WARSAW UNIVERSITY OF TECHNOLOGY

Advanced algorithms

Functional Documentation

Debelo Milkiyas Duguma,
Swetha Suresh Kumar

November 8, 2022

Contents

1 Problem

Project Topic: GOLF

The golf field has n balls, n holes, and n Golfers. Each Golfer has to put one ball in one of the holes simultaneously. For this purpose, they determine which ball is aimed at which hole so that the tracks of their ball do not intersect.

Let us assume that the balls and holes are points on the plane and that none three of them are co-linear and that the balls move in a straight line. Present an algorithm running in time $O(n^2 \log n)$ which assigns the balls to the holes so that no two tracks of the balls intersect.

2 Solution

2.1 Description

We used the 'Divide and conquer' algorithm to solve the problem. For a given set, we search for a balanced hit, i.e. one from the left and/or right side (side doesn't matter since we are given an equal n number of $\mathbf{B}$ and $\mathbf{H}$, so there's exactly the same number of balls and holes). Then we call this one, a separate algorithm for balls and holes on the left side and separately on the right side.

For empty sets of balls and holes, calling the algorithm will return an empty set. By invoking the algorithm for one ball and one hole, we return a hit consisting of this ball and this hole. All strikes found by this algorithm constitute the solution to the problem.

In order to find a balanced hit, we apply an algorithm with a similar scheme to Graham's scan (reference attached) algorithm for finding a convex envelope of points in two-dimensional space:

1. We search for a point with the lowest y-coordinate 'L'. If the lowest y-coordinate exists in more than one point, we choose the point with the lowest x-coordinate /abscissa/ out of the candidates. And we shift the origin of the coordinate system to this point (we shift all points by the vector).
2. We sort the remaining points by the angle between the OP vector from the beginning of the system to this point and the axis OX. Since any three points in both sets are not co-linear, any point will have a different value of this angle, except for the point L - for this, we can enter Lexicographic sort with distance from O as the second predicate.
3. Each point, depending on whether it is a ball or a hole, is assigned a numerical value: for the ball it is 1, and for the hole it is -1. We initialize the auxiliary variable with zero value and then go through the sorted list of points, adding the value of the current point to this variable.

   At the moment when the auxiliary variable reaches the value 0 again, the algorithm ends its operation, and the last considered point with point O creates the impact that meets the mentioned condition.

After finding a balanced hit and finding solutions to the smaller tasks to the left and right of this hit, as a result we return the set theory sum of the solution for the points on the left, the one-element set containing the found balanced hit, and the solution for the points on the right.

2.2 Pseudocode

2.2.1 General algorithm pseudocode

Entry inputs: a set of $\mathbf{B}$ balls and $\mathbf{H}$ holes

function FINDPLANARMATCHING($\mathbf{B}$, $\mathbf{H}$)

if $|\mathbf{B}| = |\mathbf{H}| = 0$ **then**
return $\emptyset$
end if

if $|\mathbf{B}| = |\mathbf{H}| = 1$ **then**
$B \coloneqq$ point from $\mathbf{B}$
$H \coloneqq$ point from $\mathbf{H}$
return $\{\}$
end if

$LS \coloneqq$ FINDBALANCEDMATCH($\mathbf{B}$, $\mathbf{H}$)

$\mathbf{B}_1 \coloneqq$ points from $\mathbf{B}$ to the left of $\overrightarrow{LS}$
$\mathbf{H}_1 \coloneqq$ points from $\mathbf{H}$ to the left of $\overrightarrow{LS}$
$\mathbf{B}_2 \coloneqq$ points from $\mathbf{B}$ to the right of $\overrightarrow{LS}$
$\mathbf{H}_2 \coloneqq$ points from $\mathbf{H}$ to the right of $\overrightarrow{LS}$

$\mathbf{M}_1 \coloneqq$ FINDPLANARMATCHING($\mathbf{B}_1$, $\mathbf{H}_1$)
$\mathbf{M}_2 \coloneqq$ FINDPLANARMATCHING($\mathbf{B}_2$, $\mathbf{H}_2$)
return $\mathbf{M}_1 \cup \{LS\} \cup \mathbf{M}_2$

end function

2.2.2 The pseudocode of the balanced hit algorithm

Entry inputs: a set of $\mathbf{B}$ balls and $\mathbf{H}$ holes

function FINDBALANCEDMATCH($\mathbf{B}$, $\mathbf{H}$)

