LAB ASSESSMENT 4

Name: Karan Sehgal

Registration Number: 22BCE3939

Course Name: Design and Analysis of Algorithms

Course Code: BCSE204P

Submitted to: Prof. Gayatri P

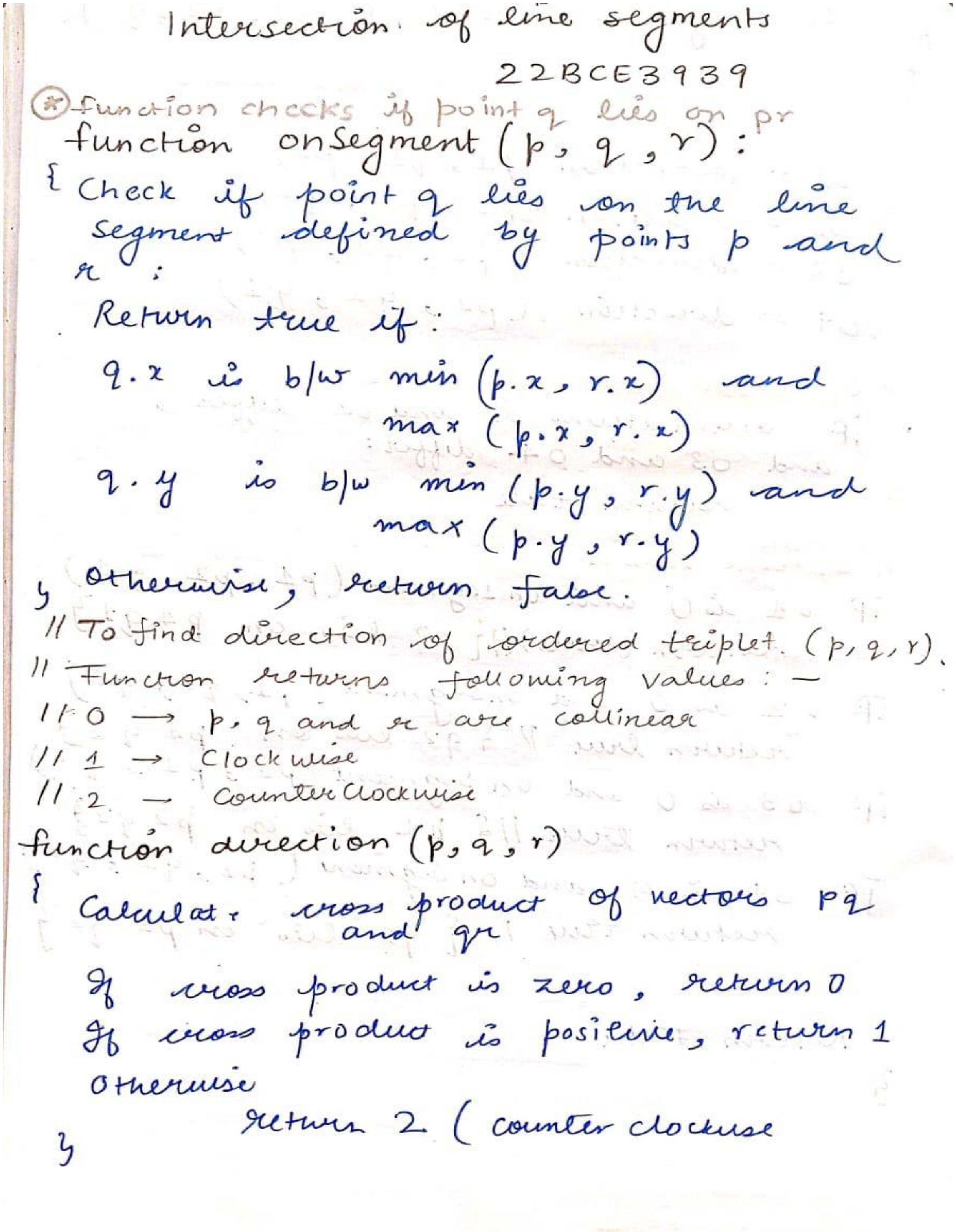
Question1:

Implementation of Intersection of line segments:

Problem Statement:

Given a pair of line segments with start points (p1 and p2) and end points (q1 and q2) determine if they will intersect or not.

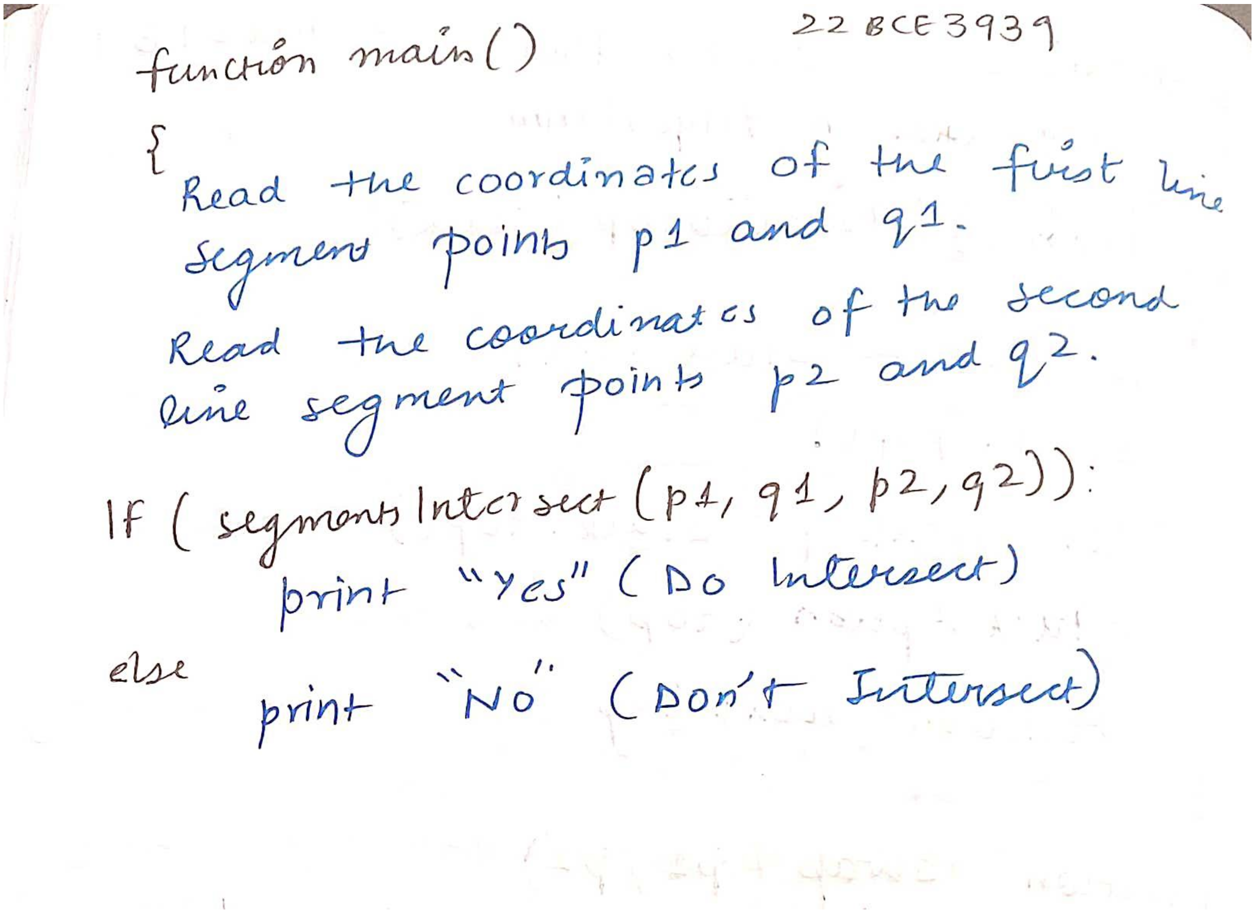
Pseudocode:



```
Intersection of line segments
                    22BCE3939
(*)-function checks if point q lies on pr
function onSegment (p, q, r):
{ Check if point q lies on the line
  segment defined by points p and
  r :

  Return true if:
    q.x is b/w min (p.x, r.x) and
                 max (p.x, r.x)
    q.y is b/w min (p.y, r.y) and
                 max (p.y, r.y)
} Otherwise, return false.
// To find direction of ordered triplet (p, q, r).
// Function returns following values: —
// 0 → p, q and r are collinear
// 1 → Clockwise
// 2 — Counter Clockwise
function direction (p, q, r)
{
  Calculate cross product of vectors pq
                 and qr

  If cross product is zero, return 0
  If cross product is positive, return 1
  Otherwise
         return 2 ( counter clockwise
}
```

22BCE3937

```
function segments Intersect(p1, q1, p2, q2)
{
    // find the four orientations needed
    for general and special cases
      o1 = direction (p1, q1, p2)
      o2 = direction (p1, q1, q2)
      o3 = direction (p2, q2, p1)
     .o4 = direction (p2, q2, q1)
    // General case.
      if orientations o1 and o2 differ,
       and o3 and o4 differ:
           return true
    // special cases
    if o1 is 0 and on segment (p1, p2, q1)
         return (true) //{p2 lies on p1q1}
    if o2 is 0 and onsegment (p1, q2, q1)
         return true // {q2 lies on p1 q1}
    if o3 is 0 and on Segment (p2, p1, q2)
         return true //{p1 lies on p2q2}
    if o4 is 0 and on Segment (p2, q1, q2)
         return true // {q1 lies on p2 q2}
    Otherwise
     Return False
}
```



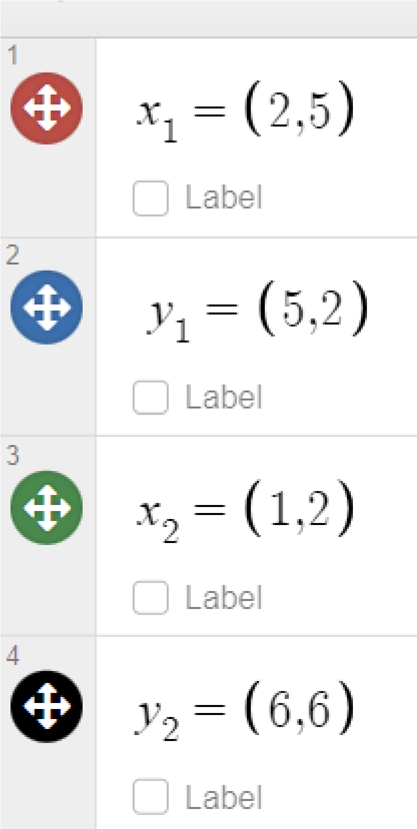
22BCE3939

```
function main()
{
  Read the coordinates of the first line
  segment points p1 and q1.
  Read the coordinates of the second
  line segment points p2 and q2.
If ( segmentsIntersect (p1, q1, p2, q2)):
      print "yes" ( Do Intersect)
else
      print "No" ( Don't Intersect)
```

Source Code:

```
#include <iostream>
using namespace std;

struct Point {
    int x;
    int y;
};

bool onSegment(Point p, Point q, Point r) {
    return (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y));
}

int direction(Point p, Point q, Point r) {
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y
- q.y);
    if (val == 0) return 0;
    return (val > 0) ? 1 : 2;
}

bool segmentsIntersect(Point p1, Point q1, Point p2, Point
q2) {
    int o1 = direction(p1, q1, p2);
    int o2 = direction(p1, q1, q2);
    int o3 = direction(p2, q2, p1);
    int o4 = direction(p2, q2, q1);

    if (o1 != o2 && o3 != o4) return true;
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;
    return false;
}

int main() {
    Point p1, q1, p2, q2;
    cout << "Enter coordinates of first line segment (p1 and q1):\n";
    cout << "p1: ";
    cin >> p1.x >> p1.y;
    cout << "q1: ";
    cin >> q1.x >> q1.y;

    cout << "Enter coordinates of second line segment (p2 and q2):\n";
    cout << "p2: ";
    cin >> p2.x >> p2.y;
    cout << "q2: ";
    cin >> q2.x >> q2.y;

    if (segmentsIntersect(p1, q1, p2, q2))
        cout << "Yes\n";
    else
        cout << "No\n";

    return 0;
}
```

Output:



