```
DividePointArray(P,l,r)
 if l < r
    then m ← ⌊(l+r)/2⌋
         DividePointArray(P,m+1,r)
         DividePointArray(P,l,m)
         FindMaximalPointArray(P,l,r)
```

```
create stack PointStack
FindMaximalPointArray(P,l,r)
 for i ← r downto l
    do if PointStack empty
          then PointStack ← P[i]
       else
          p1 ← Y coordinates of top element in PointStack
          p2 ← Y coordinates P[i]
          if p1 < p2
             then PointStack ← P[i]
```

<Implement Algorithm>

test data의 값을 Point class를 만들어서 저장하고, Point type의 Array를 생성 Array를 dividePointArray 메소드로 1/2로 recursion을 사용하여 나눈다.

findMaximalPointArray 메소드에서 array에 있는 Point객체의 y좌표와 Stack top에 있는 Point의 y좌표와 비교하여 Stack에 있는 것보다 Array에 있는 값이 더 크면 Stack에 저장한다.

<Time Complexity>

위에서 구현한 알고리즘은 divide-and-conquer 방법을 이용하였다.

data 크기 n에 대하여 동작하는 시간을 $T(n)$이라 하자.

$n=1$인 경우 상수시간이 걸리므로 $\Theta(1)$이다.

Array를 1/2로 나누어서 문제를 해결하므로 2개의 subproblem이 생기고 subproblem의 동작시간은 $T(1/2)$이다.

$D(n)$을 data 크기 n에 대하여 나누는 시간이라 하면 나누는 과정에서는 상수시간이 걸린다. $C(n)$ subproblem을 합치는데 걸리는 시간이므로 n시간 즉 $\Theta(n)$시간 걸린다.

$$T(n)=\begin{cases}\Theta(1) & (n=1)\\ 2T(n/2)+\Theta(n) & (n>1)\end{cases}$$

Master Theorem을 이용하여 $T(n)$을 구하자.

$f(n) = \Theta(n^{\log_b a})$에서 위 알고리즘에서 $a=b=2$이므로 $f(n)=\Theta(n)$

따라서 $T(n) = \Theta(n^{\log_b a}\lg n) = \Theta(nlogn)$임을 알 수 있다.