsulation

provides object with
→ ability to hide their internal characteristic & behaviour

Example

```
public class Employee
{
   private String name;
      public String getName()
      {
         return Name;
      }
      p s v setName(String name)
      {
         this.name = name;
      }
}
public class Test
{
  p s vm (String[]args)
  {
     Employee e = new Employee();
     e.setName ("John");
     SOP (e.getName());
   }
}
```

Inheritance

we can put the common behaviour and characteristic in a base class then all object with common behv. inherit from the base class.

Object class is the super class

Polymorphism

It provide a single interface that work with different underlying data type

types ① ~~[illegible]~~ Runtime polymorphism
② Compile-time polymorphism

It allow objects, methods or functions to operate differently based on the data they are working with while maintaining a common interface.

Inheritance

Allows a class like child or subclass to derive properties and behaviour from another class (Base class)

```
class Animal
 {
   void Eat()
   { SOP ("This animal eat food");
   }
class Dog extend Animal
 {  void Bark()
   {
    SOP ("The dog barks");
 }  }
```

```
public class Main {
   p s v main (String [] args)
   {
     Dog dog = new Dog();
      dog.eat();
      dog.bark();
    }
 }
```

Abstraction

focus on simplifying complex system by hiding unnecessary details and exposing only relevant features or functionalities.

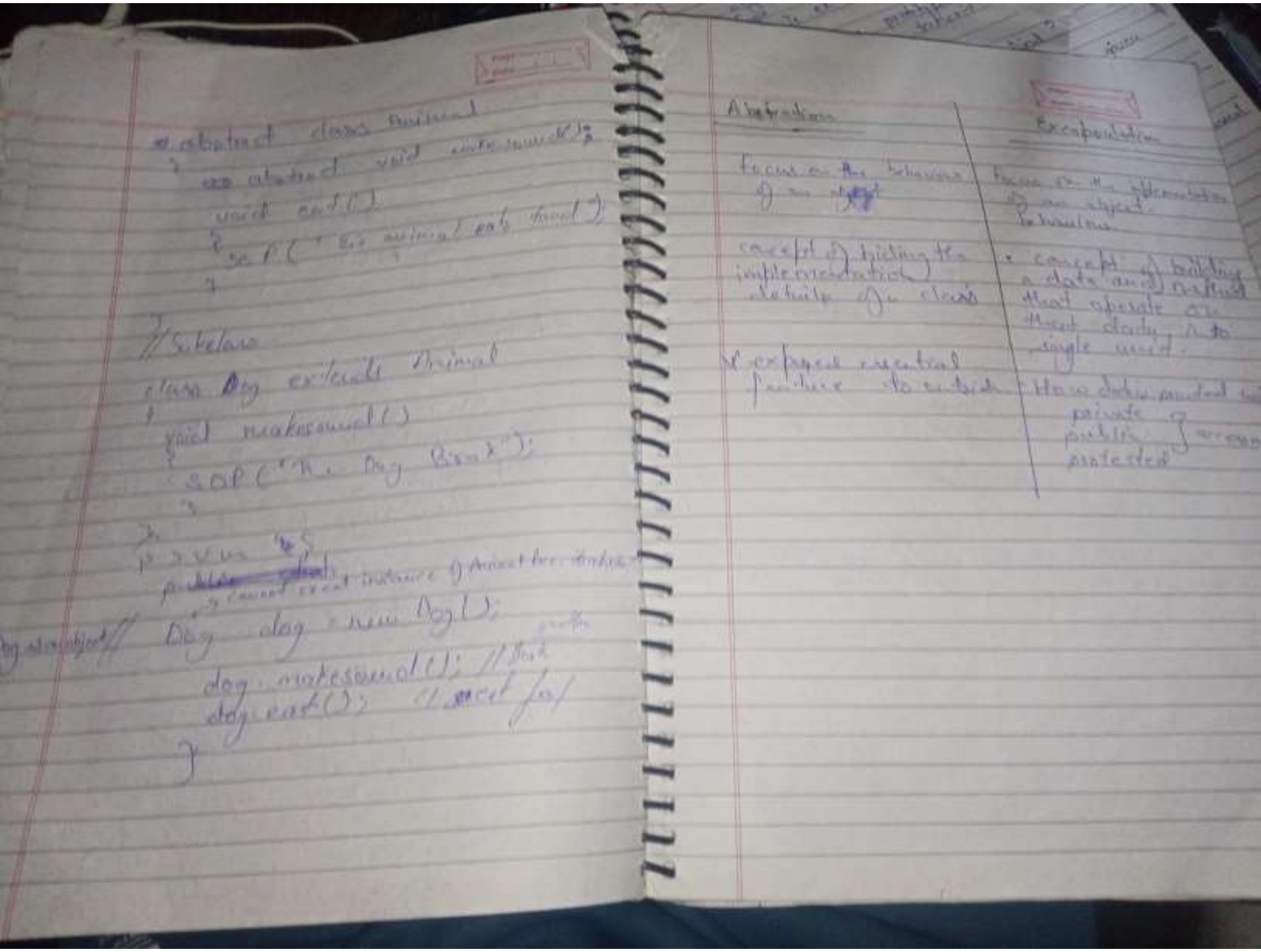
It also allows to define what an object does, without specifying how it does it.