1 $x_1 = (2,5)$

□ Label

2 $y_1 = (5,2)$

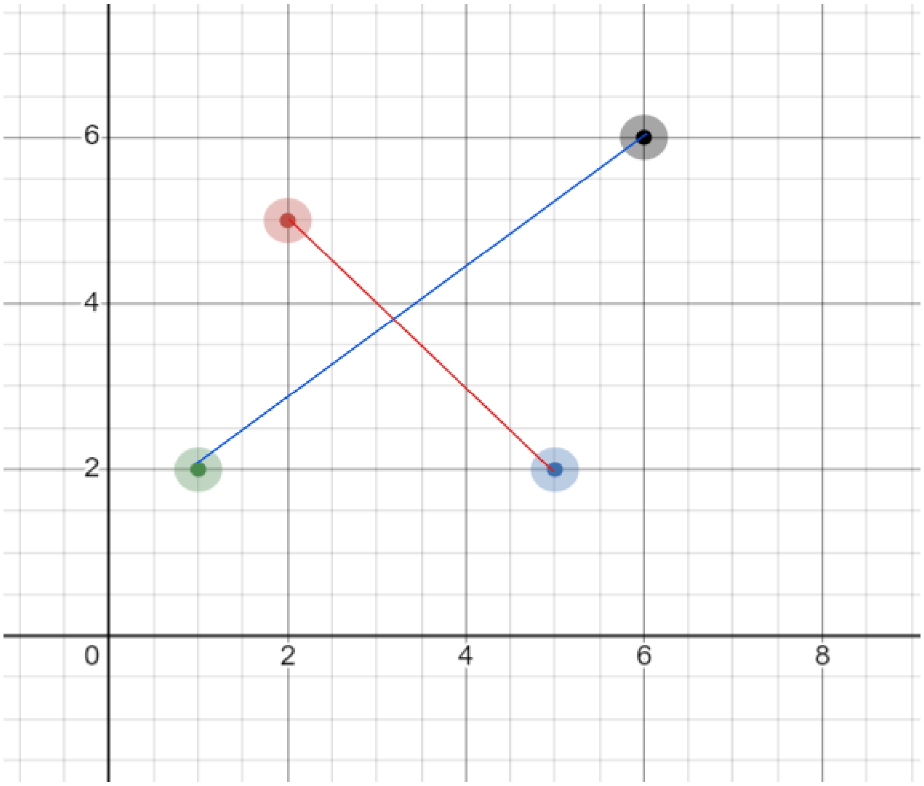
□ Label

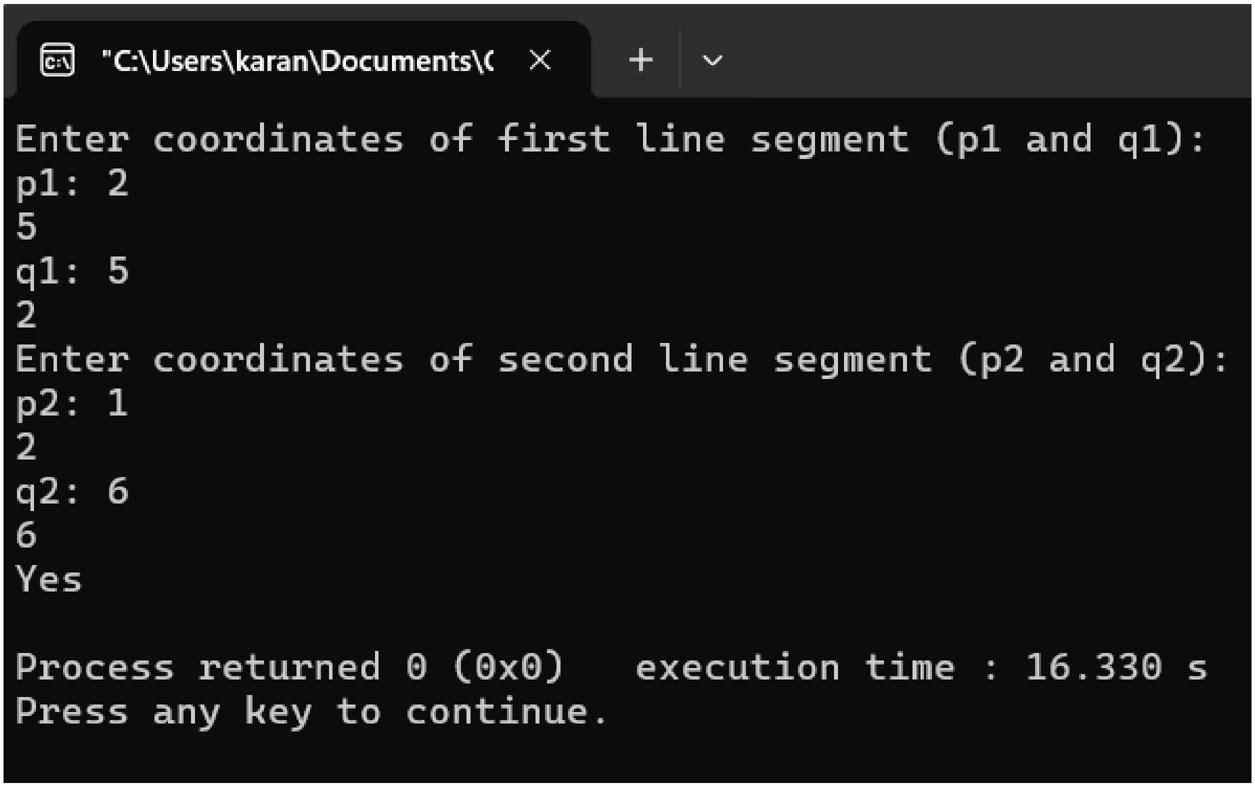
3 $x_2 = (1,2)$

□ Label

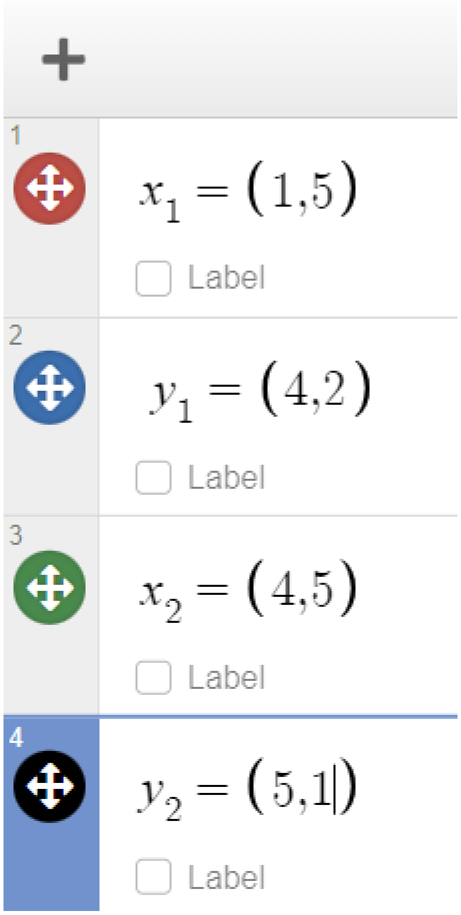
4 $y_2 = (6,6)$

□ Label





```
Enter coordinates of first line segment (p1 and q1):
p1: 2
5
q1: 5
2
Enter coordinates of second line segment (p2 and q2):
p2: 1
2
q2: 6
6
Yes

Process returned 0 (0x0)   execution time : 16.330 s
Press any key to continue.
```



1 $x_1 = (1,5)$

□ Label

2 $y_1 = (4,2)$

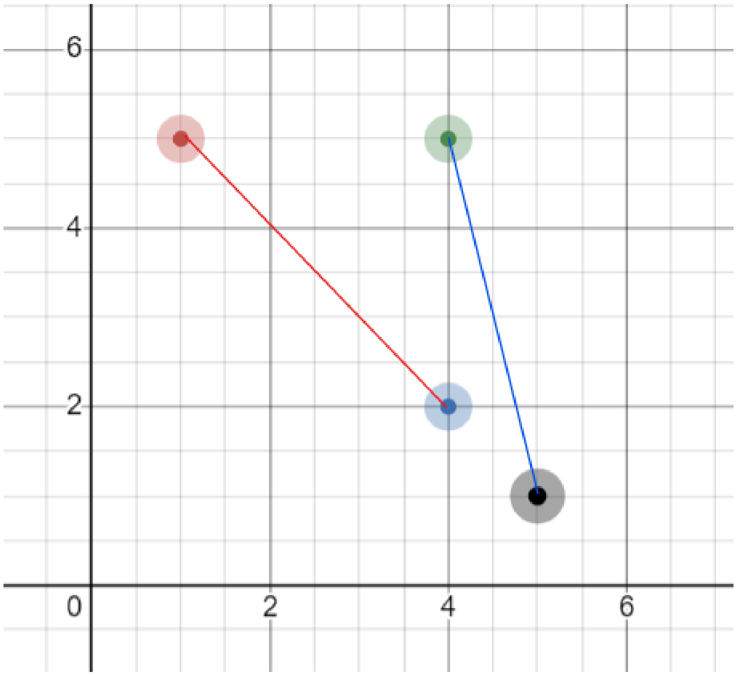
□ Label

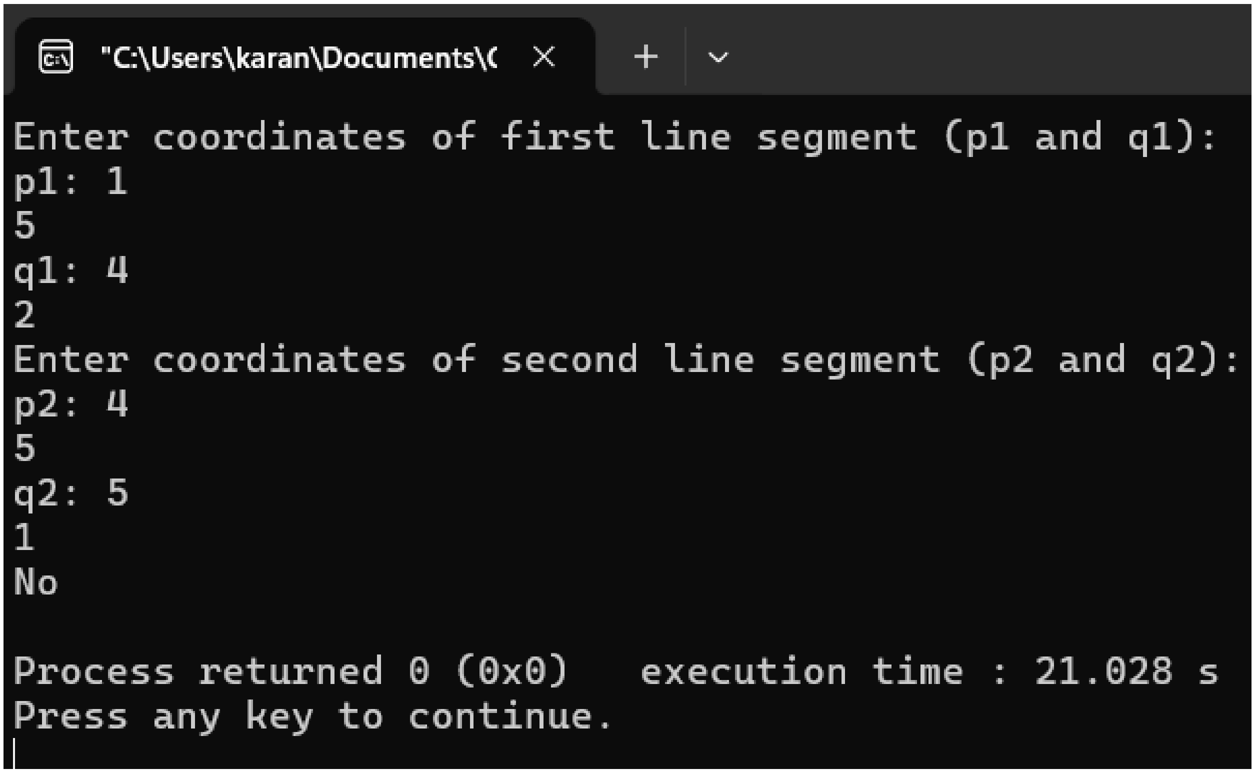
3 $x_2 = (4,5)$

□ Label

4 $y_2 = (5,1)$

□ Label





```
Enter coordinates of first line segment (p1 and q1):
p1: 1
5
q1: 4
2
Enter coordinates of second line segment (p2 and q2):
p2: 4
5
q2: 5
1
No

Process returned 0 (0x0)   execution time : 21.028 s
Press any key to continue.
```

Question2:

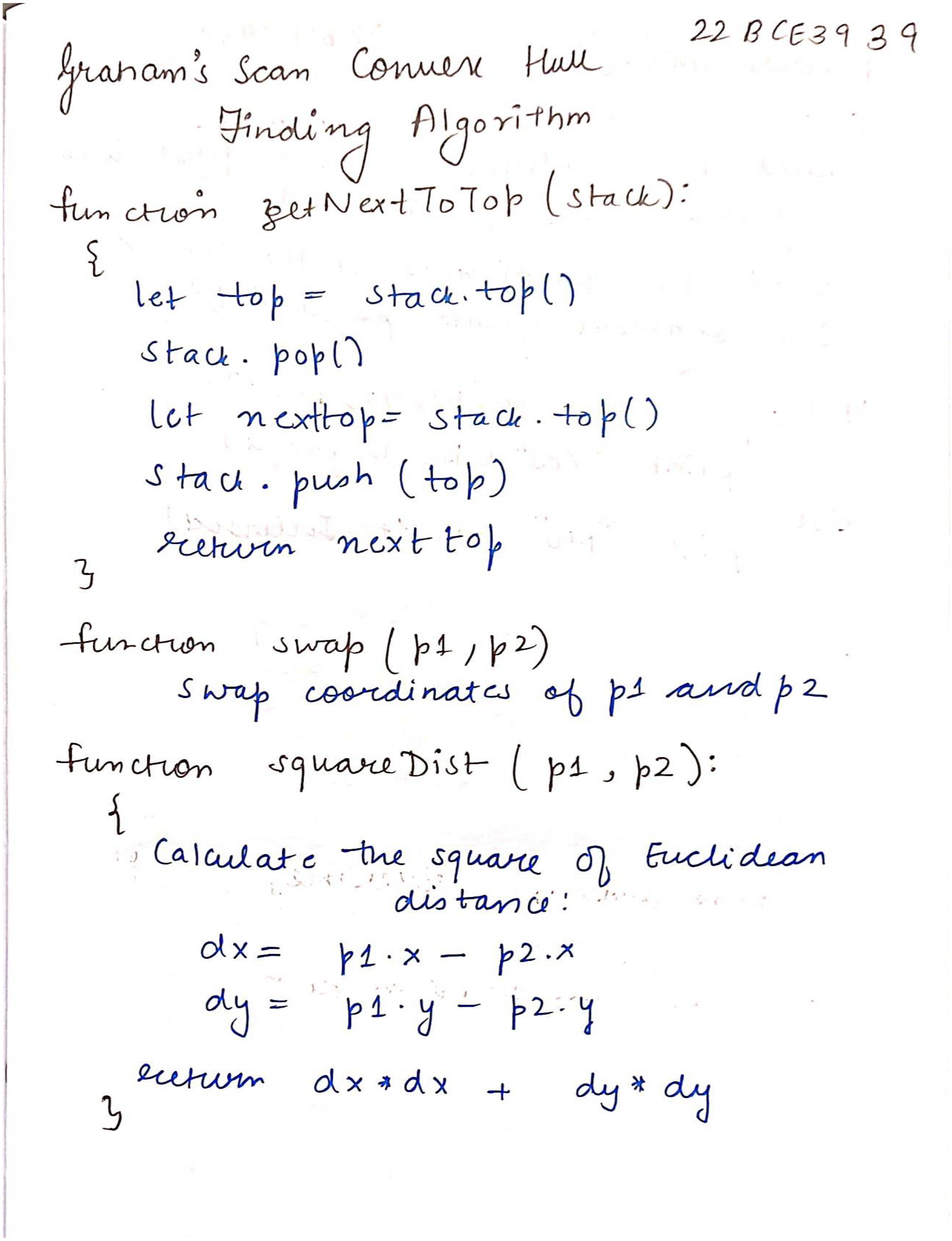
Graham’s Scan convex hull finding algorithm

Problem Statement:

Given a set of n points in the plane, find the convex hull using the Graham Scan algorithm. The convex hull is the smallest convex polygon that encloses all of the given points.

