Fundamentals of Algorithm (FA)

Omkar Basnet
Academic Incharge , CSIT/BCA
Texas international college

Revision of chapter 2:

- **Calculate the running time of following algorithms:**
  - Evaluating the given number prime or not
  - Calculating sum of n numbers.
  - Addition of 2*2 matrix using array
  - Evaluating given number even or odd

Evaluating given number even or odd

```
Print (Enter number) // constant time
(unit time)== 1
Scan("%d",n) // unit time ==1
If(n%10==0)  // constant time  c
      {
       print ("even") //1
      }
      Else {
      Print("odd")  // 1
      }
```

Total time is 1+1+C+(either if i.e. 1 or else 1) **choose larger runtime**.

But the complexity of both subroutines are same so

T.C.= 1+1+C+1

O(1)

Chapter three(8 teaching hours)

- Analysis of **Brute-Force Algorithm**

  (Running time analysis)
- Analysis of **Brute-Force maxima Algorithm**
- Introduction of **Plane Sweep Algorithm**
- Comparison **between plane sweep and Brute-Force Algorithm**

Analysis of Brute-Force Algorithm

- **Brute-Force Algorithm**
  - It is straight forward approach to solve a problem , usually based on the problem statement and definition of the concept involved.
  - Applicable to a very wide variety of problems such as sorting , searching , matrix multiplication , string matching etc.
  - The analysis of brute-force algorithm can be done in 3-sum technique to find out the unique pair of sets

Brute-Force Algorithm

- **3-sum technique to find out the unique pair of sets**

```
for(i=0; i<n; i++)
        {
                for(j=0; j<n; j++)
                        {
                        for(k=0; k<n; k++)
                                {
                                If (a[i],a[j],a[k]== 0)
                                        {
                                        count ++
                                        }
                                return (a[i],a[j],a[k]);
                                }
                        }
        }
```

- The total complexity of 3-sum brute-force algorithm is $O(n^3)$
- Space complexity is :
- They need to store given sets and the matching status along with number of matching sets so
- Space complexity is $O(n^2)$.

Brute-Force Algorithm

- How does it works ???

let n = 5.

i.e. set a = { 1, 5, 3, 2, 4 }

b = { 3, 2, 7, 9, 8 }

c = { 3, 4, 2, 8, 11 }

According to 3-sum Algorithm a[i] = a[0] i.e. 1 is compared with all the members of set b & c.

Here, element 3 & 2 are in all sets a, b, c. So the value of count is 2. because there are two same elements in the given sets.

The single number is compared with all the possible elements ~~the~~ so it is called Brute-force Algorithm.

1 is compared in following way

(1, 3, 3), (1, 3, 4), (1, 3, 2)

(1, 3, 8), (1, 3, 11) ....

(1, 2, 3), (1, 2, 4), (1, 2, 2), (1, 2, 8)

....

(1, 8, 11).

Brute-Force Algorithm

- NOW COMPLETE THIS CLASSWORK
  - Consider the value of n=3 and trace the Brute-force algorithm.

Brute-Force maxima Algorithm

```
MAXIMA(int n, print (1…… n)) // maximum of p(0….. n-1)
        {
        For i=1 to n
        {
        maximal =TRUE// p[i] is maximum by default
        For j = 1 to n
        {
If(I != j) AND (P[i].X ≤ p[j].X) AND (P[i].Y ≤ p[j].Y)
        {
        maximal =FALSE // p[i] is dominated by p[j]
        Break;
}}
If (maximal) output p[i] // no one dominate p[i]
}}
```

- In this algorithm we will analyze the 2D maxima algorithm.
- The main aim of algorithm is to find out the maximum set/number of first loop i.e.

P[i].X ≥p[j].X Or P[i].Y ≥p[j].Y

- Then only produce the output.

Brute-Force maxima Algorithm

```
MAXIMA(int n, print (1…… n)) // maximum of p(0….. n-1)
        {
        For i=1 to n
        {
        maximal =TRUE// p[i] is maximum by default
        For j = 1 to n
        {
If(I != j) AND (P[i].X ≤  p[j].X) AND (P[i].Y ≤ p[j].Y)
        {
        maximal =FALSE // p[i] is dominated by p[j]
        Break;
}}
If (maximal) output p[i] // no one dominate p[i]
}}
```

Brute-Force maxima Algorithm

Given 3 points

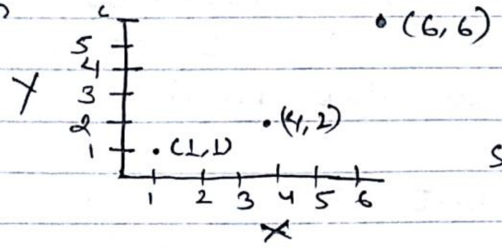
P((1,1), (4,2), (6,6))

Now, n=3. So.

for i=1 to n(3)

if we plot the given number in graph

then



point (6,6) is the maximum set.