Two way of Abstraction

(1) Abstract classes

- A class that cannot be instantiated directly
- It is meant to be extended by other class that will provide the specific implementation for certain method.

Allows us to define a basic structure that other classes can be build upon, sharing functionality while allowing flexibility for specific details



```
abstract class Animal
{
   abstract void makesound();
   void eat()
   {
    SOP(" the animal eats food");
   }
}
//Subclass
class Dog extends Animal
{
 void makesound()
 {
   SOP("The Dog Bark");
 }
}
p s v m (S
 ~~public class~~ cannot creat instance of Animal bec. abstract
Dog class object //  Dog  dog = new Dog();
   dog.makesound(); // Bark
   dog.eat();  // eat food
}
```

Abstraction	Encapsulation
Focus on the behaviour of an object	Focus on the implementation of an object. behaviour.
concept of hiding the implementation details of a class	• concept of building a data and method that operate on that data, into single unit.
* expose essential feature to outside	How data is protected using private public protected } access

constructor

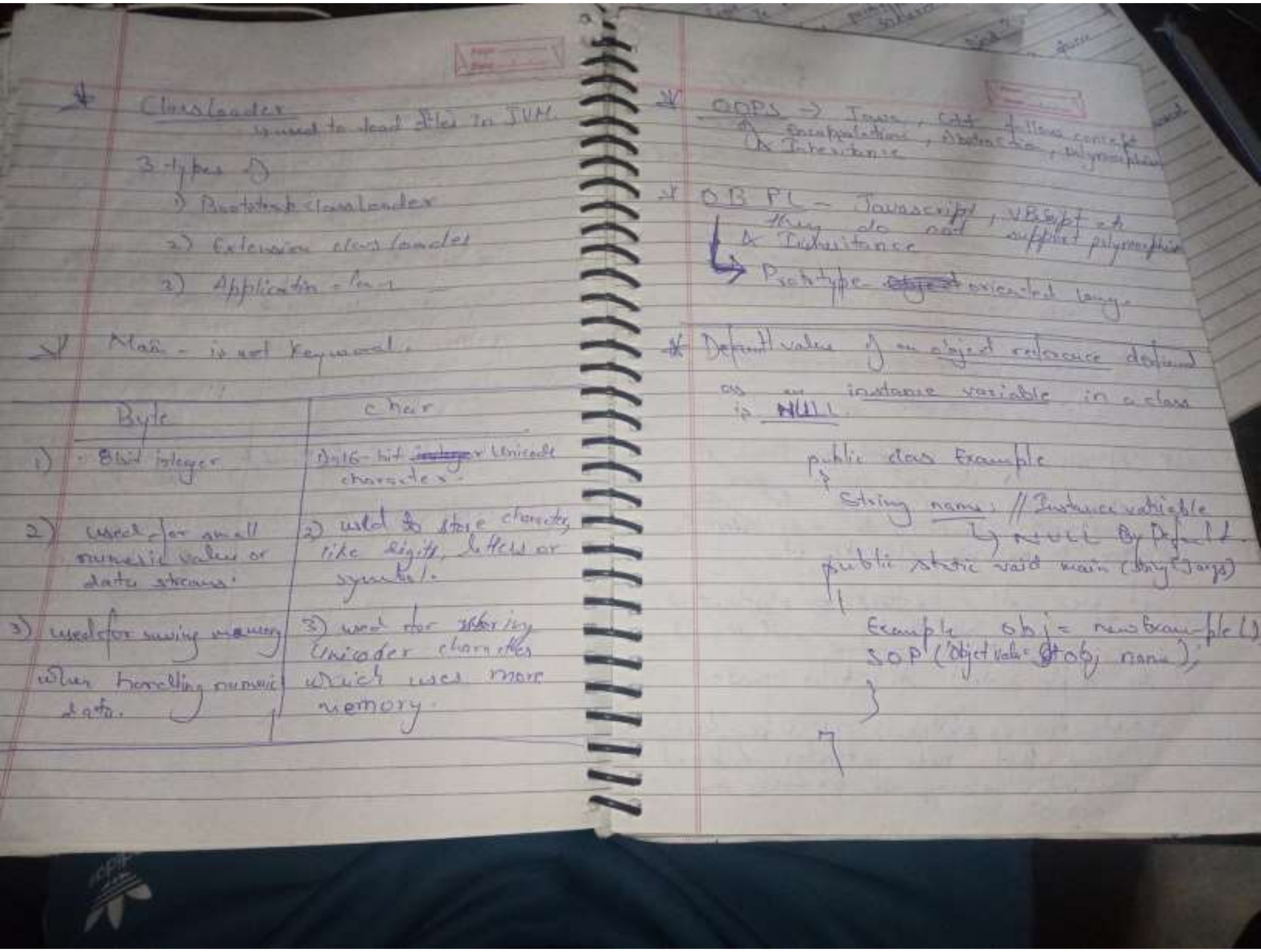
- is called when new object is created.
- constructor overloading is similar to function overloading.
- Different constructor can be created for single class.
- No return type, not even void.

creating multiple constructor in the same class with different parameters lists.

HashMap

- storing key-value pairs.