The vertices of the convex hull listed in clockwise order, starting from the point with the lowest x coordinate.

Pseudocode:



22BCE3939

Graham's Scan Convex Hull Finding Algorithm

```
function getNextToTop (stack):
 {
    let top = stack.top()
    stack.pop()
    let nexttop= stack.top()
    stack.push (top)
    return next top
 }

function swap (p1, p2)
    Swap coordinates of p1 and p2

function squareDist (p1, p2):
 {
    Calculate the square of Euclidean
                distance:
      dx = p1.x - p2.x
      dy = p1.y - p2.y
    return dx * dx + dy * dy
 }
```

```
function find Orientation (p, q, r)
 {  Calculate cross product of vectors pq and qr
    cross = (q.y - p.y) * (r.x - q.x) -
                (q. x - p.x) * (r.y - q.y)

    if  cross == 0
         return 0
 Else if  cross > 0
         Return 1
    else
         return 2
 }
function comparepoints (vp1, vp2)
 {  Cast vp1 and vp2 to point * ;
    point 1 = (point*) vp1
    point 2 = (point *) vp2
 // Find orientation from reference point to p1, p2
    Orientation = find orientation (ref point,
                                    point1,
                                    point2
 // If collinear, sort based on dist from reference
    If : orientation == 0
         if  squareDist (refpoint, point2) >=
                    square Dist (ref point, point1
              Return -1
         else
              Return 1
    Else :
```

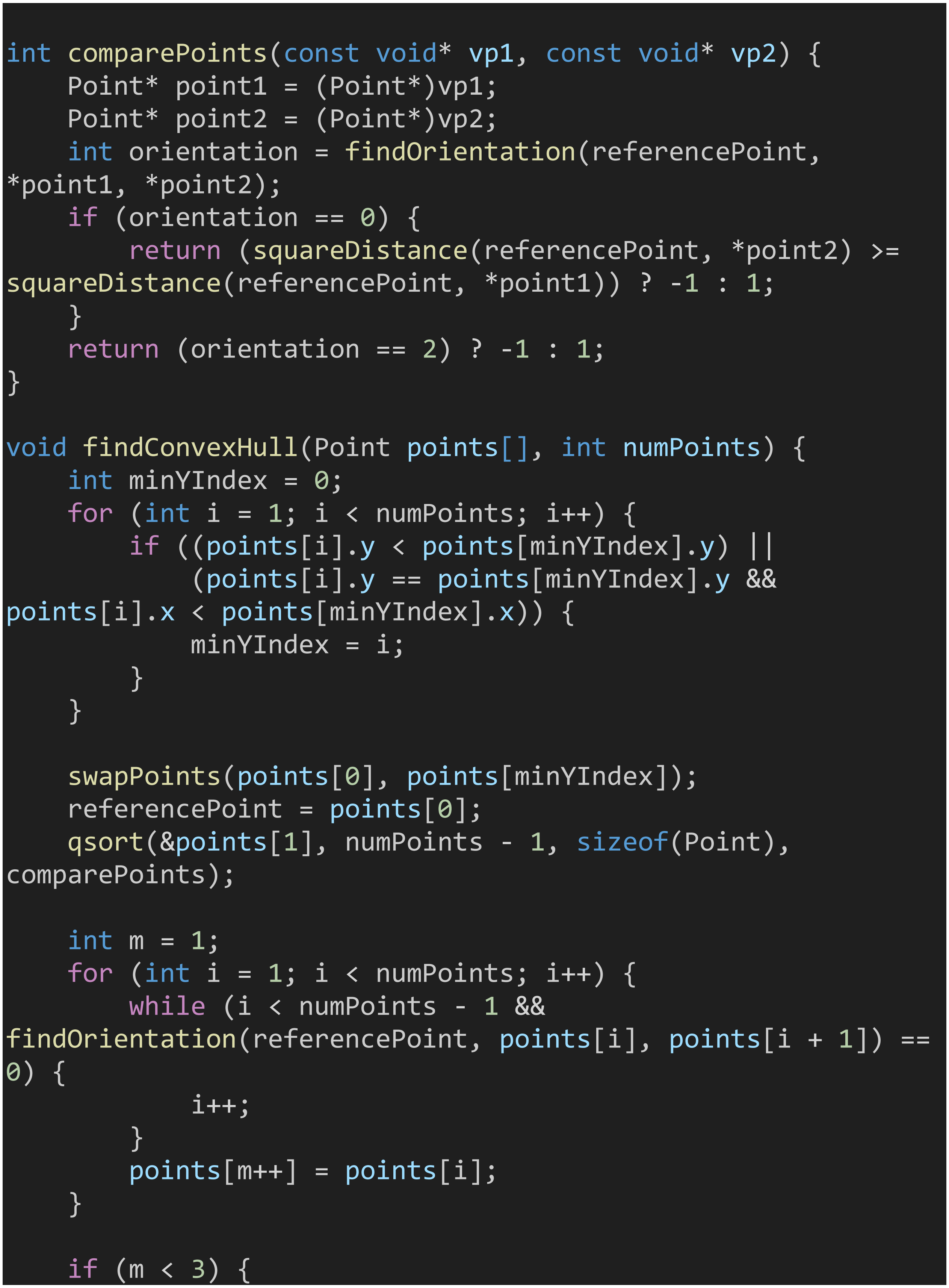
```
// sort based on orientation   22BCE3939
if orientation == 2:
    return -1
else
    return 1
}

function find convexHull ( points [], numpoints)
    // find the point with lowest y-coordinate
    Let minYIndex = 0
    for i from 1 to numpoints - 1:
        (if points[i].y < points[minYindex].y
         or
         points[i].y == points[minYindex].y
         and
         points[i].x < points[minYindex].x)
            minYIndex = i

swap (points[0], points[minYindex]
  // Make point[0] the reference point
referencePoint = points[0]
// sort remaining points by polar angle with ref
qsort (points[1:numPoints], comparePoints
// Remove collinear points, keeping only farthest point
Let m = 1
for i from 1 to numpoints - 1:
    while i < numpoints - 1 and
    find orientation (refpoint, points[i],
                       points[i+1]) == 0
        i += 1
    points[m] = points[i]
    m += 1
```

22BCE3939

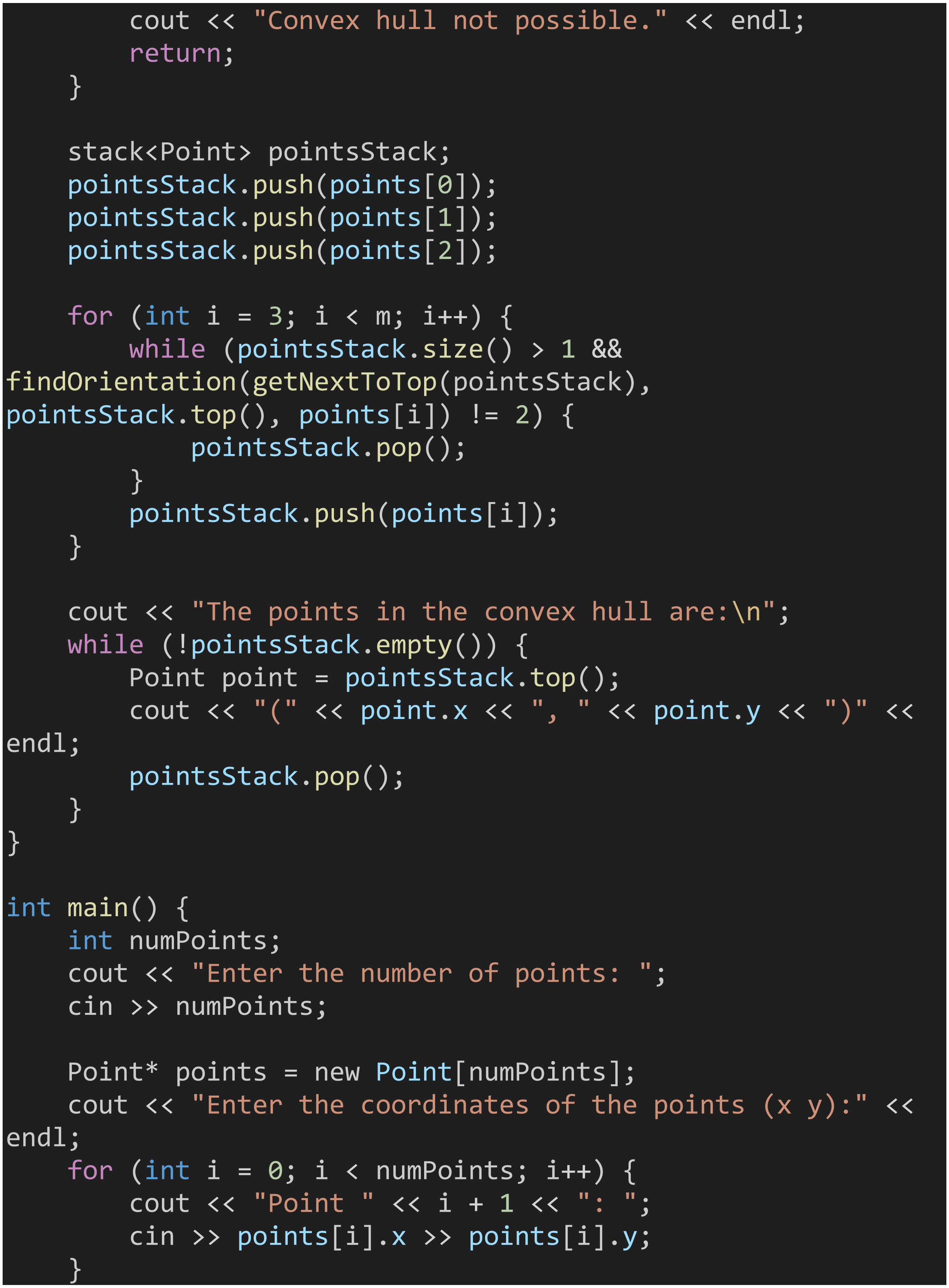
```
if m < 3:
    print " Convex Hull not possible."
        return.
// Initialise stack with first three points
let stack = new stack
stack.push( points[0])
stack.push (points[1])
stack.push(points[2])
// Process remaining points & form convex Hull
for i from 3 to m-1 :
    while stack.size() > 1 and
    find orientation( getNextToTop(stack,
                    stack.top(), points[i])!=
            stack.pop()
    stack.push (points[i])
// Print the output
print " The points in convexHull are:"
while not stack.empty():
    let point = stack.top()
    print point.x, point.y
    stack.pop()
```

Source Code:

```
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point {
    int x, y;
};

Point referencePoint;

Point getNextToTop(stack<Point>& pointsStack) {
    Point topPoint = pointsStack.top();
    pointsStack.pop();
    Point nextToTopPoint = pointsStack.top();
    pointsStack.push(topPoint);
    return nextToTopPoint;
}

void swapPoints(Point& point1, Point& point2) {
    Point temp = point1;
    point1 = point2;
    point2 = temp;
}

int squareDistance(Point point1, Point point2) {
    return (point1.x - point2.x) * (point1.x - point2.x) +
(point1.y - point2.y) * (point1.y - point2.y);
}

int findOrientation(Point p, Point q, Point r) {
    int value = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) *
(r.y - q.y);
    if (value == 0) return 0; // Collinear
    return (value > 0) ? 1 : 2; // Clockwise or
counterclockwise
}
```

