Durmuş Gülbahar
20180808076

ConvexHull Problem solved with **QuickHull** Algorithm.
Source → https://github.com/durmusgulbahar/convexhull.git

Convex Hull definition: Given a set of points in a 2D space, we have to find the convex hull of those points. (shape that includes all of the points that in the given set.)

Real world applications:
Collision Meshes In computer graphics.

QuickHull Algorithm definition: A method of computing the convex hull of a finite set of points in n-dimensional space. It uses a divide and conquer method.
Worst Case = $O(n^2)$
Best Case = O(nlogn)

There are two main methods:
1. quickHull(set)

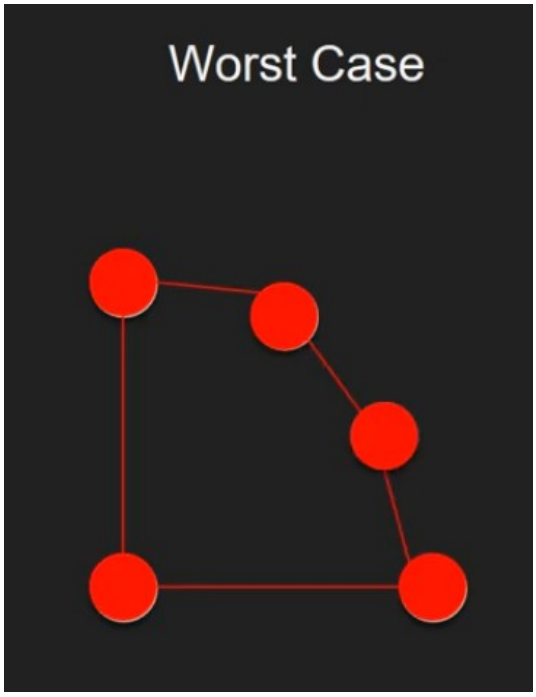
1. find left and right most points furthest each other A&B
2. groups the points that stay right and left of the A-B line
3. call findHull(set, A,B) method
4. call findhull(set,B,A) method
5. return convexPoints

2. hullSet(A,B,set,convexPoints),

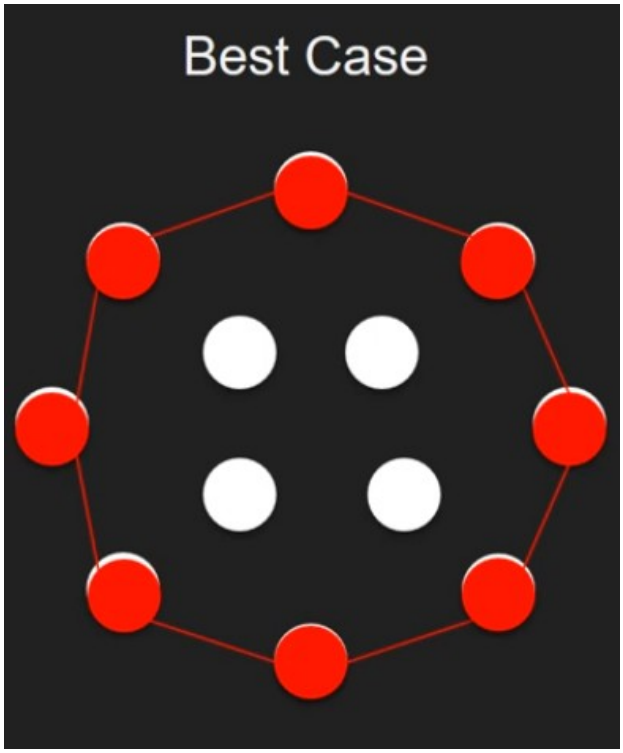
1.if set no point return
2.find furthest point from A – B say it C
3.groups points as outside of the A – C and outside of the C – B, call hullSet(A,C,leftAC,convexPoints) and hullset(C,B, leftCB, convexPoints) itself. Recursive part.