Now findout by using Brute force maxima.

for i=1 to 3, i=1

P[i].x = P[1].x = 1 & P[i].Y = P[1].Y = 1

maximal = (1,1)

j=1 if (i!=j) i.e. (1!=1) ⇒ False

loop is not executed

j=2 if (1!=2) && (1≤4) && (1≤2)

maximal = FALSE

Break;

i=2, P[2].X=4 P[2].Y = 2, maximal = (4,2)

j=1 if (2!=1) && (4≤1) && (2≤1)

⇒ FALSE

J=2 if (2!=2) ⇒ False

Brute-Force maxima Algorithm

MAXIMA(int n, print (1…… n)) // maximum of p(0….. n-1)

{

For i=1 to n

{

maximal =TRUE// p[i] is maximum by default

For j = 1 to n

{

If(I != j) AND (P[i].X ≤ p[j].X) AND (P[i].Y ≤ p[j].Y)

{

maximal =FALSE // p[i] is dominated by p[j]

Break;

}}

If (maximal) output p[i] // no one dominate p[i]

}}

J = 3 (2 != 3) && (4 ≤ 6) && (2 ≤ 6)

maximal = False

Break;

i = 3 P[3].X = 6 , P[3].Y = 6

maximal = (6, 6).

j = 1 If (3 != 1) && (6 ≤ 1) && (6 ≤ 1) ⇒ False

j = 2

If (3 != 2) && (6 ≤ 4) && (6 ≤ 2) ⇒ False

j = 3

If (3 != 3) ⇒ False.

}

loop Ends. (Both for loop Ends i & j)

Now

Print latest maximal i.e. (6, 6)

output = (6, 6) #

We know the highest/ maximum number is (6, 6) from the given set P((1,1), (4,2), (6,6)).

#

Plane Sweep Algorithm

- **Working Mechanism**
  - Sweep the vertical line across the plane from left to right.
  - After sweeping from left to right , we will find out the maximum point sets.
  - For simplicity we will assume no Y- coordinates are same.

Plane Sweep Algorithm

- **Algorithm**

Plane-sweep –maxima(n, p[1,…….., n])

Sort the p in increasing order of x.

Initialize stack s.

For i=1 to n

DO

While (S.notempty() && s.top.Y< p[i].Y)

DO S.POP()

Do S.PUSH(p[i]

Output The content of STACK

- The total complexity of plane sweep algorithm is O(nlogn)

Plane Sweep Algorithm

- **Algorithm**

Plane-sweep –maxima(n, p[1,…….., n])

Sort the p in increasing order of x.

Initialize stack s.