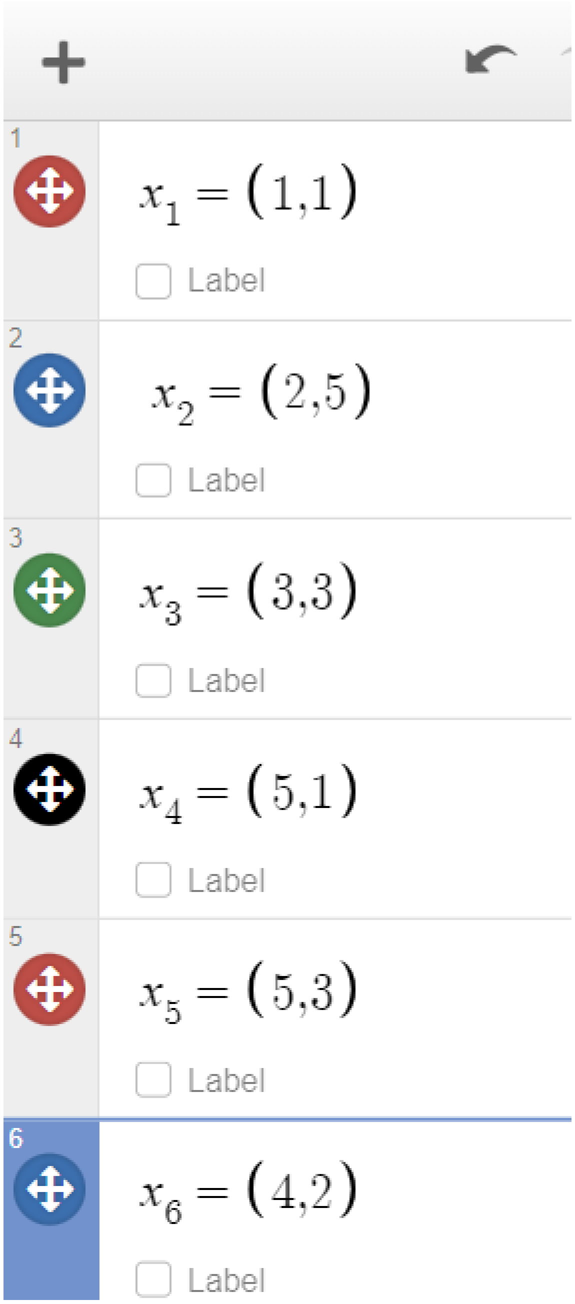


```
int comparePoints(const void* vp1, const void* vp2) {
    Point* point1 = (Point*)vp1;
    Point* point2 = (Point*)vp2;
    int orientation = findOrientation(referencePoint,
*point1, *point2);
    if (orientation == 0) {
        return (squareDistance(referencePoint, *point2) >=
squareDistance(referencePoint, *point1)) ? -1 : 1;
    }
    return (orientation == 2) ? -1 : 1;
}

void findConvexHull(Point points[], int numPoints) {
    int minYIndex = 0;
    for (int i = 1; i < numPoints; i++) {
        if ((points[i].y < points[minYIndex].y) ||
            (points[i].y == points[minYIndex].y &&
points[i].x < points[minYIndex].x)) {
            minYIndex = i;
        }
    }

    swapPoints(points[0], points[minYIndex]);
    referencePoint = points[0];
    qsort(&points[1], numPoints - 1, sizeof(Point),
comparePoints);

    int m = 1;
    for (int i = 1; i < numPoints; i++) {
        while (i < numPoints - 1 &&
findOrientation(referencePoint, points[i], points[i + 1]) ==
0) {
            i++;
        }
        points[m++] = points[i];
    }

    if (m < 3) {
```



```
        cout << "Convex hull not possible." << endl;
        return;
    }

    stack<Point> pointsStack;
    pointsStack.push(points[0]);
    pointsStack.push(points[1]);
    pointsStack.push(points[2]);

    for (int i = 3; i < m; i++) {
        while (pointsStack.size() > 1 &&
findOrientation(getNextToTop(pointsStack),
pointsStack.top(), points[i]) != 2) {
            pointsStack.pop();
        }
        pointsStack.push(points[i]);
    }

    cout << "The points in the convex hull are:\n";
    while (!pointsStack.empty()) {
        Point point = pointsStack.top();
        cout << "(" << point.x << ", " << point.y << ")" <<
endl;
        pointsStack.pop();
    }
}

int main() {
    int numPoints;
    cout << "Enter the number of points: ";
    cin >> numPoints;

    Point* points = new Point[numPoints];
    cout << "Enter the coordinates of the points (x y):" <<
endl;
    for (int i = 0; i < numPoints; i++) {
        cout << "Point " << i + 1 << ": ";
        cin >> points[i].x >> points[i].y;
    }
    findConvexHull(points, numPoints);
    delete[] points;

    return 0;
}
```

Outputs:



1 $x_1 = (1,1)$

□ Label

2 $x_2 = (2,5)$

□ Label

3 $x_3 = (3,3)$

□ Label

4 $x_4 = (5,1)$

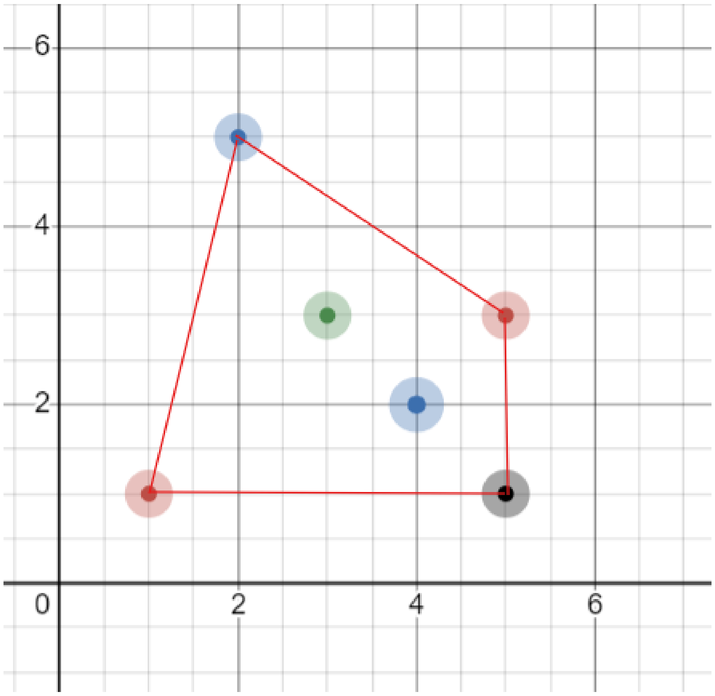
□ Label

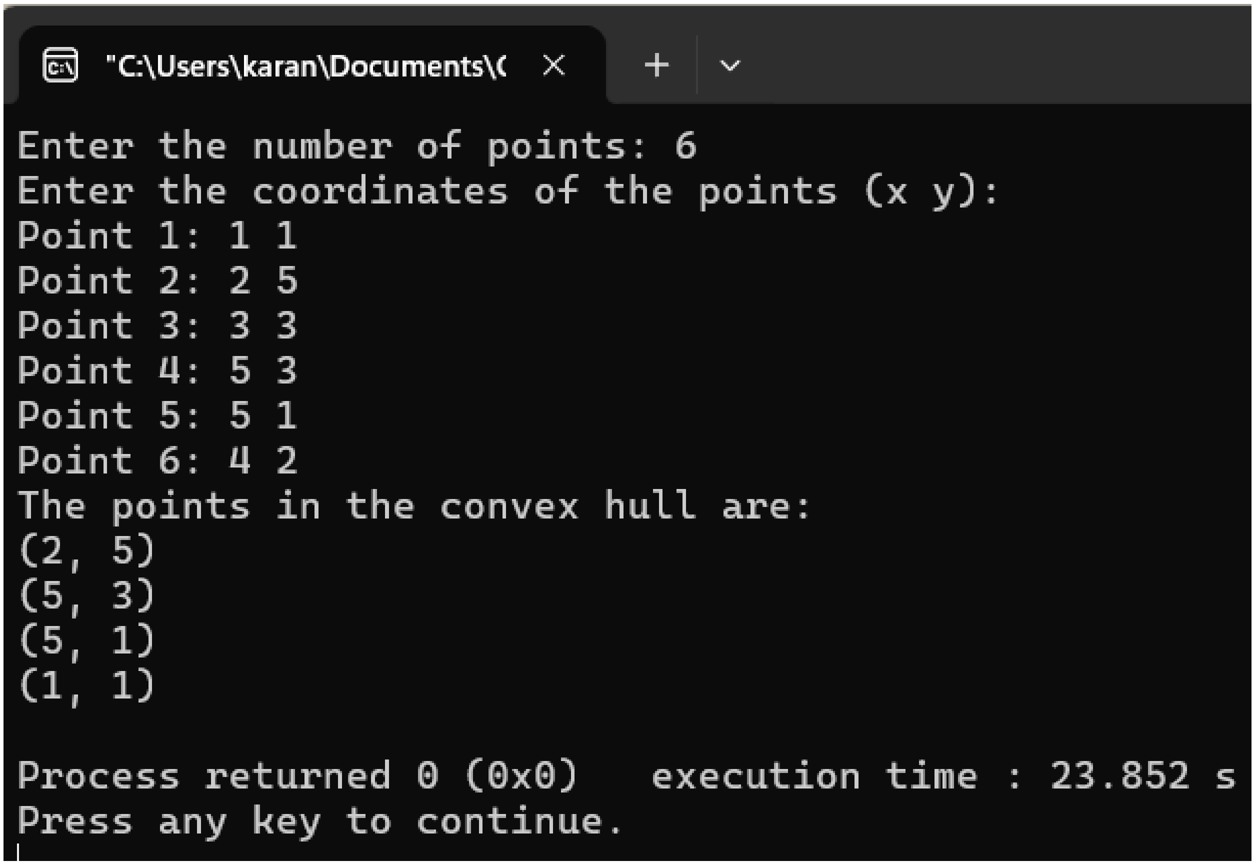
5 $x_5 = (5,3)$

□ Label

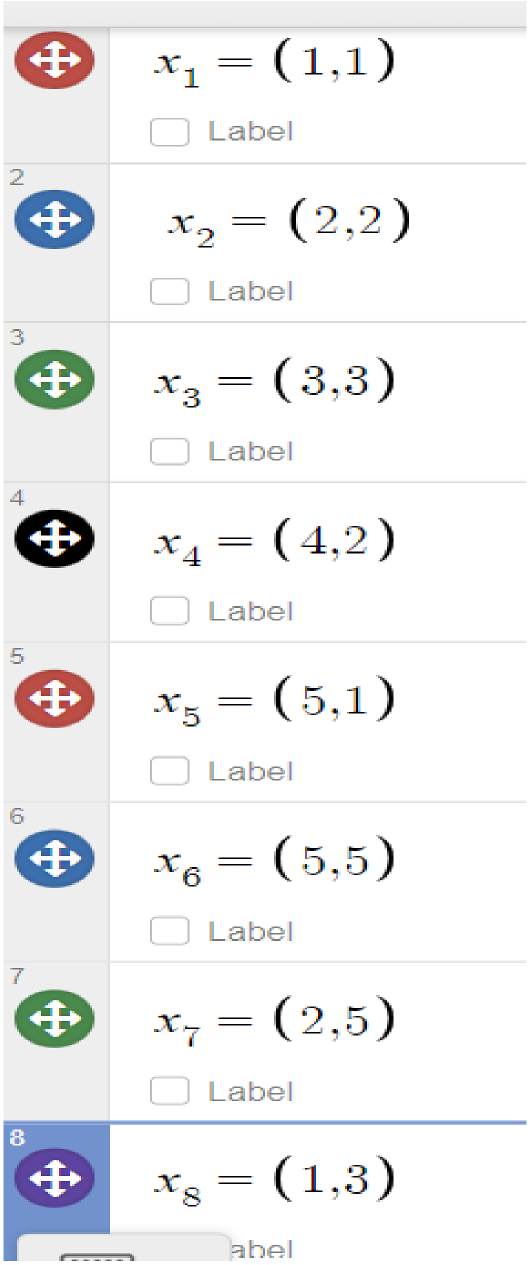
6 $x_6 = (4,2)$

□ Label





```
Enter the number of points: 6
Enter the coordinates of the points (x y):
Point 1: 1 1
Point 2: 2 5
Point 3: 3 3
Point 4: 5 3
Point 5: 5 1
Point 6: 4 2
The points in the convex hull are:
(2, 5)
(5, 3)
(5, 1)
(1, 1)

Process returned 0 (0x0)   execution time : 23.852 s
Press any key to continue.
```