Time Complexity

Worst Case scenario is algorithm visits each point in the set. Thus complexity become **$O(n^2)$**



Best Case scenario is algorithm groups each segmentation evenly and number of points of convex hull is smaller than points that stayed inline. In this scenario time complexity becomes **O(nlogn)**



Implementation

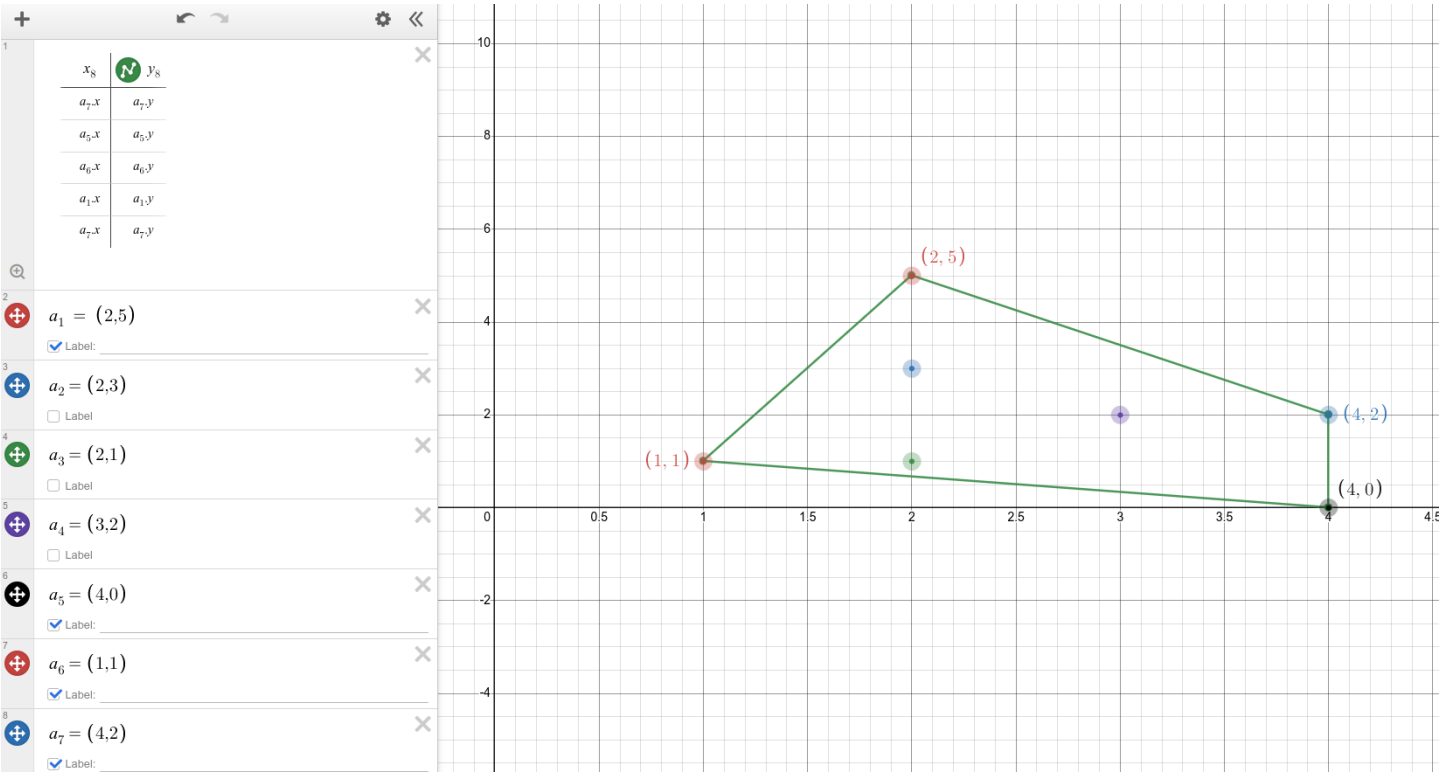
```
Run | Debug
public static void main(String[] args) {
    System.out.println(x:"Quick Hull Convex Hull Example");
    ArrayList<Point> points = new ArrayList<Point>();
    // Add points here
    // Result should be outer points. (1,1), (2,5), (4,2), (4,0)
    points.add(new Point(x:2, y:5));
    points.add(new Point(x:2,y:3));
    points.add(new Point(x:2, y:1));
    points.add(new Point(x:3, y:2));
    points.add(new Point(x:4, y:0));
    points.add(new Point(x:1, y:1));
    points.add(new Point(x:4, y:2));
    QuickHull qh = new QuickHull();
    ArrayList<Point> p = qh.quickHull(points);
    for (int i = 0; i < p.size(); i++)
        System.out.println("(" + p.get(i).x + ", " + p.get(i).y + ")");

}
```

Main

```
Quick Hull Convex Hull Example
(1, 1)
(2, 5)
(4, 2)
(4, 0)
```

Output



Graph

Implementation - Helpers

```java
/*
 * Point class has (x,y)
 */
public static class Point {
    int x, y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Point Object