- Require hash function and uses hashcode and equal methods in order to put and retrieve elements to and from the collection.

HashMap(collection framework)	Hashtable (legacy frame
Allow one null key and multiple null value.	~~Allow~~ Do not a null or key va
16 Default capacity value.	11 (DCV)
Preferred in Single-threaded	Preferred in M threaded.
Uses → fail-fast iterator	Uses fail- enumerat



★ Classloader
is used to load files in JVM

3-types of

1) Bootstrap classloader

2) Extension class loader

3) Application class

★ Main - is not Keyword.

Byte	Char
1) 8bit integer	1) 16-bit ~~integer~~ Unicode character.
2) used for small numeric value or data streams.	2) used to store character like digits, letters or symbol.
3) used for saving memory when handling numeric data.	3) used for storing Unicode characters which uses more memory.

★ OOPS -> Java, C++ follow concept of Encapsulation, Abstraction, Polymorphism & Inheritance

★ OBPL - Javascript, VBScript etc
they do not support polymorphism & Inheritance

↳ Prototype ~~object~~ oriented lang.

★ Default value of an object reference defined as an instance variable in a class is NULL.

```
public class Example
{
  String name; // Instance variable
       ↳ NULL By Default.
 public static void main (String[] args)
 {
   Example obj = new Example();
   SOP ("Object value: " + obj. name);
 }
}
```

X No constructor

- constructor do not return value because it is not method
  ↓ Not even void

- Java Do Not support INHeritance of constructor.

- Why constructor cannot be final, Static or abstract in Java?