$\mathbf{P} \coloneqq \mathbf{B} \cup \mathbf{H}$
$O \coloneqq$ origin of the coordinate system
$L \coloneqq$ leftmost of the lowest points of $\mathbf{P}$

move all points from $\mathbf{P}$ to vector LO
lexicographically sort $\mathbf{P}$ on the angle between the vector OP and the axis OX and the distance from $\mathbf{O}$

balance $\coloneqq 0$
for all $P \in \mathbf{P}$ **do**
if $P \in \mathbf{B}$ **then**
balance $+= 1$
else
balance $-= 1$
end if
if balance $= 0$ **then**
return
end if
end for

end function

3 Solution for a sample task

Let us assume that we have two equal sets and each of the elements of the set has a certain point on a plane:

- Set of balls $\mathbf{B} = \{B1, B2, B3, B4\}$
- Set of holes $\mathbf{H} = \{H1, H2, H3, H4\}$

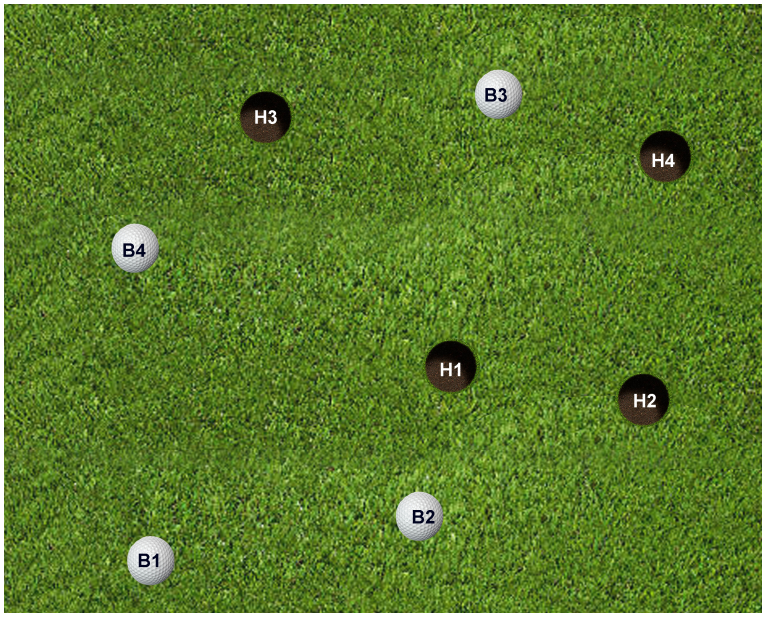


Figure 1: Initial arrangement of holes and balls.

The first step is to select the point that has the smallest Y coordinate value, which is B1. Then, after shifting the center of the coordinate system in its place, we sort the remaining points as it is in the description of the algorithm.

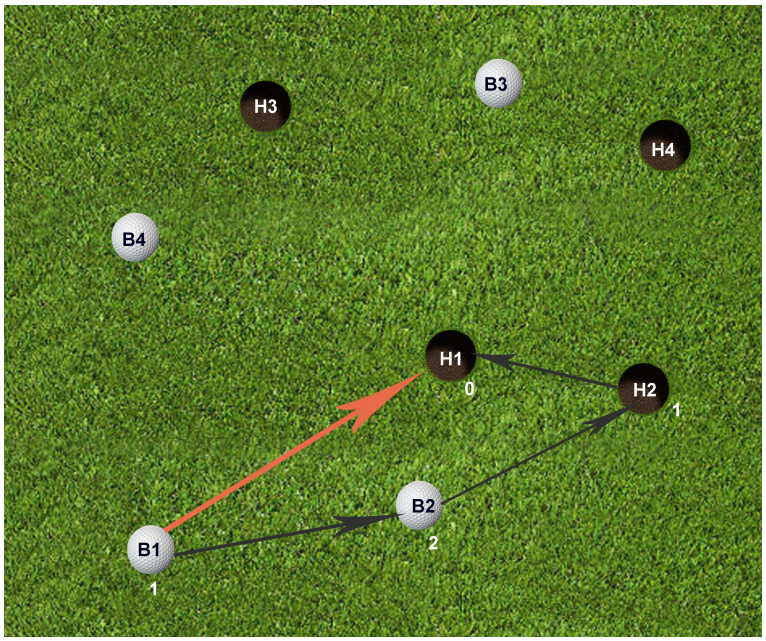


Figure 2: Finding the first plane dividing point

Then we go through the sorted points, respectively, increasing or decreasing the balance value. When the H1 point is reached, the balance value is 0, therefore we remember the hit B1, H1, and we can omit some of these points in further searches.