For i=1 to n

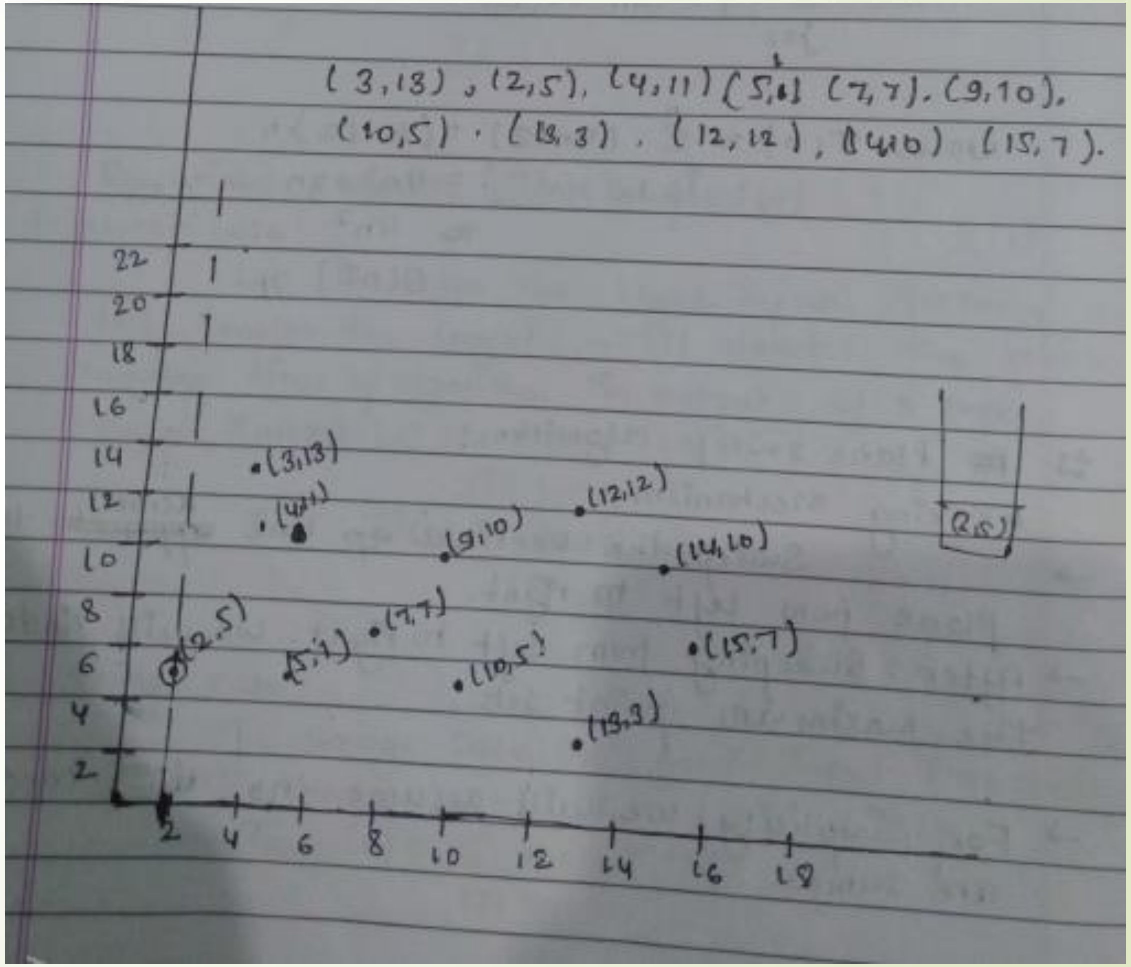
DO

While (S.notempty() && s.top.Y< p[i].Y)

DO S.POP()

Do S.PUSH(p[i])

Output The content of STACK



Plane Sweep Algorithm

- **Algorithm**

Plane-sweep –maxima(n, p[1,…….., n])

Sort the p in increasing order of x.

Initialize stack s.

For i=1 to n

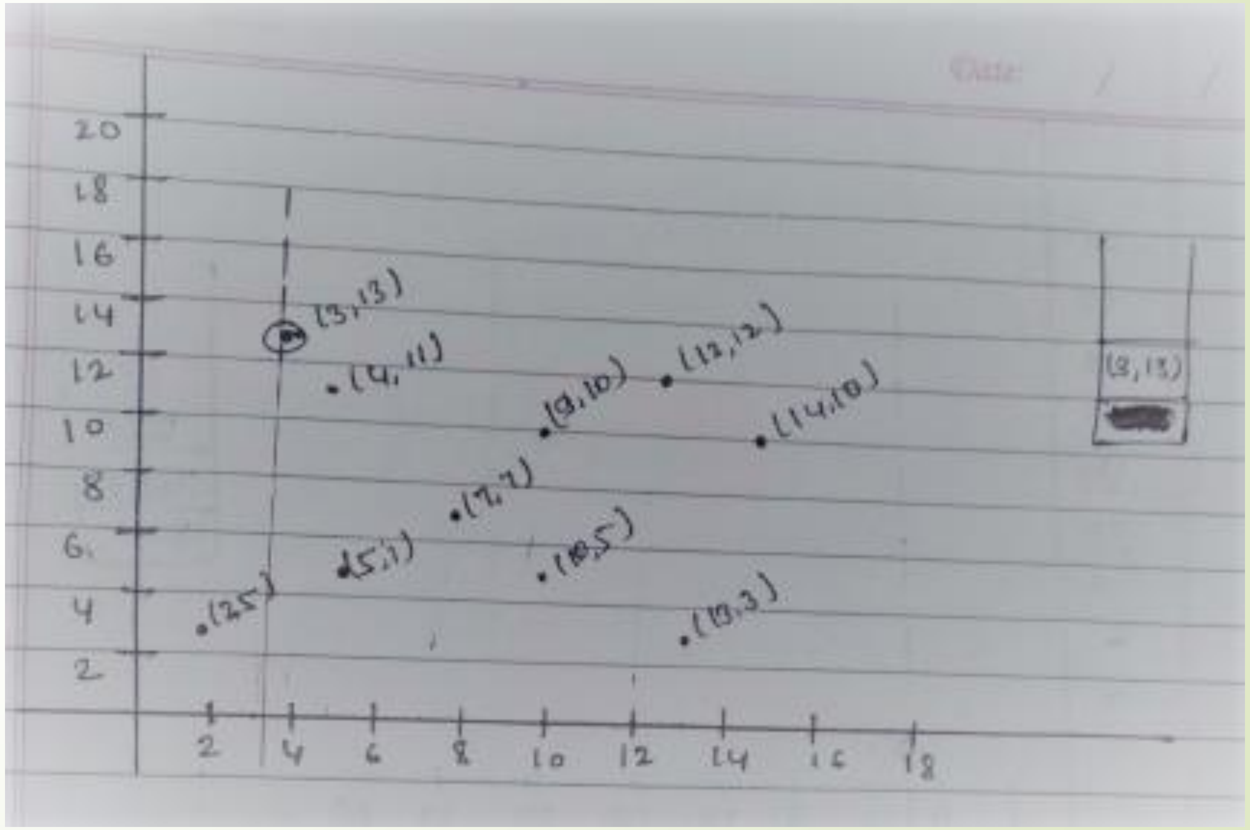
DO

While (S.notempty() && s.top.Y< p[i].Y)

DO S.POP()

Do S.PUSH(p[i])

Output The content of STACK



Plane Sweep Algorithm

- **Algorithm**

Plane-sweep –maxima(n, p[1,…….., n])

Sort the p in increasing order of x.