- If we set a method as final it mean we do not want any class to override it. But the constructor cannot be overridden. So thu is no use of making it.

If we set a method as abstract → It has No Body and should be implamented in child class-

Constructor as static → It belong to the class but not a particular object
Constructor alway use to initialise an object.

-this- Keyword is used to refer to current instance of the obje
-to differentiate b/w instance var
& local variable.

- Object class is the superclas every other class.

Abstraction

is the process of hiding certain implementation details of an object and showing only essential features of the object to outside world or user.

How Abstraction differ from Encapsulation?

1 Abstraction ~~is~~ happen at class level design while encap. happen at implementation level.

2 Abstraction focus - on defining the behaviour of object by hiding unnecessary details and exposing only the essential features.

) Encapsulation ~~happen at the~~ is the process of bundling data and method that operate on the data into a single unit.

Implemented → private, protected, public

---

* Can we instantiate an abstract class in java?

No, we cannot create an instance of an abstract class in Java.

* Interface ? (used to achive abstraction)

is a a bluepaint of class. It defines a contract for behaviour without specifying the implemendation details.

* Interface cannot be marked as final? because it condradicts the fundamental purpose and behaviour of interface.

* final keyword is used to prevent Inheritance

final class cannot be extended.
final class cannot be overridden

* Marker Interface is an interface that does not contain any method or fields.
Ex:- Serializable, cloneable, Remo

Abstract class	Interface
1) Can have both abstract & ~~[illegible]~~ concrete methods	1) Can only have abstract methods
2) A class can inherit only one abstract class	2) A class can implement multiple interface.
3) Instance variable [illegible] for	3) Instance variable

* Integer class is wrapper for int.

Integer class is marked as final because Integer class are Immutable, means once the integer class is created its value cannot be changed. Making class as final Ensures that no subclass can alter this behviour.

* Serialization
is the process of converting an object state to byte stream
[illegible] represent internal state of object.

* two method of garbage collection?
System.gc() & Runtime.gc()

* finalize()
method is used to
perform any cleanup be
Garbage collection.
This is in the object cla

112·

Nested Class

class that is defined inside another class.

Types

1) Static Nested Class

- static keyword
- No instance variable/method

```
class outer
{
 static class staticNested {
 {
   void display()
     S.O.P ("Aditya");
   }
  }
 }
```

2) Non static class (Inner class)

is associated with a instance of enclosing class.

Types INNER

1) ~~INNER class~~ member inner class
2) Anonymous inner class
3) Local inner class

* two ways to create a string object
    1) String literals
    2) New operator

String	StringBuffer
Immutable	Mutable

147

Sorting Techniques

1) Bubble Sort

~~worst~~ case - $O(n^2)$
Best case - $O(n)$.

~~Reapeate~~:
Reapeatedly compares adjecents elements and swaps them if they are in the wrong order.

The largest element Bubble up to the end in each pass.

program

```
void bubble sort (int arr[])
{
 int n= arr. length;
for (i=0 ; i<n-1; i++)
 {
  for (j = 0; j<n-i-1 ;j++)
  {
   if (arr [j] > arr [j+1])
   {
      int temp = arr [j] ;
        arr [j] = arr [j+1];
        arr [j+1] = temp;
   }
  }
 }
}
```

Selection Sort

~~Reapeatedly steps through the list compares adjects element and swap if wrong.~~

Key points

Reapeatedly selecting the smallest elements ~~part element~~ from the unsorted part of the array and move at the correct position

1) Divide the array into 2 part

(Empty) Sorted

unsorted (Entire array)