We qualify the points that have been found until the point that divides the plane is found to a separate sub-problem (Group 1), as well as points not yet visited (Group 2)

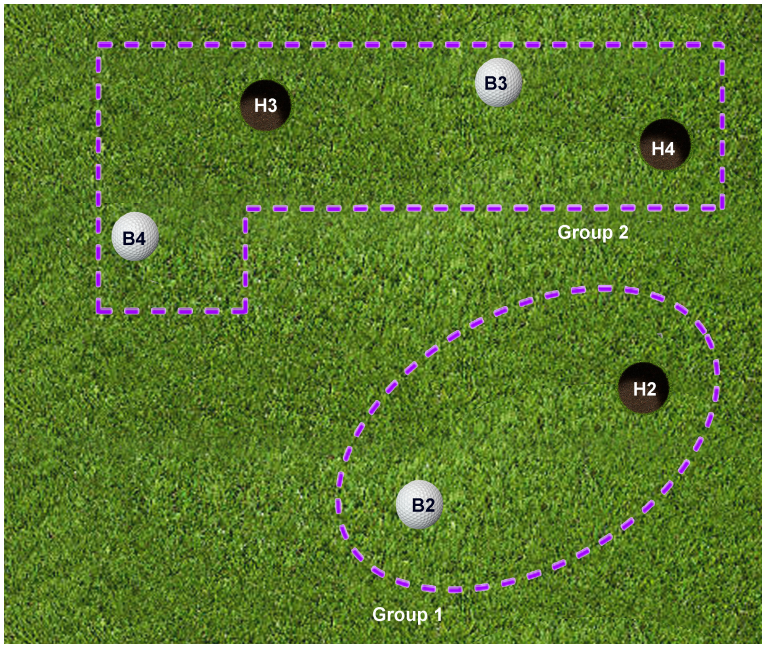
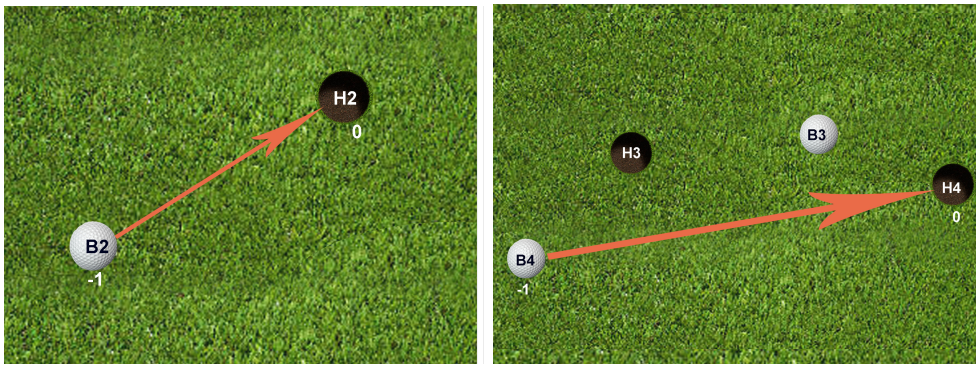


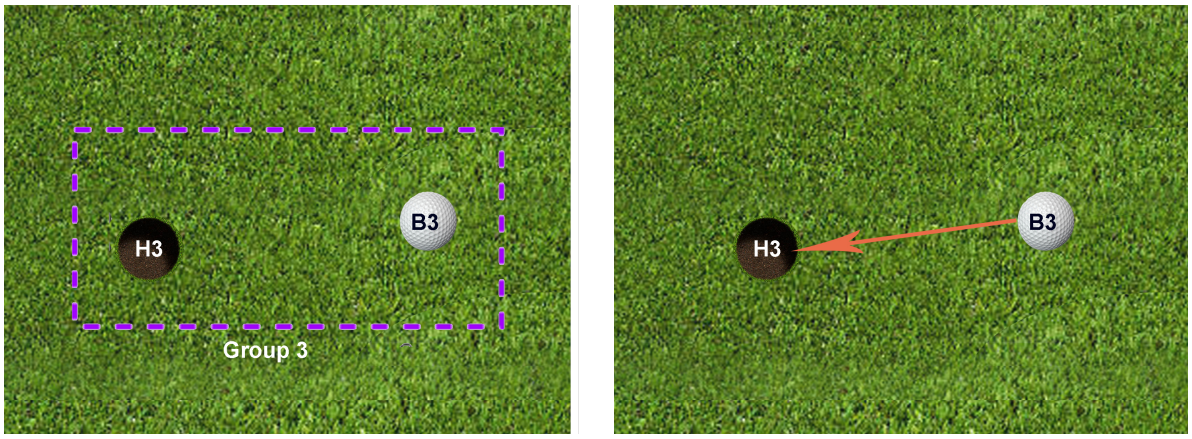
Figure 3: Division into subproblems

Proceeding analogously in the resulting sub-problems, we find the next pairs of points, which we add to the resulting set of strikes.