Initialize stack s.

For i=1 to n

DO

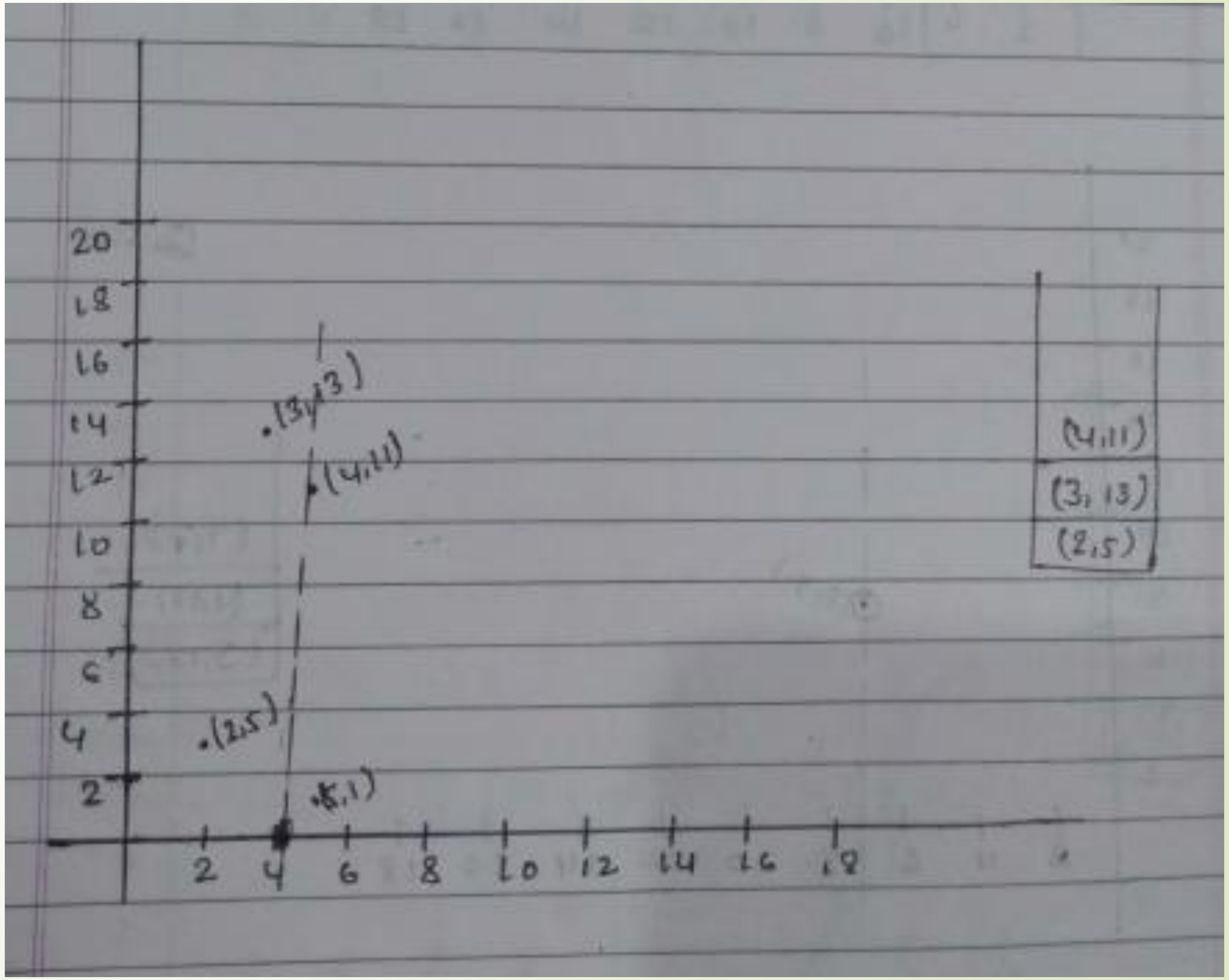
While (S.notempty() && s.top.Y< p[i].Y)

DO S.POP()

Do S.PUSH(p[i])

Output The content of STACK

- Note : (2,5) is not the stack element



Plane Sweep Algorithm

- **Algorithm**

Plane-sweep –maxima(n, p[1,…….., n])

Sort the p in increasing order of x.

Initialize stack s.