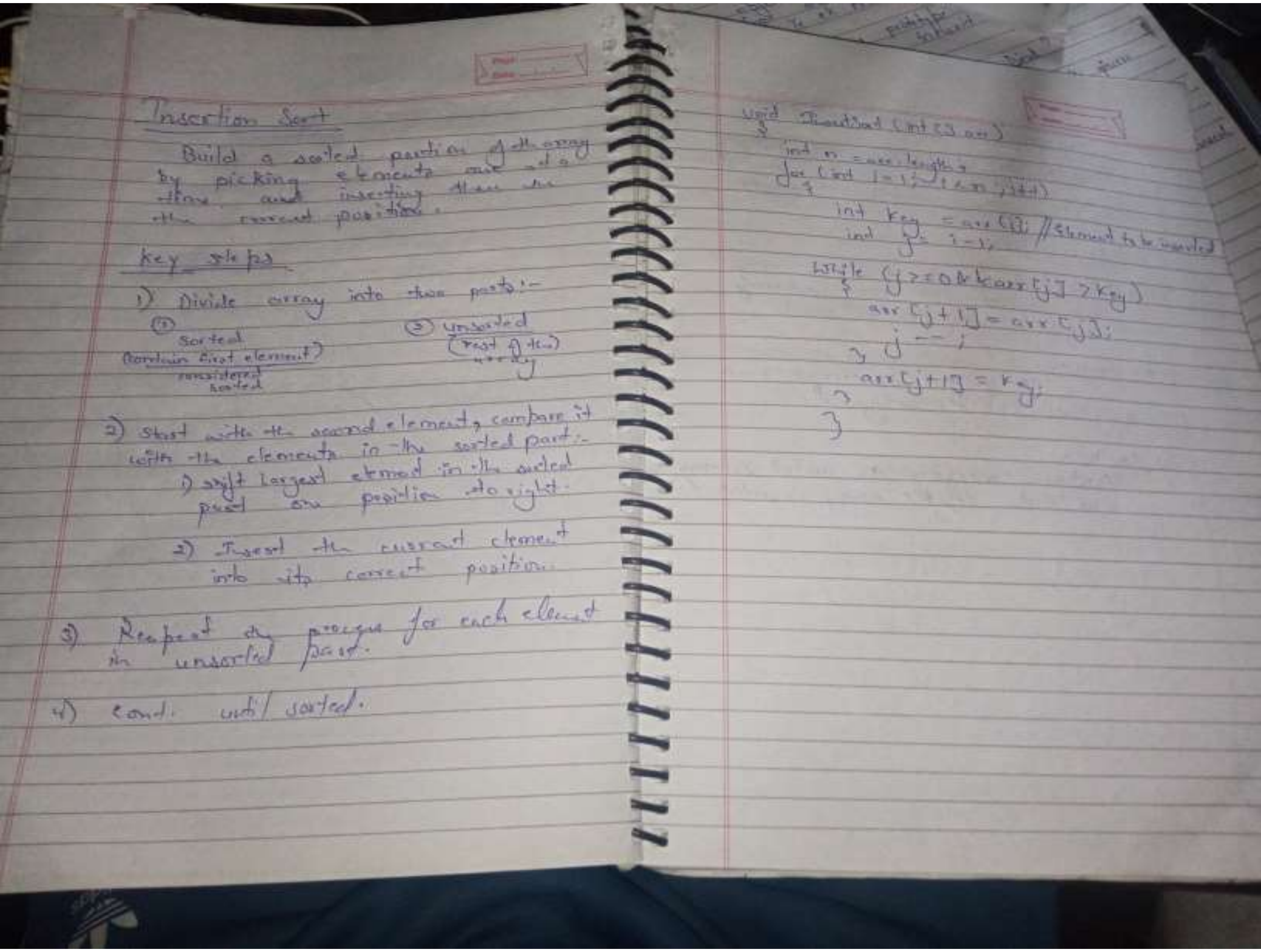
1) Find ~~the~~ smallest element in the unsorted part.

2) Swap the smallest ~~part~~ elem

3) Move the boundary of the sorted part one element to the right

4) Repeat until sorted.

```
void Selection sort (int [] arr)
{
   int n = arr.length;
  for (i=0 ; i<n-1 ; i++)
       int minIndex = i;
  for (j= i+1 ; j<n ; j++)
  {
      if (arr[j] < arr[minIndex])
      {
            minIndex = j;
       }
   }
   int temp = arr [minIndex];
   arr [minIndex] = arr [i];
      arr [i] = temp;
}
```



Insertion Sort

Build a sorted portion of the array by picking elements one at a time and inserting them in the correct position.