(a) Group 1 (b) Group 2

Figure 4: Solving sub-problems



(a) Create another sub-problem. (b) Group 3.

Figure 5: Solving sub-problems

When there are no unresolved sub-problems left (all points are paired), we can return the final set of strokes $\mathbf{U} = \{\{B1, H1\}, \{B2, H2\}, \{B3, H3\}, \{B4, H4\}\}$ resulting from the algorithm.

4 Input Output Description

4.1 Input

The input file has the following information:

- the number of balls or the number of holes, ie n in the first line;
- ball coordinates separated by commas and each ball is described on a new line;
- the coordinates of the holes are separated by commas, each hole is described on a new line ;

Example:

```
4
0.1,0.1
0.8,0.2
1.0,1.8
0.0,1.4
0.9,0.8
1.5,0.7
0.5,1.7
1.6,1.6
```

4.2 Output

The output file has the following information:

- the index of the ball in the input file and the index of the hole in the input file that make up the hit, separated by a space; each hit is described on a new line;

Example:

```
1,1
2,2
3,3
4,4
```

5 Proof of Correctness of the Solution

Lemma 1. *Let $\mathbf{B}$ be a set of balls, and $\mathbf{H}$ a set of holes, and let $|\mathbf{B}| = |\mathbf{H}|$. If LS is a hit with the same number of balls on its left (right) side as holes, then the match is a set-theoretical sum of:*

- *planar matching $\mathbf{M}_1$ balls to the left of LS with holes to the left of LS*
- *one-element set $\{LS\}$*
- *planar match $\mathbf{M}_2$ of balls to the right of LS with holes to the right of LS is planar*

Proof. For the case of $n = 1$, the thesis is obvious.

Note that if satisfies the assumption on the above proposition, then from equality of $\mathbf{B}$ and $\mathbf{H}$ it follows that: if there are as many balls on one side of the strike as there are holes, the other side also has the same number of balls as well as holes (because in total there must also be the same number of balls as holes).

Let $\mathbf{B}_1$ and $\mathbf{H}_1$ be sets of balls and holes to the left of LS, respectively, and let $\mathbf{B}_2$ and $\mathbf{H}_2$ be sets of balls and holes to the left of LS. Since $|\mathbf{B}_1| = |\mathbf{H}_1|$ and $|\mathbf{B}_2| = |\mathbf{H}_2|$, are the pairs $\mathbf{B}_1$ and $\mathbf{H}_1$ as well as $\mathbf{B}_2$ and $\mathbf{H}_2$ are also tasks belonging to the set of tasks for the problem under consideration.

Assuming that $\mathbf{M}_1$ is planar, we know that for any pair of hits with $\mathbf{M}_1$, they have no common point. Additionally, since $\mathbf{M}_1$ is the set of ball hits to the left from LS to holes to the left of LS, then any strike of $\mathbf{M}_1$ has no common point with LS. Likewise, any pair of hits from $\mathbf{M}_2$ has no common point and no hits from $\mathbf{M}_2$ has no point in common with LS.

As it is easy to notice, in the set $\mathbf{M}_1 \cup \{LS\} \cup \mathbf{M}_2$ there are only strikes. In addition, any pair of strikes in this set has no common point and these strikes (hits) cover the set of points $\mathbf{B} \cup \mathbf{H}$. From this it follows that $\mathbf{M}_1 \cup \{LS\} \cup \mathbf{M}_2$ is a planar fit.

□

Theorem 1. *The algorithm for finding a balanced hit finds such a match for any finite and equal sets of $\mathbf{B}$ balls and $\mathbf{H}$ holes.*

Proof. For $n = 0$ and $n = 1$, the proof results directly from the definition of balanced hits.

The initial operations are to prepare the points to execute the right part of the algorithm. The resulting properties will help us to prove main loop invariants. Let's assume that:

1. we found the lowest, leftmost point L among all the balls and holes,
2. we moved the origin of the O coordinate system to the point L
3. we sorted the points from $\mathbf{P}$ on the angle between the vector OP and the OX axis and the distances from O

The last point does not require sorting all points along a distance from O, because we assume no three points z are not collinear. It only forces the point $O = L$ to be first in sequence.