For i=1 to n

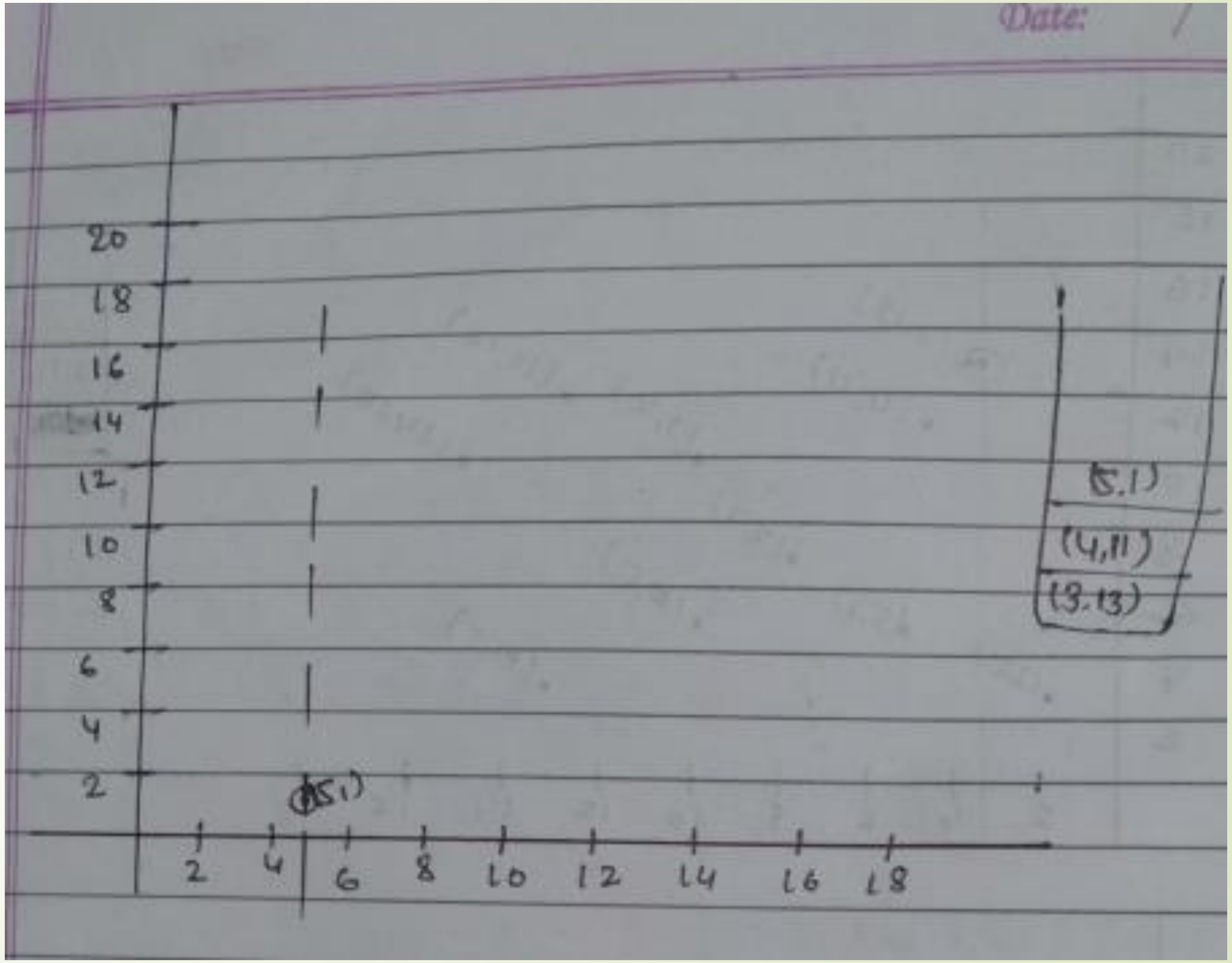
DO

While (S.notempty() && s.top.Y< p[i].Y)

DO S.POP()

Do S.PUSH(p[i])

Output The content of STACK



Plane Sweep Algorithm

- **Algorithm**

Plane-sweep –maxima(n, p[1,…….., n])

Sort the p in increasing order of x.

Initialize stack s.