Key steps

1) Divide array into two parts:-

(1) sorted (contain first element) considered sorted

(2) unsorted (rest of the array)

2) Start with the second element, compare it with the elements in the sorted part:-

1) shift largest element in the sorted part one position to right.

2) Insert the current element into its correct position.

3) Repeat the process for each element in unsorted part.

4) cont. until sorted.

```
void InsertSort (int [] arr)
{
   int n = arr.length;
   for (int i = 1; i < n; i++)
   {
      int key = arr[i]; //Element to be inserted
      int j = i-1;
    while (j >= 0 && arr[j] > key)
    {
       arr [j+1] = arr [j];
       j--;
    }
     arr[j+1] = key;
   }
 }
```

Merge Sort (Divide - Conquer)

Divide the array into halves, sorts them recursively, and then merges the sorted halves

Keys steps

1) Divide Recursively divide the array into halves until each subarray contains a single elements.

2) Conquer
Merge the subarrays back together while sorting them.

3) Combine
Continue merging the sorted subarray until the entire array is sorted.

```
- Function to sort array
Mergesort (int[] arr, int left, int right)
{
  if (left < right)
  {
     int mid = left + (right - left) / 2;
   mergeSort (arr, left, mid);
   mergeSort (arr, mid+1, right);
   merge (arr, left, mid, right);
- Function to ~~merge~~ Combine two sorted halves

public static void merge(int[] arr, int left, int mid,
                                  int right)
   {
      int n1 = mid - left + 1; // find the size of
      int n2 = right - mid;    //   two subarrays

  // create temporary array
   int [] leftArr = new int [n1];
   int [] RightArr = new int [n2];
  // copy data to the temporary arrays
 for (int i = 0; i < n1; i++)
   {
       leftArr [i] = arr [left + i];
   }
 for (int j = 0; j < n2; j++)
   {
     RightArr [j] = arr [mid + 1 + j];
   }
```