In order to prove the correctness of our algorithm, we will show that some invariants hold main loop of our algorithm

Invariant 1. *The algorithm will not finish in the first step of the loop.*

Proof. Quite obvious - in this step the `balance` variable will be -1 or 1, so the algorithm will not break the loop and continue

□

Invariant 2. *The algorithm will end in one of the steps of the loop.*

Proof. The `balance` variable is initialized to zero before entering the loop. Then, at each step of the loop it is increased by one (when the point considered in the iteration is a ball) or decreased by one (when the point considered in the iteration is a hole).

If the algorithm did not end in one of the loops, it would mean that the variable `balance` was not set to zero in any of the steps in the loop, especially the last one. Note, however, that after the last step of the loop we have considered all the balls and all the holes of the same number. This means that we added exactly as much to the `balance` variable as we subtracted from it. After the loop is executed, it should be equal to its initial value, i.e. 0. Contradiction - the algorithm will end at the last step of the loop at the latest.

□

Invariant 3. *In the k-th step of the loop (where $k \geq 3$) to the right (clockwise) from of the OP segment, there are points considered in steps from the second to the $k-1$st.*

Proof. The points are sorted ascending by the angle between the vector OP and the axis OX. Therefore, for different points P_i and P_j, for which the angles are α_i and α_j, respectively, if $i < j$, then $\alpha_i < \alpha_j$.

Therefore, the angle between the vector OP_{ji} andOP_j is $\alpha_i - \alpha_j < 0$, therefore point P_j is on clockwise from the vector OP_i

□

Invariant 4. *The `balance` variable after executing the kth step of the loop (except for the first step of the loop) for the point P indicates the difference between the number of balls and the number of holes in the half-plane to the right of the line containing the vector OP,*

Proof (inductive). After the first step of the loop, the variable `balance` is -1, when it is lowest, most to the left, the point is a hole, or 1 if that point is a ball. It is not important in terms of invariant 1 stating that the algorithm will not end in the first step.

1. For $k = 2$ there are no points to the right of the vector OP. The `balance` variable is then a sum values for the first and second points. It can be:
   (a) 2, when both points are balls,
   (b) 0, when one of the points is a ball and the other is a hole,
   (c) -2, when both points are holes.
2. Let us assume that for some $k < 2n$ the invariant thesis is satisfied. Then consider the step $(k+1)$ -th, in which we consider point P.

   By invariant 3, to the right of the vector OP are points considered in steps from the second to the k-th. On the other hand, on the basis of the inductive assumption, the variable `balance` before performing the $(k+1)$ -th step indicates the difference between the number of balls and the number of holes for the points considered in the previous steps of the algorithm.

   The variable balance will increase by 1 when the considered point P is a ball, or decrease by 1 when it is a hole. Likewise, the difference between the number of balls and holes will also increase by 1 when the P considered point is a ball, or decrease by 1 when it is a hole.

Therefore, after performing the $(k+1)$-th step, `balance` will still show the difference between the number of balls and the number of holes for the points considered in steps from the first to $(k+1)$-th.

□

Invariant 5. *The algorithm finds the path which is the hit.*

Proof. Let us assume that the segment found by the algorithm is not a hit. Therefore, in the first and last step of the algorithm, the `balance` value either increased or decreased. To establish the remark, let us assume that in these steps balls were considered and the value of the variable `balance` was increased (the second case is symmetrical).

After the first step, the value of the variable `balance` was 1. On the other hand, if the variable `balance` was 0 after the last step, it had to be -1 before it (in this step we considered the ball). Therefore, in steps from the second to the penultimate, the value of `balance` changed from 1 to -1.

Since in one step the `balance` variable can be smaller by a maximum of 1, then after a certain step between the second and the penultimate variable the value of the variable `balance` was 0. Contradiction - the algorithm should end when the variable `balance` reaches zero for the first time

□

Based on these invariants, we know that the algorithm will end in one of the steps (except the first). We also know that after this step the variable `balance`, which indicates the difference between the number of balls considered so far and the number of holes considered so far, is 0. We also know that all the points considered so far lie to the right or on the straight line passing through the vector OP, where P is the currently considered point.

From all this it follows that among the points to the right of the vector OP there are exactly the same number of balls as holes, and also the segment OP connects the ball with the hole. Therefore, the OP hit found is a balanced hit.