For i=1 to n

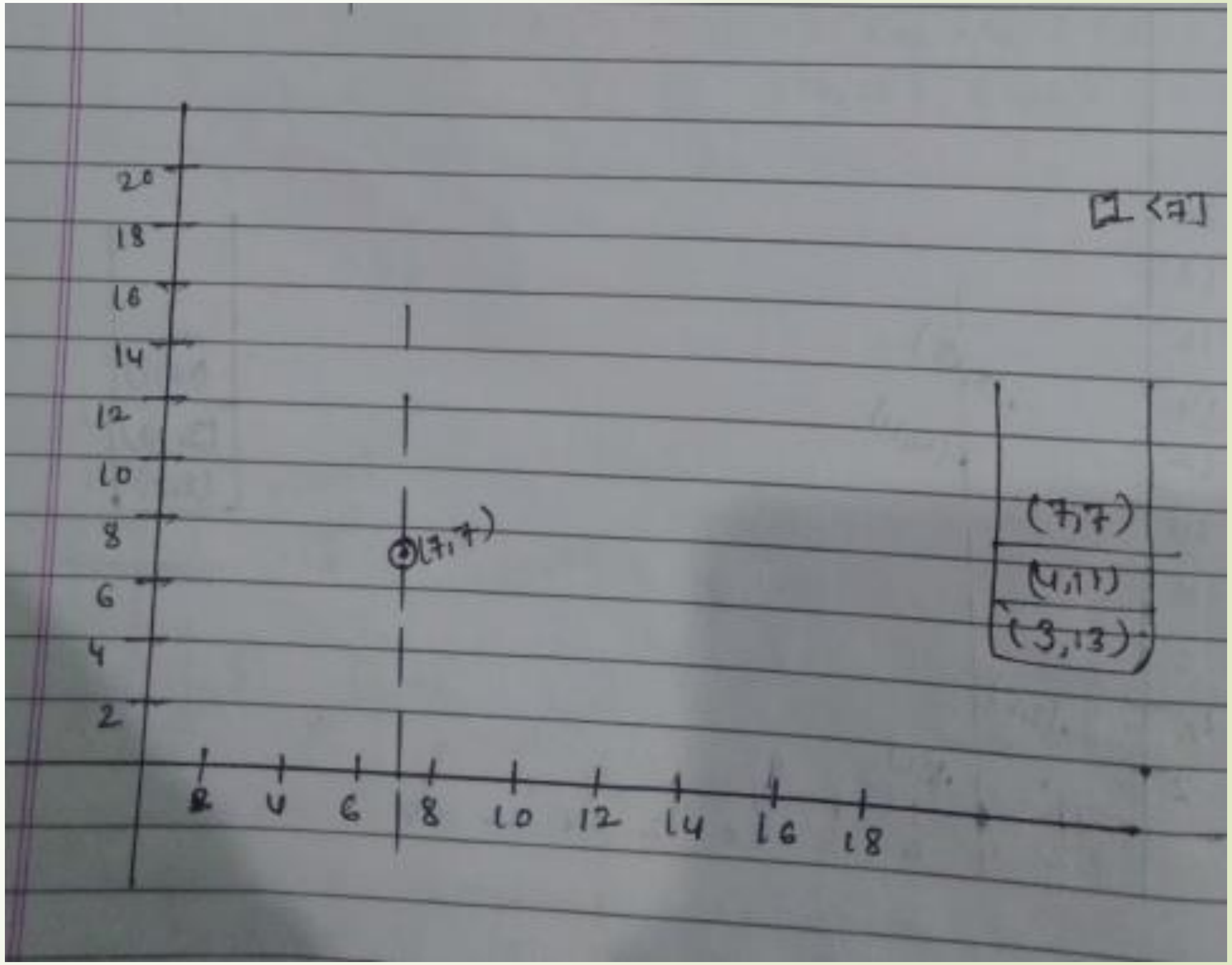
DO

While (S.notempty() && s.top.Y< p[i].Y)

DO S.POP()

Do S.PUSH(p[i])

Output The content of STACK



Plane Sweep Algorithm

- **Algorithm**

Plane-sweep –maxima(n, p[1,…….., n])

Sort the p in increasing order of x.

Initialize stack s.