```java
/*
 * Returns the points that are in the convex hull.
 * If point is in the triange(A-B-P) then returns 1.
 * If point is outside of the line(A-B-P) then returns -1.
 */
public int pointLocation(Point A, Point B, Point P) {
    int cp1 = (B.x - A.x) * (P.y - A.y) - (B.y - A.y) * (P.x - A.x);
    if (cp1 > 0)
        return 1;
    else if (cp1 == 0)
        return 0;
    else
        return -1;
}
```

PointLocation method

```java
/*
* Returns distance between A-B points and C point.
*/

public int distance(Point A, Point B, Point C) {
    int ABx = B.x - A.x;
    int ABy = B.y - A.y;
    int num = ABx * (A.y - C.y) - ABy * (A.x - C.x);
    if (num < 0)
        num = -num;
    return num;
}
```

Distance method

```java
public ArrayList<Point> quickHull(ArrayList<Point> points) {
    ArrayList<Point> convexHull = new ArrayList<Point>();
    if (points.size() < 3)
        return (ArrayList) points.clone();

    int minPoint = -1, maxPoint = -1;
    int minX = Integer.MAX_VALUE;
    int maxX = Integer.MIN_VALUE;
    for (int i = 0; i < points.size(); i++) {
        if (points.get(i).x < minX) {
            minX = points.get(i).x;
            minPoint = i;
        }
        if (points.get(i).x > maxX) {
            maxX = points.get(i).x;
            maxPoint = i;
        }
    }
    Point A = points.get(minPoint);
    Point B = points.get(maxPoint);
    convexHull.add(A);
    convexHull.add(B);
    points.remove(A);
    points.remove(B);

    ArrayList<Point> leftSet = new ArrayList<Point>();
    ArrayList<Point> rightSet = new ArrayList<Point>();

    for (int i = 0; i < points.size(); i++) {
        Point p = points.get(i);
        if (pointLocation(A, B, p) == -1)
            leftSet.add(p);
        else if (pointLocation(A, B, p) == 1)
            rightSet.add(p);
    }


    hullSet(A, B, rightSet, convexHull); // clockwise area of the A-B
    hullSet(B, A, leftSet, convexHull);  // counter clockwise area of the A-B

    return convexHull;
}
```

QuickHull method

```java
/*
 * Recursive part of the QuickHull Algorithm. It takes two point that min and max,
 *  left/right set and convexHull Points to update
 * if there is no elm in set means A and B consist all of the points
 * if there is one element in the set that means there is a only one
 * point outside of the line A-B and this point replaced with B
 *
 * if set has more than one elements, we find furthest distance and we
 * are gonna create new triangle with A-B-C and method call
 * itself again on this triangle and points that left and right side of A-C line.
 *
 */
public void hullSet(Point A, Point B, ArrayList<Point> set, ArrayList<Point> hull) {
    int insertPosition = hull.indexOf(B);
    if (set.size() == 0)
        return;

    if (set.size() == 1) {
        Point p = set.get(index:0);
        set.remove(p);
        hull.add(insertPosition, p);
        return;
    }
    int dist = Integer.MIN_VALUE;
    int furthestPoint = -1;
    for (int i = 0; i < set.size(); i++) {
        Point p = set.get(i);
        int distance = distance(A, B, p);
        if (distance > dist) {
            dist = distance;
            furthestPoint = i;
        }
    }
    Point P = set.get(furthestPoint);
    set.remove(furthestPoint);
    hull.add(insertPosition, P);

    // Determine who's to the left of A-P
    ArrayList<Point> leftSetAP = new ArrayList<Point>();
    for (int i = 0; i < set.size(); i++) {
        Point M = set.get(i);
        if (pointLocation(A, P, M) == 1) {
            //set.remove(M);
            leftSetAP.add(M);
        }
    }

    // Determine who's to the left of P-B
    ArrayList<Point> leftSetPB = new ArrayList<Point>();
    for (int i = 0; i < set.size(); i++) {
        Point M = set.get(i);
        if (pointLocation(P, B, M) == 1) {
            //set.remove(M);
            leftSetPB.add(M);
        }
    }
    hullSet(A, P, leftSetAP, hull);
    hullSet(P, B, leftSetPB, hull);

}
```

References:

https://www.youtube.com/watch?v=2EKIZrimeuk

https://www.youtube.com/watch?v=5D9F1HA6-f4

https://en.wikipedia.org/wiki/Quickhull

https://medium.com/smith-hcv/convex-hull-algorithms-in-2d-976803538452

https://en.wikipedia.org/wiki/Convex_hull_algorithms

https://www.desmos.com/calculator