□

Theorem 2. *The general algorithm for the given problem is correct.*

Proof. By theorem 1 for any input, it is possible to find a balanced hit, on the left (and right) side there are the same number of balls as holes. From the proof lemma 1, we can divide this problem into two subtasks by finding matches of balls to holes on the left and right sides of the balanced shot.

Due to the fact that $\mathbf{B}_1 \subsetneq \mathbf{B}$ and $\mathbf{H}_1 \subsetneq \mathbf{H}$ as well as $\mathbf{B}_2 \subsetneq \mathbf{B}$ and $\mathbf{H}_2 \subsetneq \mathbf{H}$, the size of the subtask decreases with each recursion. For $n = 0$ or $n = 1$ we come to the bottom of the recursion and there is a return. By virtue of the lemma 1, this strategy allows you to find the correct planar fit of balls to holes at any level of recursion, including the very top of it.

□

6 Proof of algorithm complexity

Lemma 2. *The algorithm for finding the balanced hit has a complex order of at most* $O(n \log n)$.

Proof. In order to analyze the complexity, let us consider the individual operations performed in the algorithm.

1. connection of sets of points can be realized in linear time (assuming that $|\mathbf{B}| = |\mathbf{H}|$); the task is all the more simple because in the input conditions we demand that these points were pairs miscellaneous - $O(n)$;

2. finding the minimum point, assuming the unit cost of comparison, will be required a linear traversal of the entire collection of points -$O(n)$;

3. the translation of each point by a constant vector also has a linear time complexity (we shift each point in the collection) -$O(n)$;

4. the comparison of two points P_1 and P_2 on the basis of the angle between the vector OP_i and the axis OX and the distance from O can be done in constant time (e.g. using the monotonic trigonometric function on the interval $[0, \pi]$) ;
   under this assumption, for the quicksort method with pivot selection with the magic fives algorithm, the sorting of points will have the superseline time complex - $O(n \log n)$;

5. inside the loop, two comparisons will be made, one assigned and at most one return from functions; because the loop in a pessimistic case runs over the entire collection of points, the complexity of this operation is linear $O(n)$

Summing up this short analysis, we can see that the most costly operation is to sort all the points. Therefore, the time complexity of the whole algorithm for finding the balanced hit is super-linear, i.e. it is $O(n \log n)$. □

Lemma 3. *In the general algorithm for solving the problem, a function that performs the general algorithm for sets of* 0 *or* 1 *will be called* n *times, and for sets of* 2 *to* n *—* $n - 1$ *times.*

Proof. Each call to the encapsulation function of the general problem-solving algorithm is reduced to two possible situations:

1. return an empty or single-element set (recursion stop condition) - type I execution

2. he function that finds the balanced shot is called, which then determines the division into smaller (according to the "divide and conquer" strategy), for which this function - type II execution;

We can notice that in a recursion tree the first calls will form leaves, and the latter calls will form parents in the tree. Remember that the recursion tree is a 2-regular tree, i.e. each node has zero or two children.

Since the balanced hit does not go to any subtask for which we recursively execute the function, each subtask is smaller than the task it originated from, by at least 1. Also, the total size of the subtasks is 1 smaller than the size of the task. which has been divided (it results directly from the method of division). Recursion stops for jobs of size 0 or 1.

On the basis of the above properties, it can be seen that the divisions under the "divide and conquer" strategy, i.e. the execution of a type II function (parents in the recursion tree) is $n-1$, while the execution of a type I function (list in recursion tree) is 1 more, i.e. n. □

Theorem 3. *The general algorithm for solving the problem has the order of* $O(n^2 \log n)$.

Proof. Based on the Lemma 3 general functions for an input of size 0 or 1 will be called n times, while the others - $n - 1$ times.

General functions for input of size 0 or 1 execute at a constant time - return empty set or select the only elements from sets and return a single-element set containing the only possible hit .

For general function calls for an input larger than 1, the most expensive operation is to find the balanced hit - it takes $O(n \log n)$ time. Checking whether a point is positioned clockwise or counterclockwise about a vector can be done with a simple vector product over constant time. The division of the remaining

points into the clockwise and counterclockwise sides can then be done in linear time. Likewise, merging disjoint sets into one also requires a linear investment of time. All this leads to the conclusion that the type II call of a general function has the time complexity $O(n \log n)$).

Therefore, the entire algorithm is executed in $(n-1) \cdot O(n \log n) + n \cdot O(1)$. It is equivalent to the algorithm of time complexity $O(n^2 \log n)$.

□