For i=1 to n

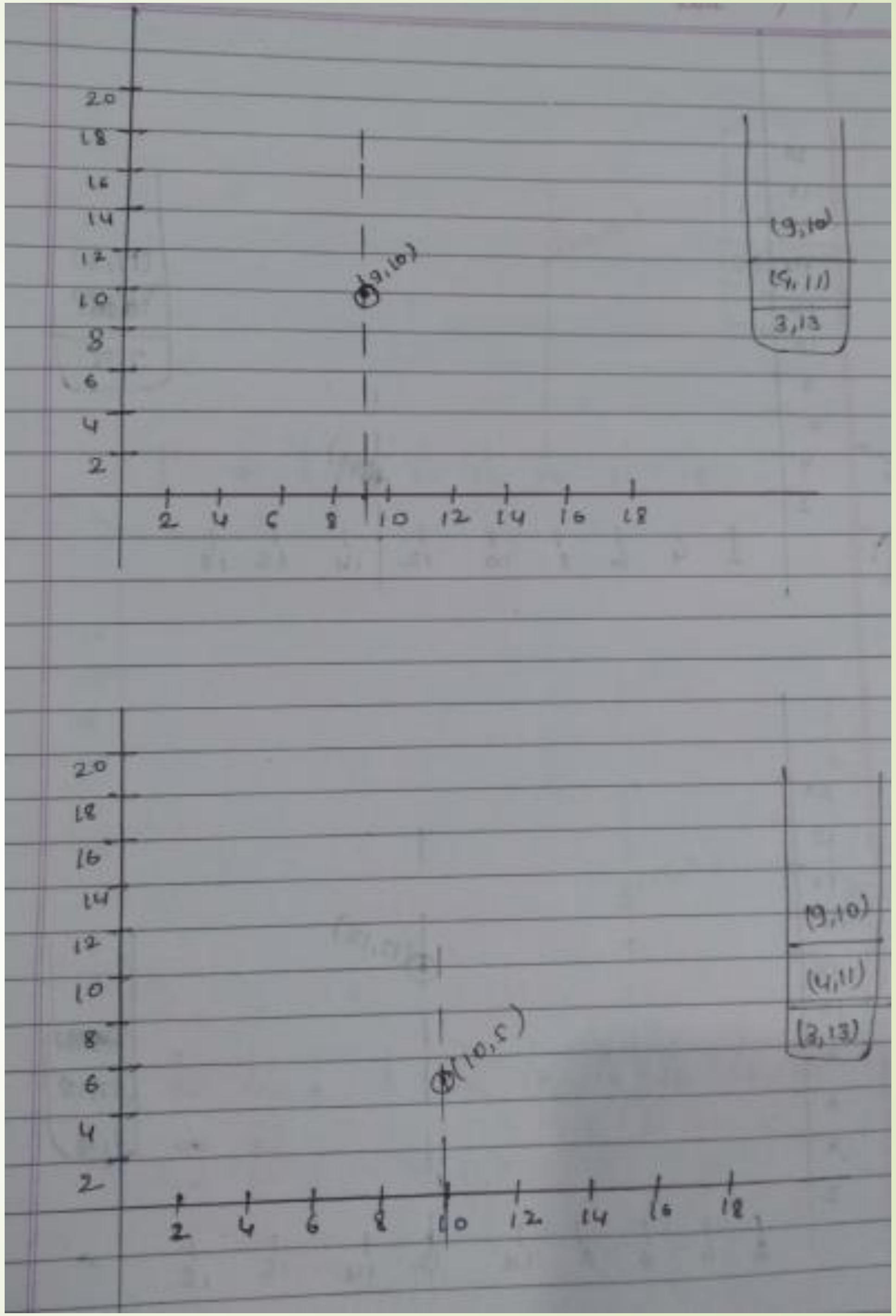
DO

While (S.notempty() && s.top.Y< p[i].Y)

DO S.POP()

Do S.PUSH(p[i])

Output The content of STACK



Plane Sweep Algorithm

- **Algorithm**

Plane-sweep –maxima(n, p[1,…….., n])

Sort the p in increasing order of x.

Initialize stack s.

For i=1 to n

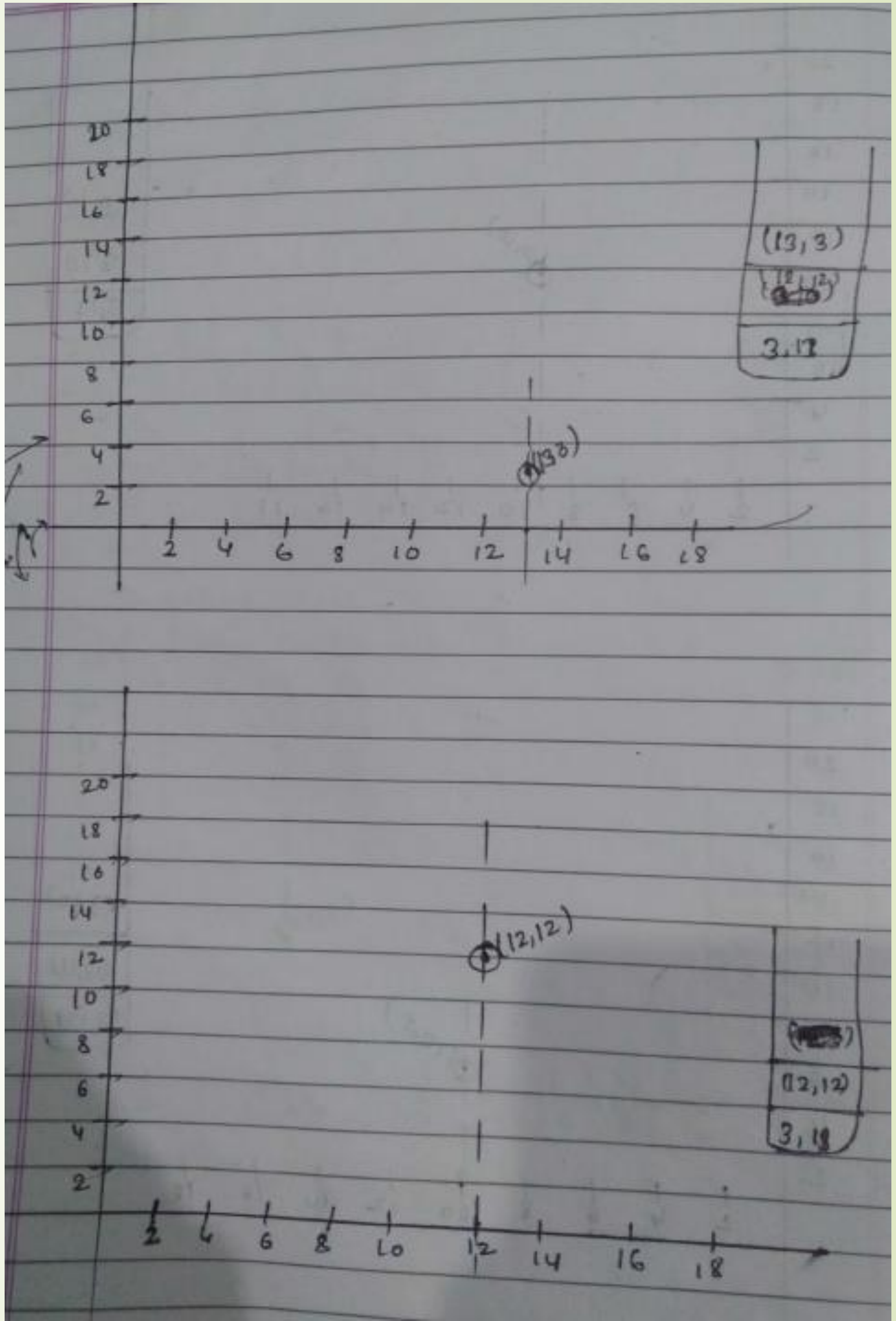
DO

While (S.notempty() && s.top.Y< p[i].Y)