$x_1 = (1,1)$

□ Label

2 $x_2 = (2,2)$

□ Label

3 $x_3 = (3,3)$

□ Label

4 $x_4 = (4,2)$

□ Label

5 $x_5 = (5,1)$

□ Label

6 $x_6 = (5,5)$

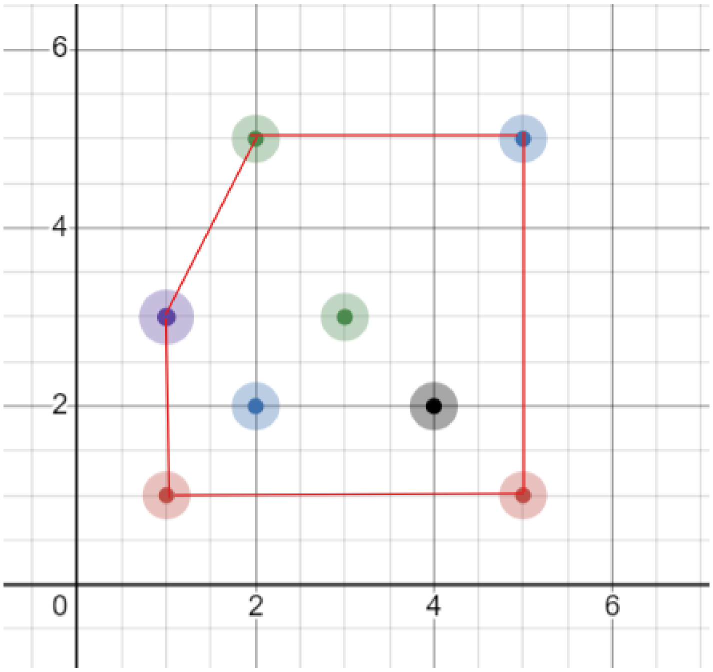
□ Label

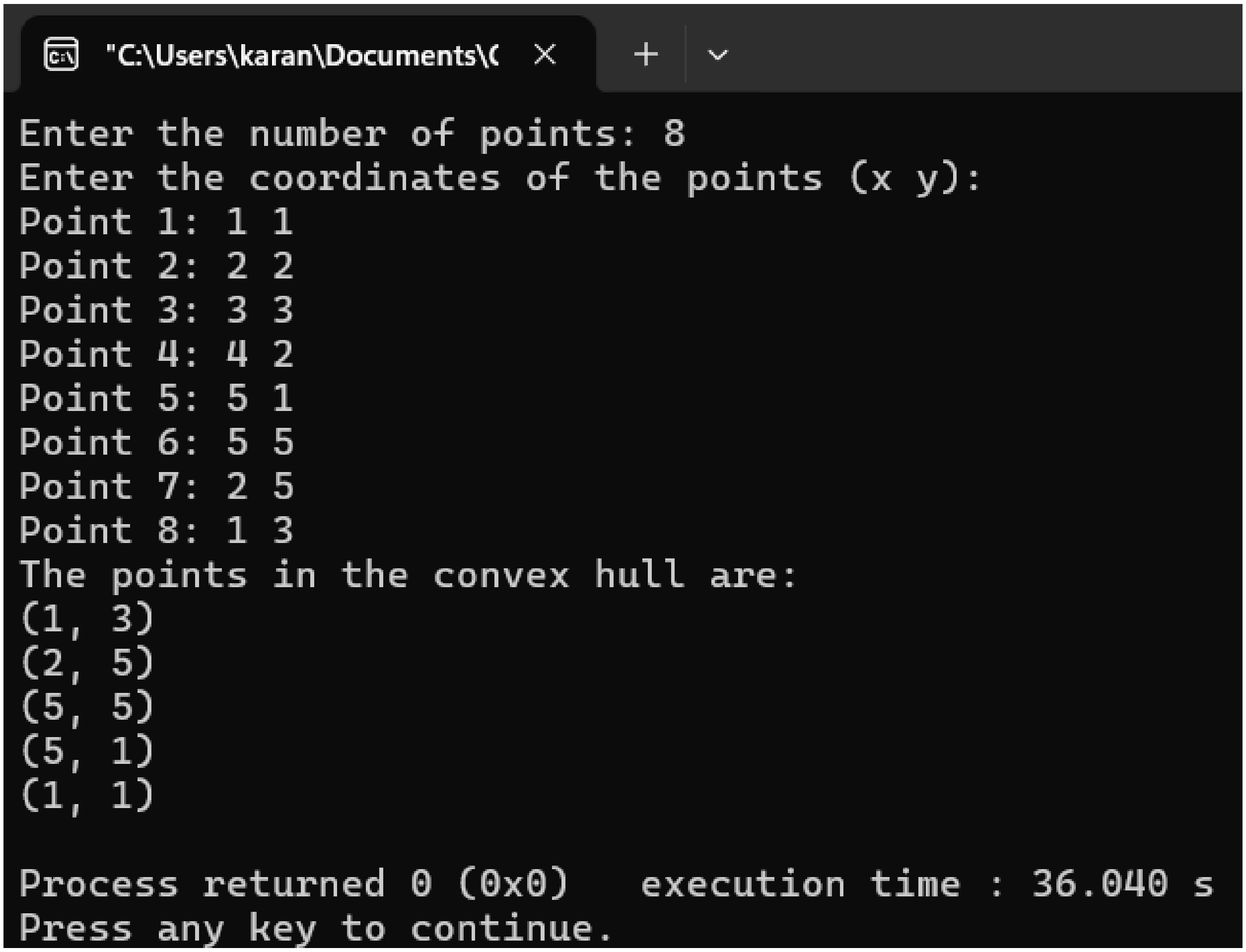
7 $x_7 = (2,5)$

□ Label

8 $x_8 = (1,3)$

abel





```
Enter the number of points: 8
Enter the coordinates of the points (x y):
Point 1: 1 1
Point 2: 2 2
Point 3: 3 3
Point 4: 4 2
Point 5: 5 1
Point 6: 5 5
Point 7: 2 5
Point 8: 1 3
The points in the convex hull are:
(1, 3)
(2, 5)
(5, 5)
(5, 1)
(1, 1)

Process returned 0 (0x0)   execution time : 36.040 s
Press any key to continue.
```

Question3:

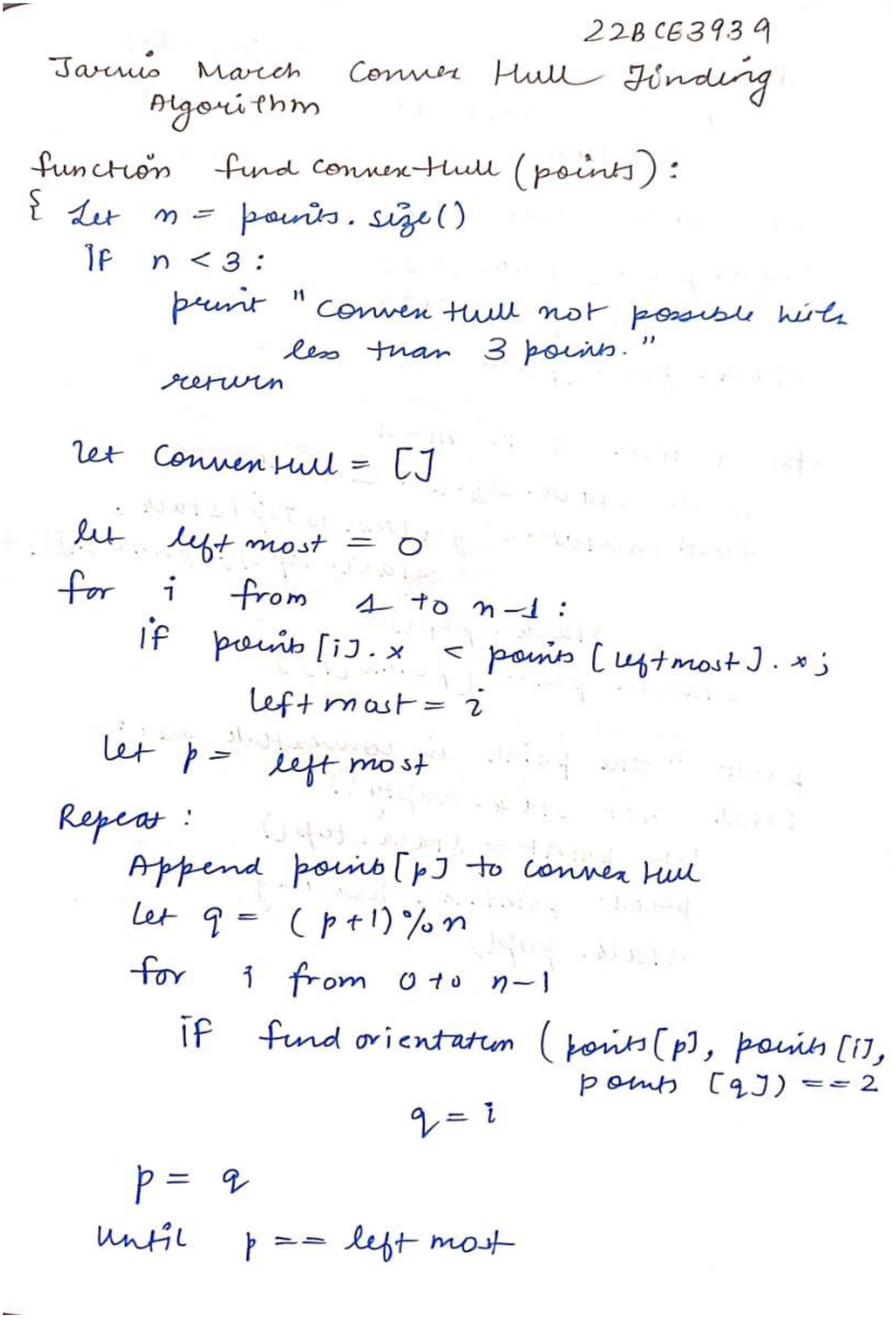
Jarvis March convex hull finding algorithm

Problem Statement:

Given a set of n points in the plane, find the convex hull using the Graham Scan algorithm. The convex hull is the smallest convex polygon that encloses all of the given points.

The vertices of the convex hull listed in clockwise order, starting from the point with the lowest x coordinate.

Pseudocode:



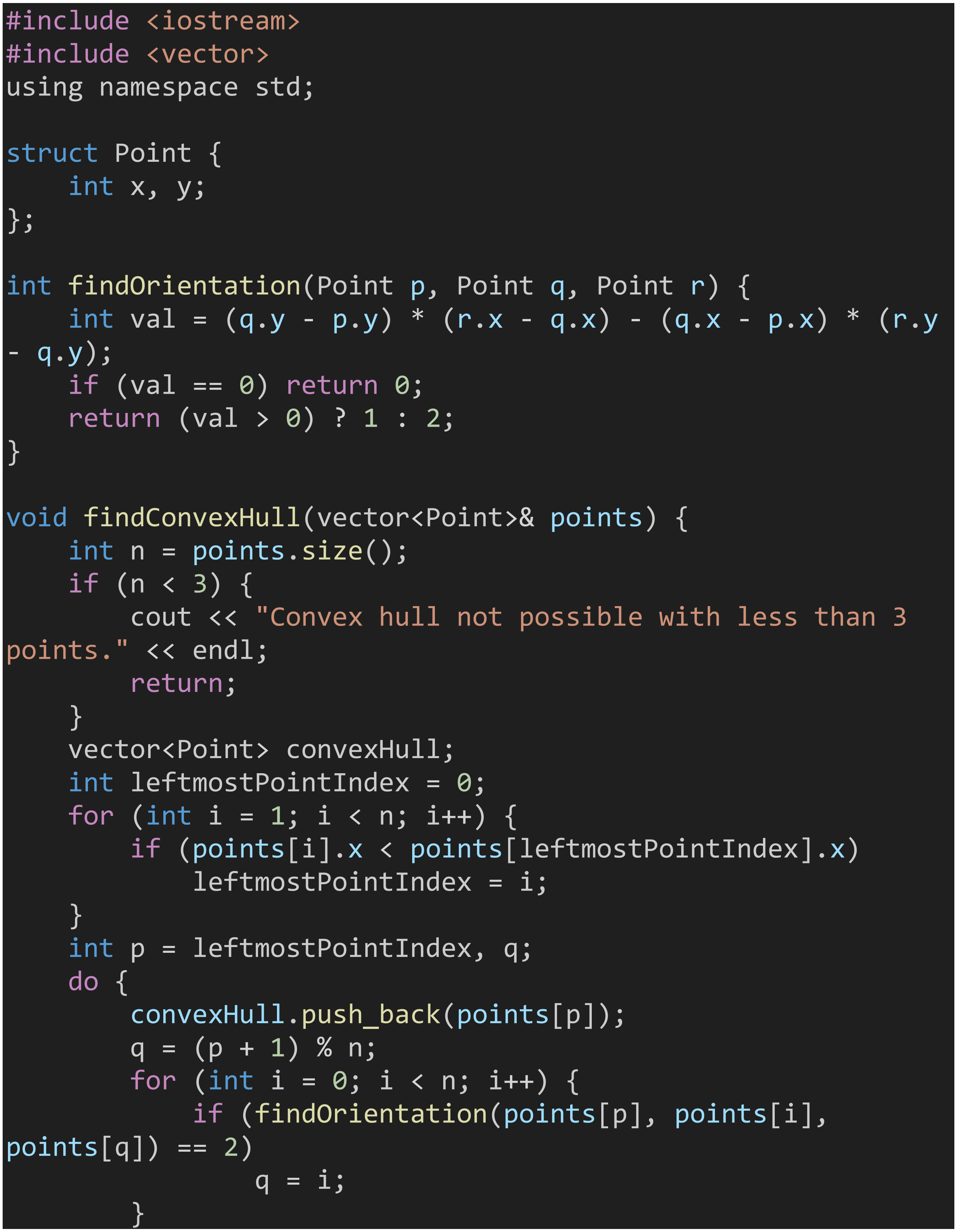
22BCE3939

Jarvis March Convex Hull Finding Algorithm

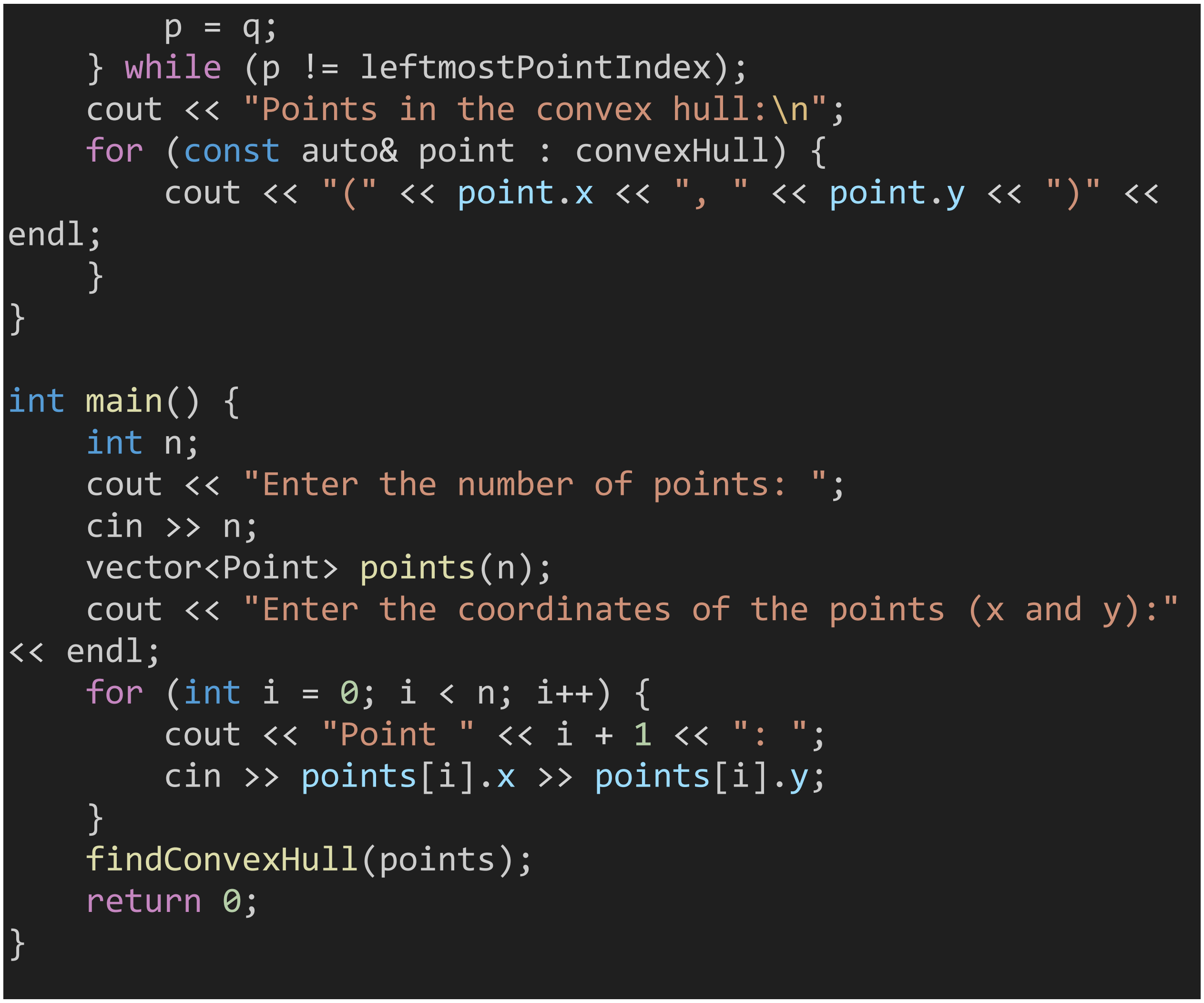
```
function find convex Hull (points):
{ Let n = points. size()
   If n < 3 :
        print "convex Hull not possible with
               less than 3 points."
        return

   let ConvexHull = []

   let left most = 0
   for i from 1 to n-1 :
       if points [i]. x < points [leftmost]. x ;
            leftmost = i
   Let p = left most
   Repeat :
      Append points [p] to convex Hull
      Let q = (p+1) % n
      for i from 0 to n-1
         if find orientation (points[p], points [i],
                                points [q]) == 2
                    q = i

      p = q
   Until p == left most
```

```
  print "points in the Convex Hull:"
  For each point in convex Hull:
      Print the coordinates (points.x, point.y)
  }
function main()
  {
    Read the number of points, n
    Initialise a list of point objects,
     of size n
    Read the coordinates of each
    point from user input
    Call find convexHull with points
}
```

Source code:

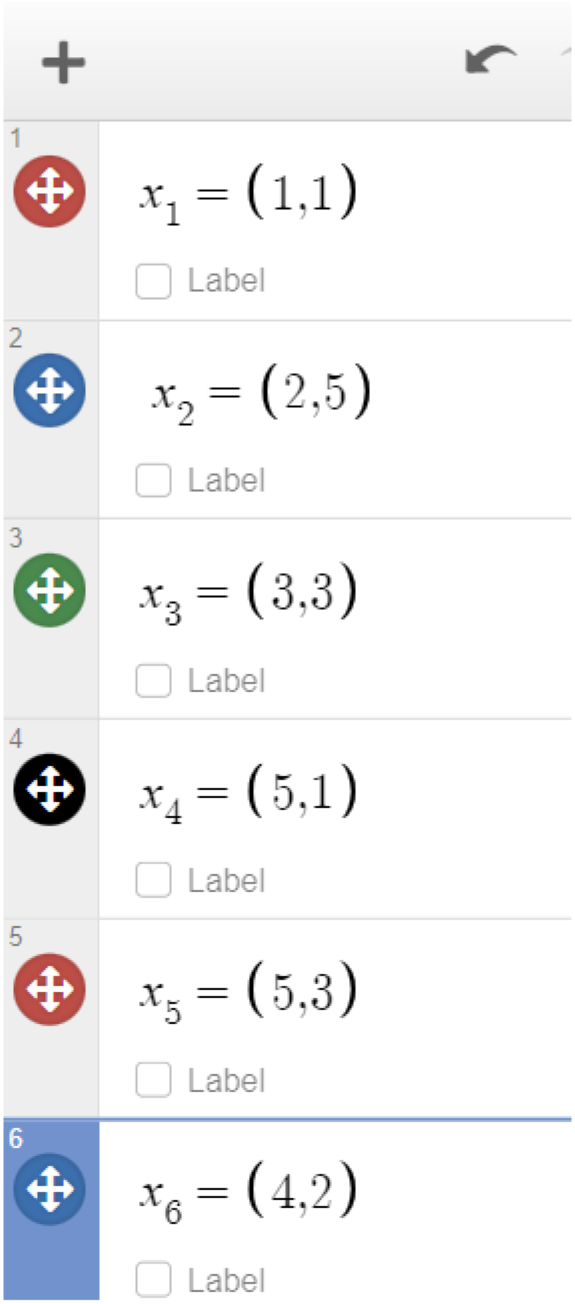


```
#include <iostream>
#include <vector>
using namespace std;

struct Point {
    int x, y;
};

int findOrientation(Point p, Point q, Point r) {
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y
- q.y);
    if (val == 0) return 0;
    return (val > 0) ? 1 : 2;
}

void findConvexHull(vector<Point>& points) {
    int n = points.size();
    if (n < 3) {
        cout << "Convex hull not possible with less than 3
points." << endl;
        return;
    }
    vector<Point> convexHull;
    int leftmostPointIndex = 0;
    for (int i = 1; i < n; i++) {
        if (points[i].x < points[leftmostPointIndex].x)
            leftmostPointIndex = i;
    }
    int p = leftmostPointIndex, q;
    do {
        convexHull.push_back(points[p]);
        q = (p + 1) % n;
        for (int i = 0; i < n; i++) {
            if (findOrientation(points[p], points[i],
points[q]) == 2)
                q = i;
        }
```



```
        p = q;
    } while (p != leftmostPointIndex);
    cout << "Points in the convex hull:\n";
    for (const auto& point : convexHull) {
        cout << "(" << point.x << ", " << point.y << ")" <<
endl;
    }
}

int main() {
    int n;
    cout << "Enter the number of points: ";
    cin >> n;
    vector<Point> points(n);
    cout << "Enter the coordinates of the points (x and y):"
<< endl;
    for (int i = 0; i < n; i++) {
        cout << "Point " << i + 1 << ": ";
        cin >> points[i].x >> points[i].y;
    }
    findConvexHull(points);
    return 0;
}
```

Output:



1 $x_1 = (1,1)$

□ Label

2 $x_2 = (2,5)$

□ Label

3 $x_3 = (3,3)$

□ Label

4 $x_4 = (5,1)$

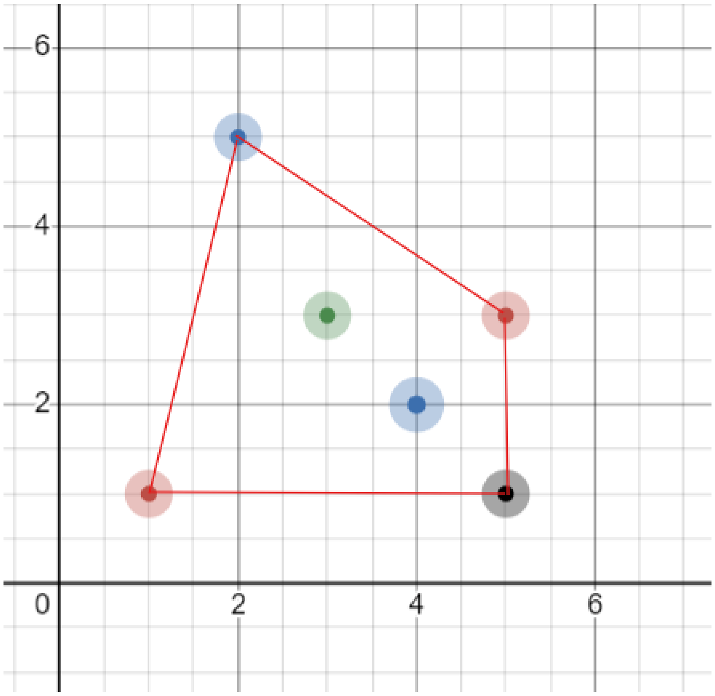
□ Label

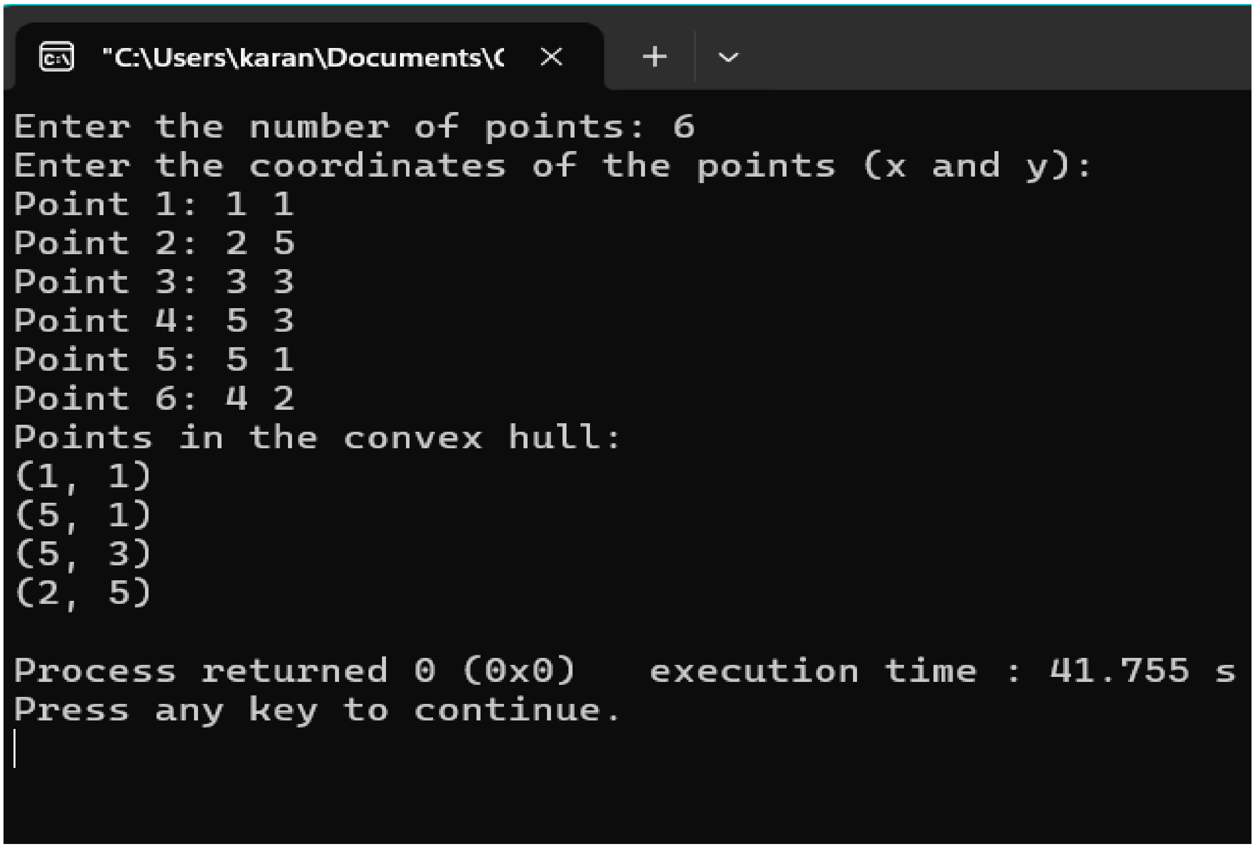
5 $x_5 = (5,3)$

□ Label

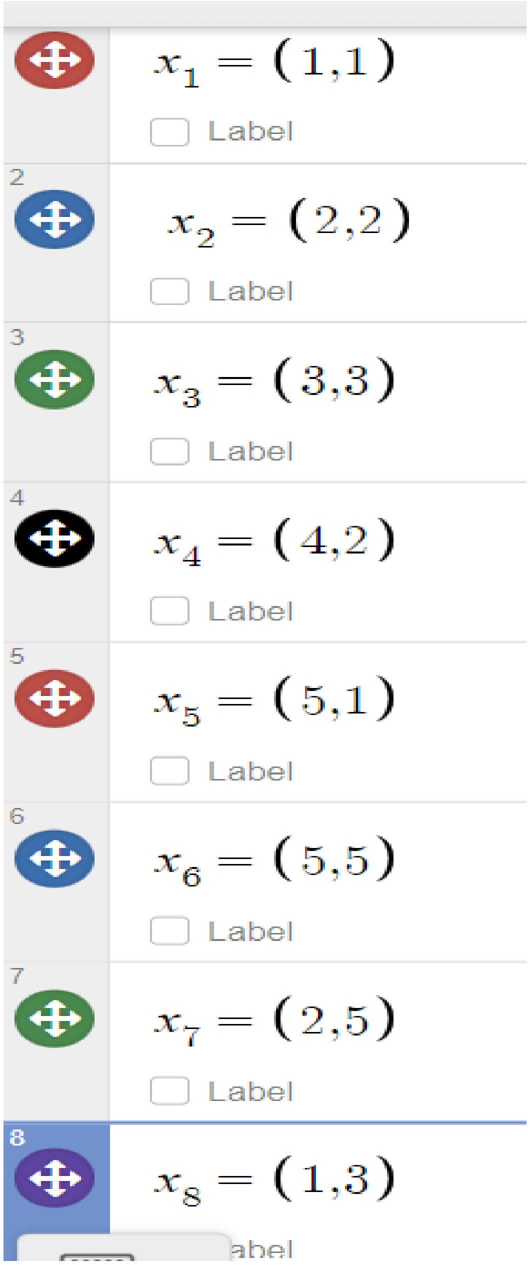
6 $x_6 = (4,2)$

□ Label





```
Enter the number of points: 6
Enter the coordinates of the points (x and y):
Point 1: 1 1
Point 2: 2 5
Point 3: 3 3
Point 4: 5 3
Point 5: 5 1
Point 6: 4 2
Points in the convex hull:
(1, 1)
(5, 1)
(5, 3)
(2, 5)

Process returned 0 (0x0)   execution time : 41.755 s
Press any key to continue.
```