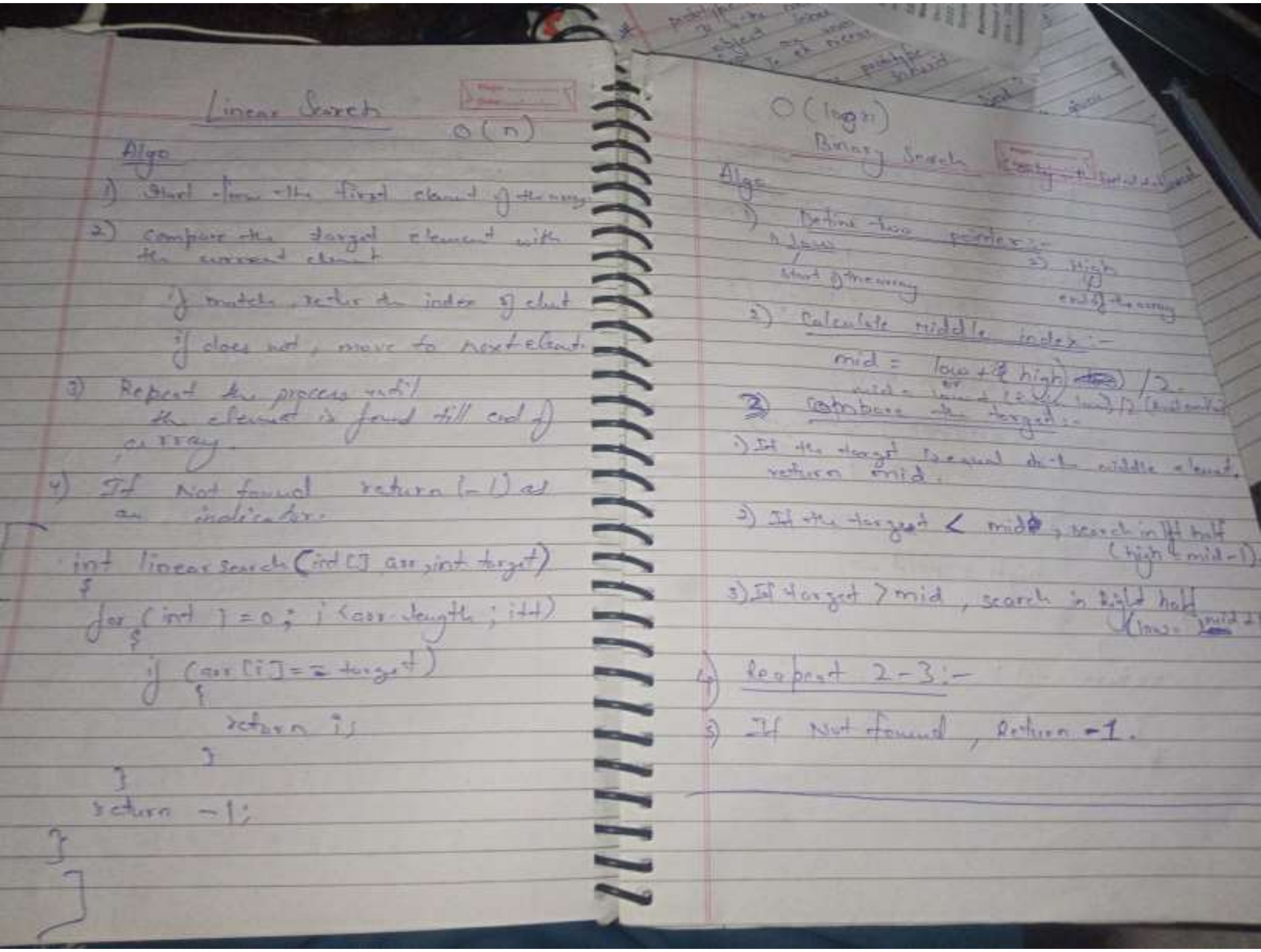
O(n log n)

```
// Merge the temporary arrays back into the
original array

while (i < n1 && j < n2)
{
    if (leftArr[i] <= rightArr[j])
    {
        arr[k] = leftArr[i];
        i++;
    }
    else {
        arr[k] = rightArr[j];
        j++;
    }
    k++;
}
while (j < n1)
{
    arr[k] = leftArr[i];
    i++;
    k++;
}
while (j < n2)
{
    arr[k] = rightArr[j];
    j++;
    k++;
}
}
```

Built-in Sort

```
Arrays.sort(arr);
SOP(Arrays.toString(arr));
```



Linear Search

O(n)

Algo

1) Start from the first element of the array.

2) compare the target element with the current element

   if match, return the index of element

   if does not, move to next element.

3) Repeat the process until the element is found till end of array.

4) If Not found return (-1) as an indicator.

```
int linearsearch(int[] arr, int target)
{
 for (int i = 0; i < arr.length ; i++)
  {
    if (arr[i] == target)
     {
        return i;
     }
  }
 return -1;
}
```

O(log n)

Binary Search

Algo

1) Define two pointer:-

   1) low — start of the array
   2) high — end of the array

2) Calculate middle index:-

   mid = (low + high) / 2.

   or mid = low + (high - low) / 2 (avoid overflow)

3) Compare the target:-

   1) If the target is equal to the middle element, return mid.

   2) If the target < mid, search in left half (high = mid - 1).

   3) If target > mid, search in right half (low = mid + 1)

4) Repeat 2-3:-

5) If Not found, Return -1.

```
int BinarySearch (int [] arr, int target)
{
    int low = 0; ~~high = arr.~~ // start
    int high = arr. length -1 ; // End form
while ( low < = high )
    {
      int mid = (low + high ) /2 ;
    if ( arr [mid] = = target )
      {
         return mid;
      }
else if (arr [mid ] < target)
      {
        low = mid + 1;
      }
      else
      {
        high = mid -1;
      }
    }
return -1;
}
```