DO S.POP()

Do S.PUSH(p[i])

Output The content of STACK



Plane Sweep Algorithm

- **Algorithm**

Plane-sweep –maxima(n, p[1,…….., n])

Sort the p in increasing order of x.

Initialize stack s.

For i=1 to n

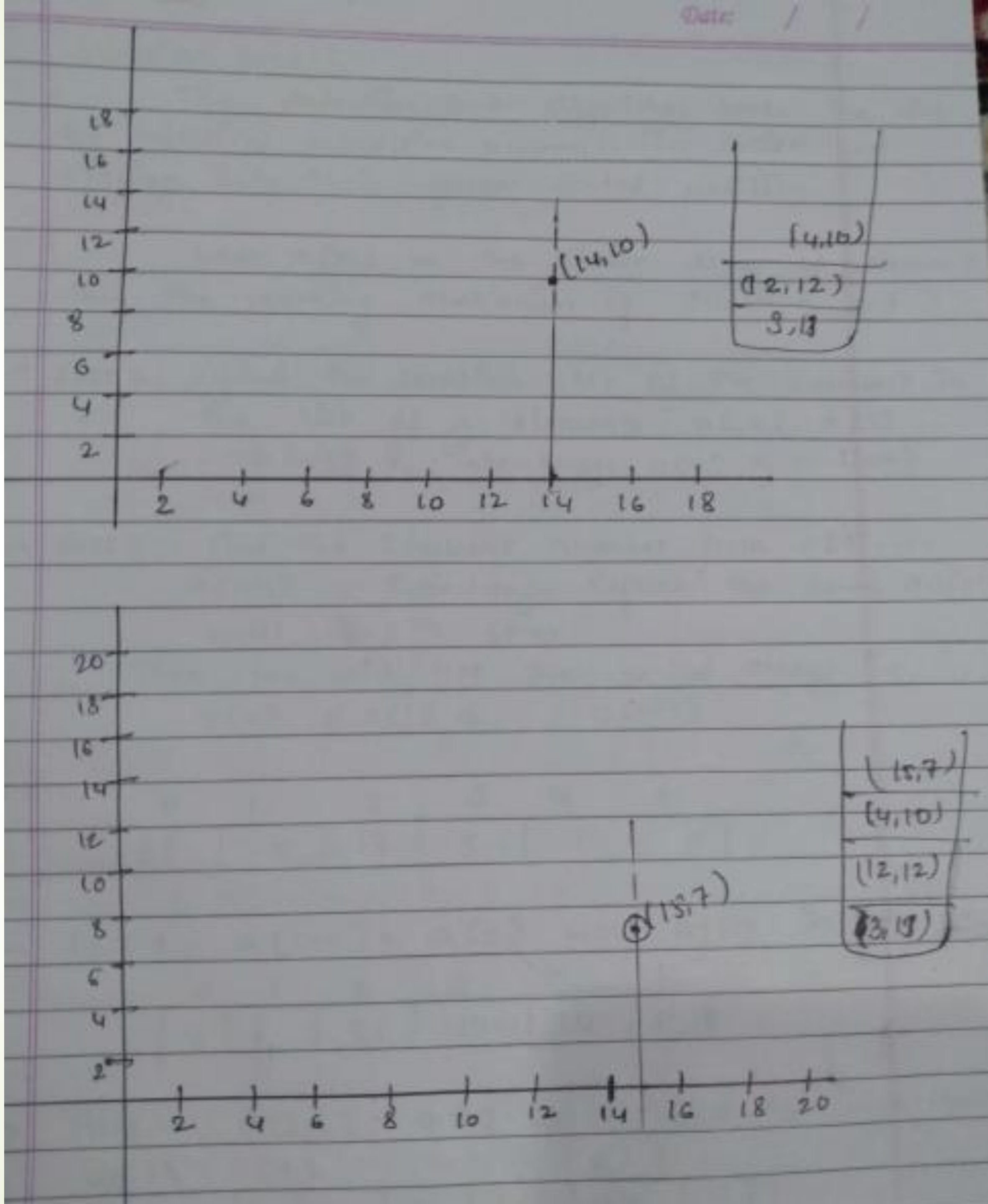
DO

While (S.notempty() && s.top.Y< p[i].Y)

DO S.POP()

Do S.PUSH(p[i])

Output The content of STACK



LETS SEE YOU ON NEXT CLASS