$x_1 = (1,1)$

□ Label

2 $x_2 = (2,2)$

□ Label

3 $x_3 = (3,3)$

□ Label

4 $x_4 = (4,2)$

□ Label

5 $x_5 = (5,1)$

□ Label

6 $x_6 = (5,5)$

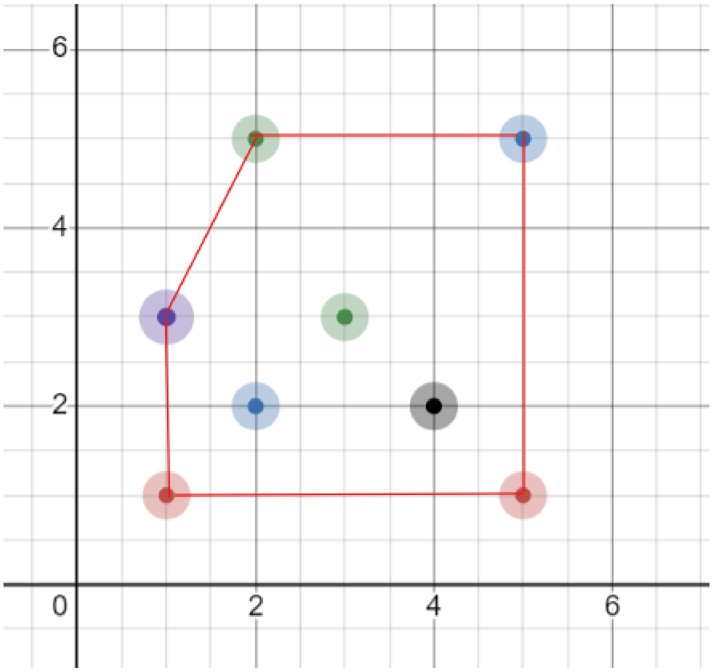
□ Label

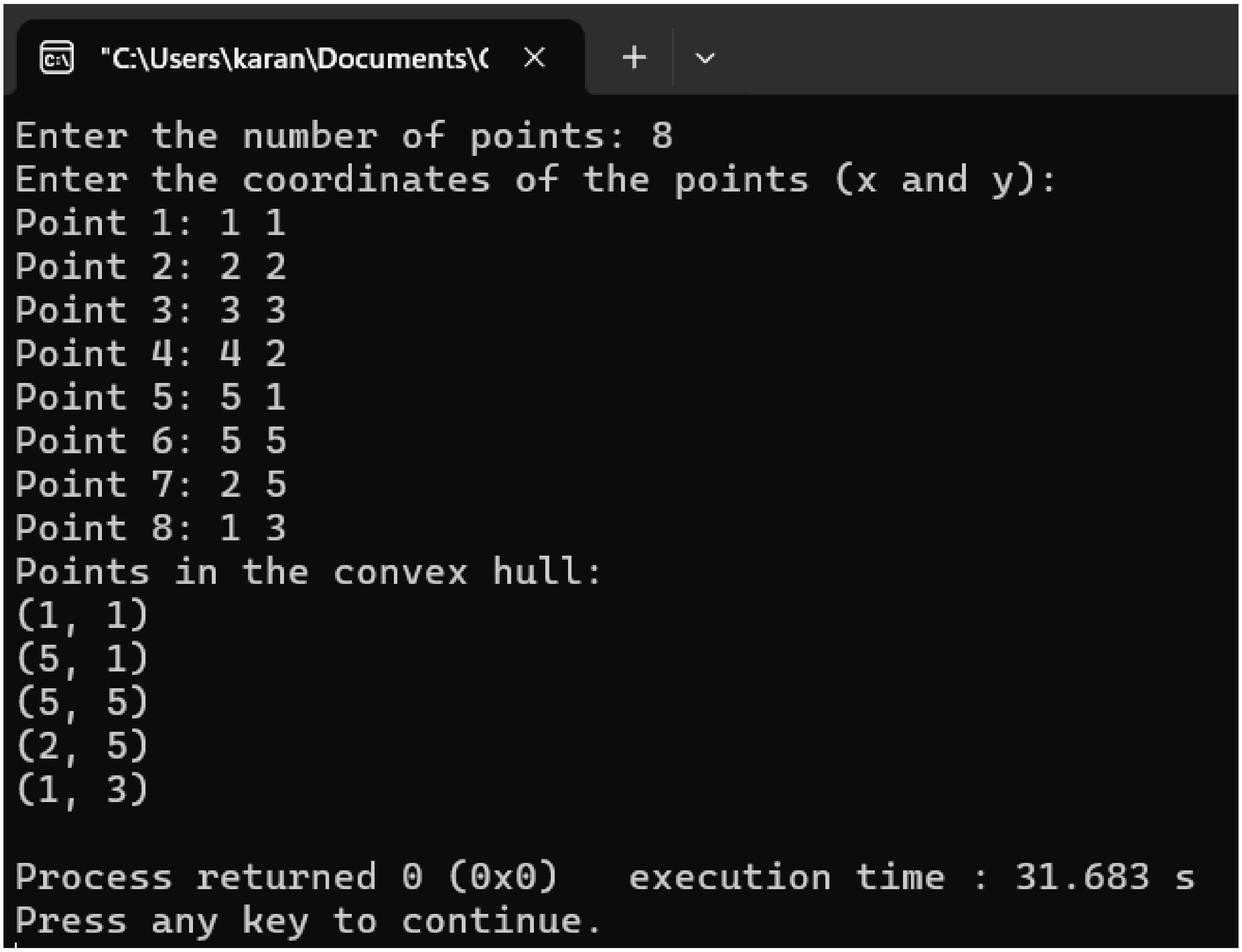
7 $x_7 = (2,5)$

□ Label

8 $x_8 = (1,3)$

abel





```
Enter the number of points: 8
Enter the coordinates of the points (x and y):
Point 1: 1 1
Point 2: 2 2
Point 3: 3 3
Point 4: 4 2
Point 5: 5 1
Point 6: 5 5
Point 7: 2 5
Point 8: 1 3
Points in the convex hull:
(1, 1)
(5, 1)
(5, 5)
(2, 5)
(1, 3)

Process returned 0 (0x0)   execution time : 31.683 s
Press any key to continue.
```

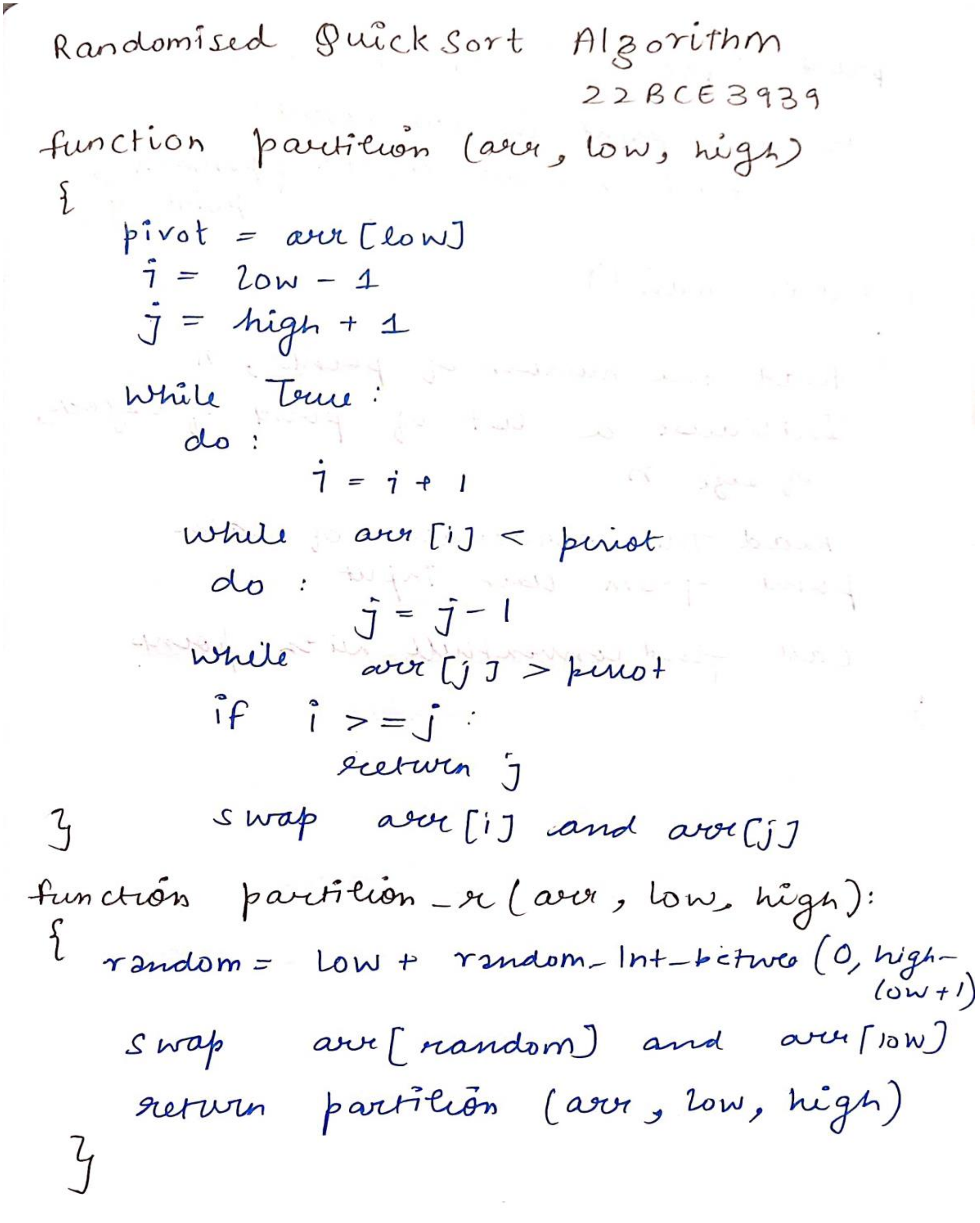
Question4:

Randomized Quicksort

Problem Statement:

You are given an array of n integers. Implement the Randomized Quicksort algorithm to sort this array in non-decreasing order.

Pseudocode:



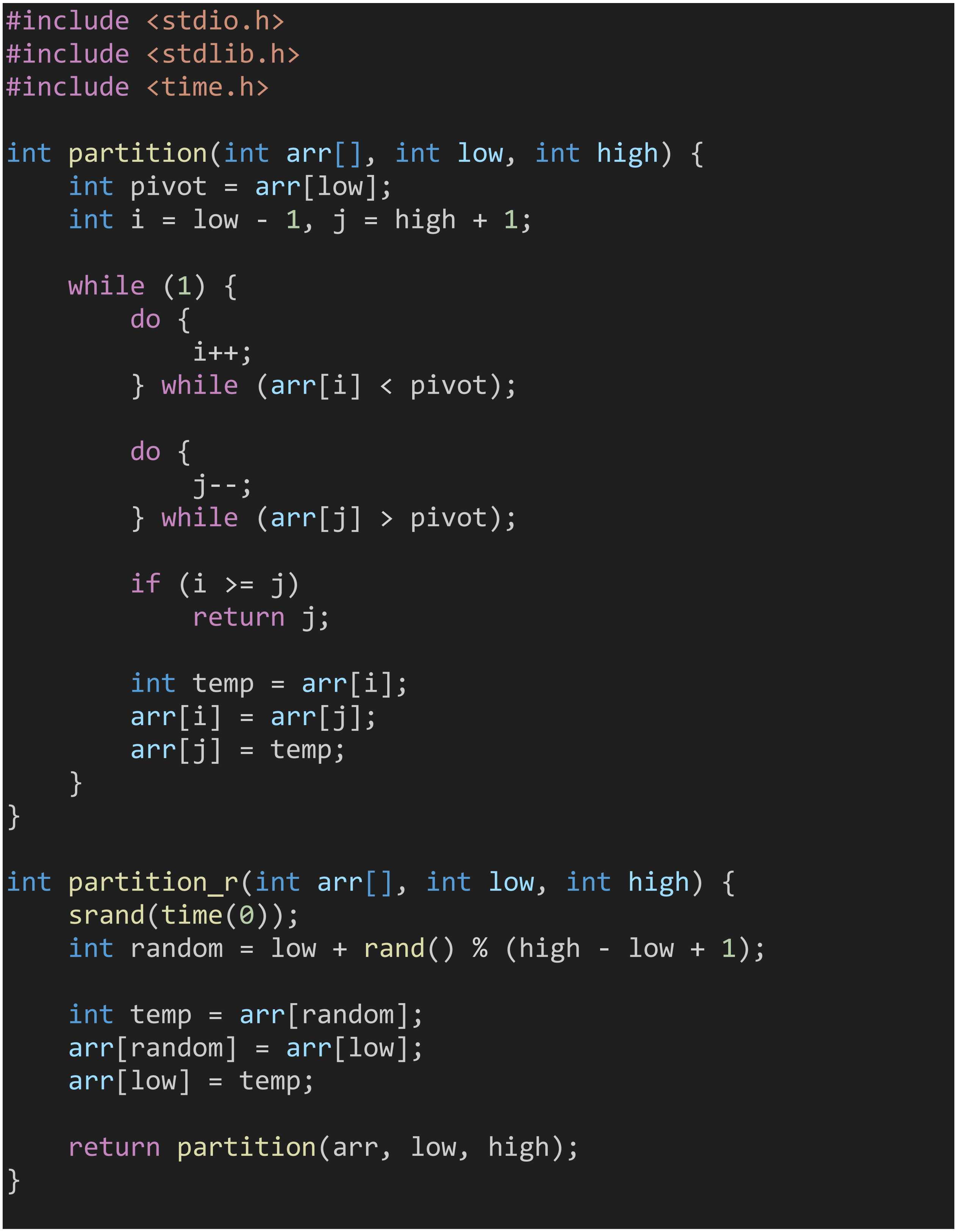
Randomised Quick Sort Algorithm

22BCE3939

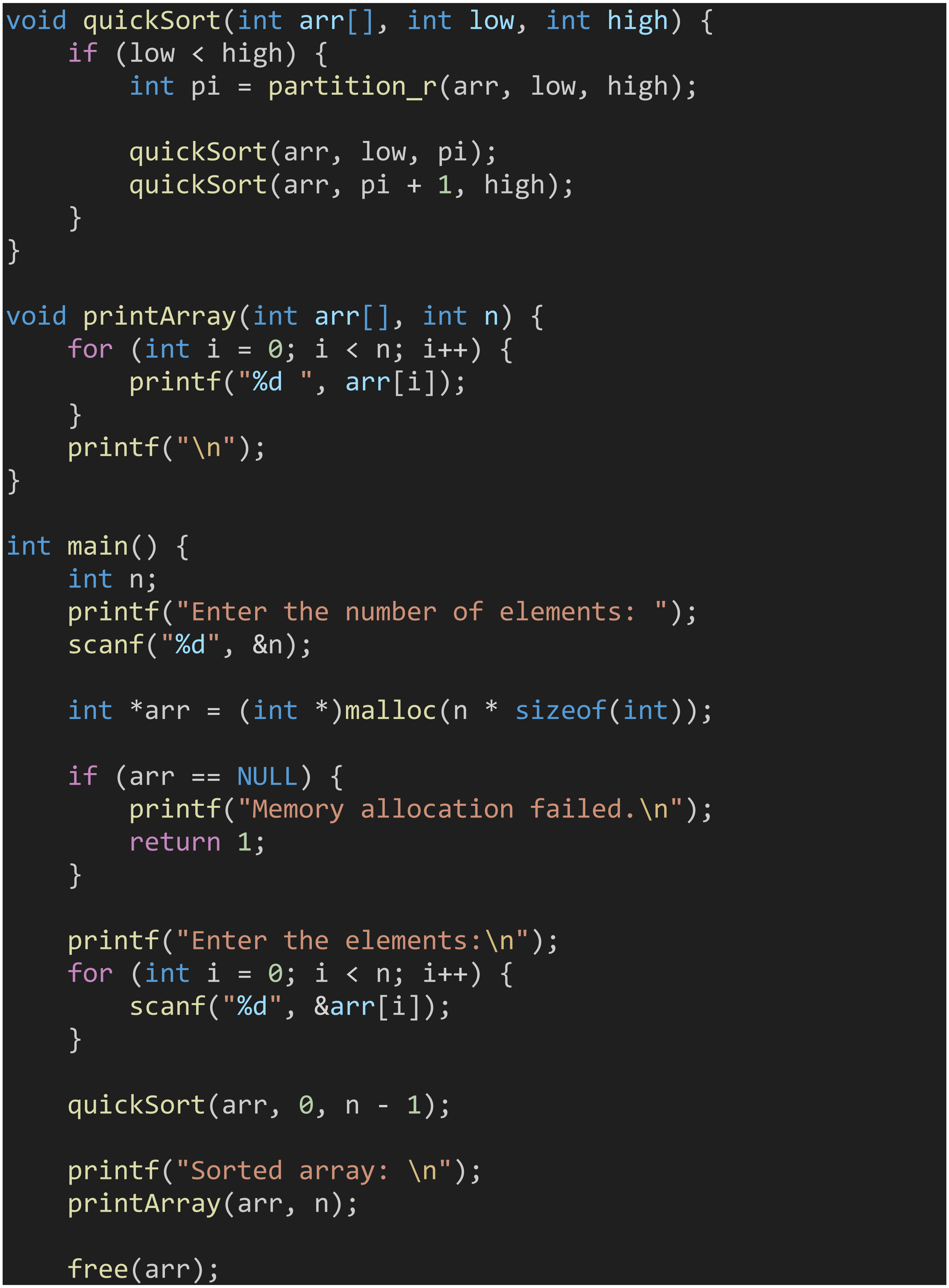
```
function partition (arr, low, high)
{
    pivot = arr[low]
    i = low - 1
    j = high + 1

    while True:
        do:
            i = i + 1
        while arr[i] < pivot
        do:
            j = j - 1
        while arr[j] > pivot
        if i >= j:
            return j
        swap arr[i] and arr[j]
}
function partition_r (arr, low, high):
{
    random = low + random_Int_betwee (0, high-(low+1))
    swap arr[random] and arr[low]
    return partition (arr, low, high)
}
```

```
function quick sort (arr, low, high):
 {  if low < high:
        pi = partition_r (arr, low, high)

       quicksort ( arr, low, pi)
       quicksort (arr, pi+1, high)
    }
function printArray (arr, n)
    {
      for  i from 0 to n-1 :
             print  arr [1]
    } print  new line

function main ()
   {
      read n
      read elements
      call quicksort ( arr, 0, n-1):
      print Array (arr, n)
   } free - memory
```

Source Code:

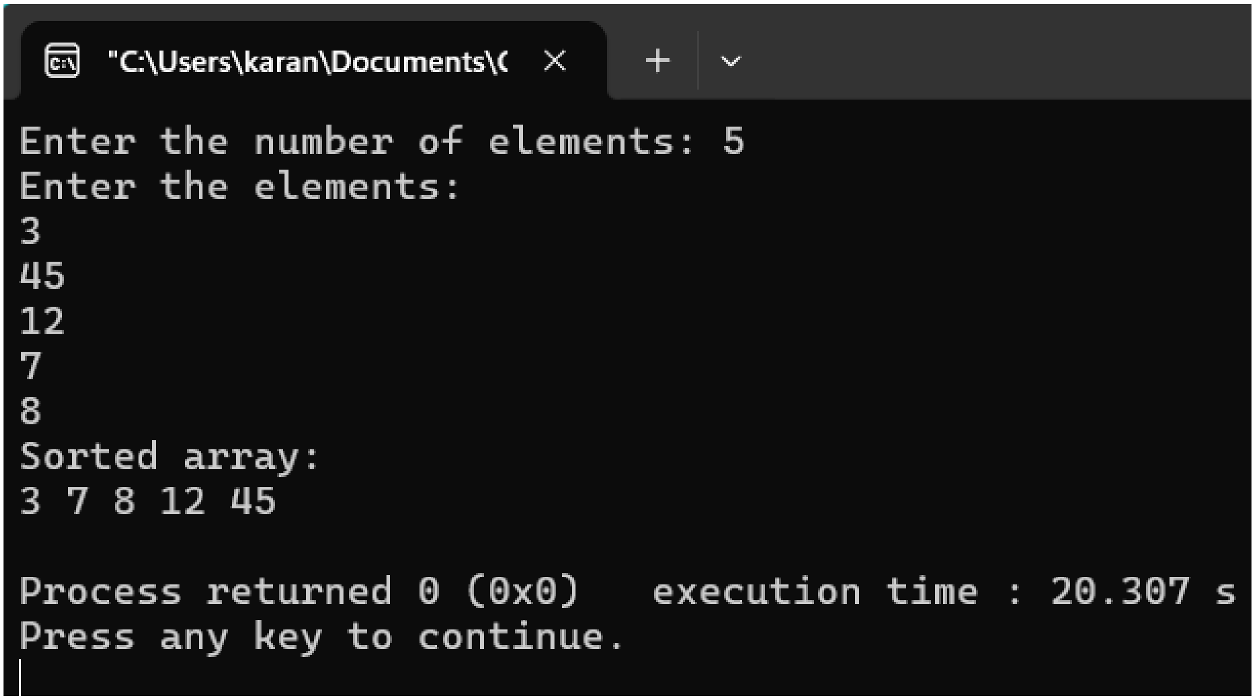


```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int partition(int arr[], int low, int high) {
    int pivot = arr[low];
    int i = low - 1, j = high + 1;

    while (1) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i >= j)
            return j;

        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

int partition_r(int arr[], int low, int high) {
    srand(time(0));
    int random = low + rand() % (high - low + 1);

    int temp = arr[random];
    arr[random] = arr[low];
    arr[low] = temp;

    return partition(arr, low, high);
}
```

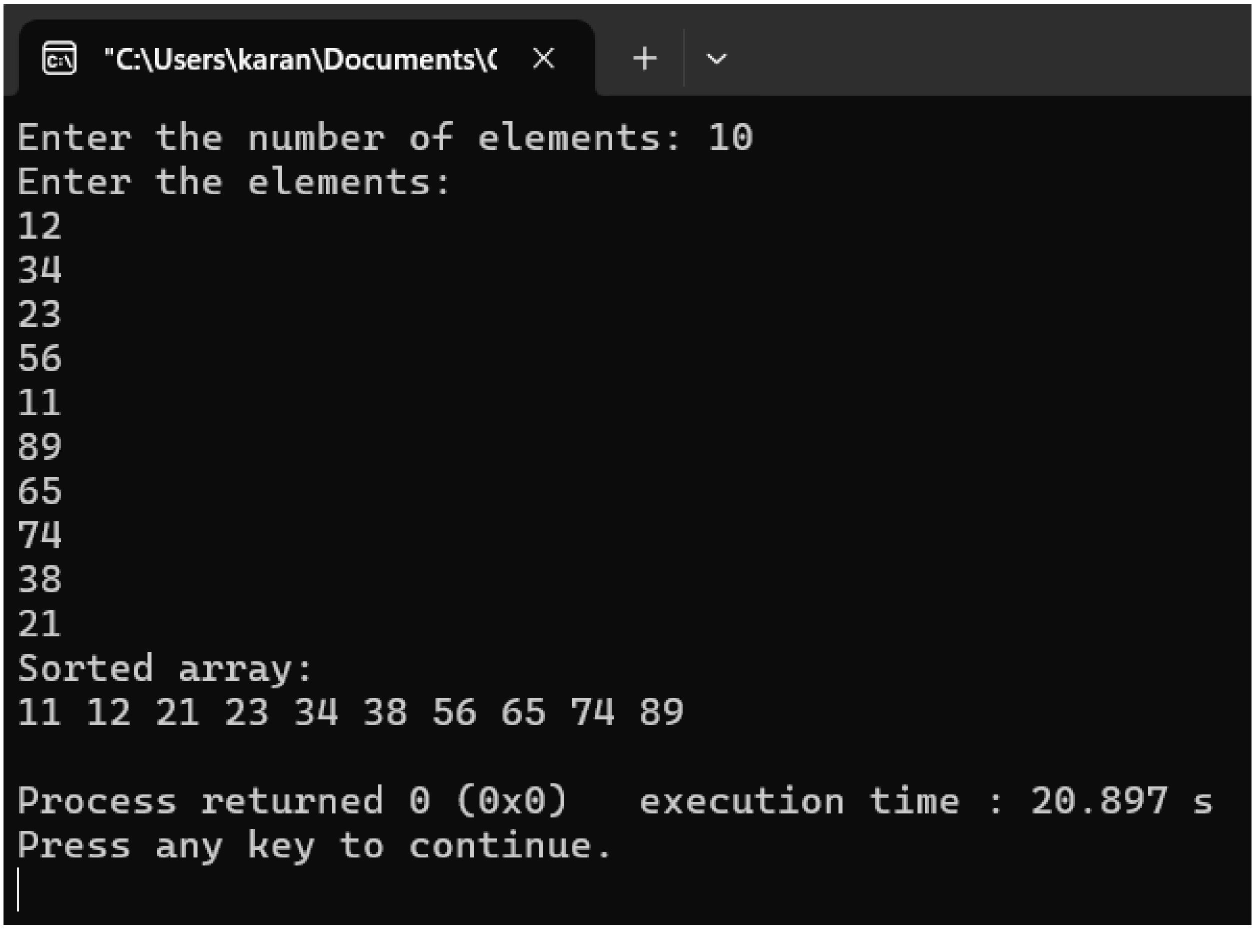


```c
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition_r(arr, low, high);

        quickSort(arr, low, pi);
        quickSort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int *arr = (int *)malloc(n * sizeof(int));

    if (arr == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    printf("Enter the elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    quickSort(arr, 0, n - 1);

    printf("Sorted array: \n");
    printArray(arr, n);

    free(arr);
    return 0;
}
```

Output:



```
Enter the number of elements: 5
Enter the elements:
3
45
12
7
8
Sorted array:
3 7 8 12 45

Process returned 0 (0x0)   execution time : 20.307 s
Press any key to continue.
```



```
Enter the number of elements: 10
Enter the elements:
12
34
23
56
11
89
65
74
38
21
Sorted array:
11 12 21 23 34 38 56 65 74 89

Process returned 0 (0x0)   execution time : 20.897 s
Press any key to continue.
```