Presentation writer: Tigran Hayrapetyan

Lecturer | Programmer | Researcher

www.linkedin.com/in/tigran-hayrapetyan-cs/

Convex hull
(QuickHull)

Prerequisites:

- *recursion.*

QuickHull

The divide an conquer algorithm is analogous to Merge sort, used for sorting a sequence.

Can we find another convex hull algorithm, which will be analogous to QuickSort?

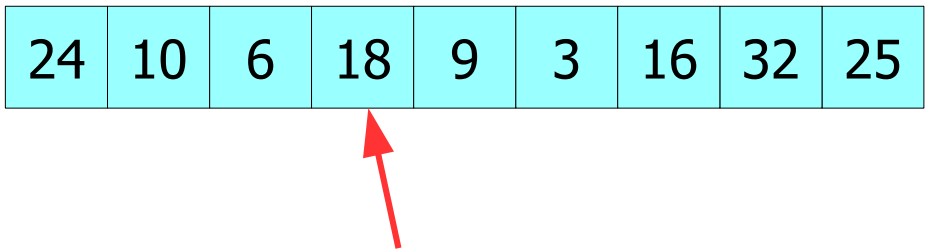
QuickHull

24	10	6	18	9	3	16	32	25

Let's recall how QuickSort works:

- given a sequence which must be sorted,

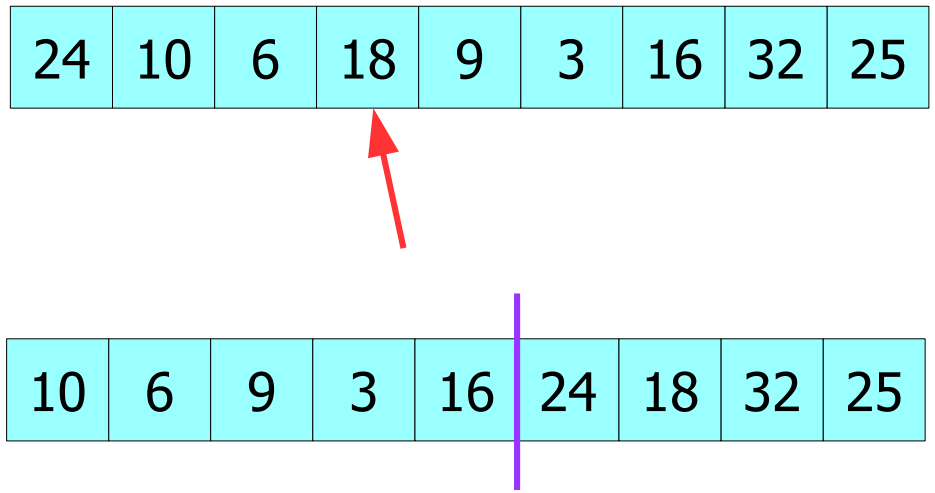
QuickHull



Let's recall how QuickSort works:

- given a sequence which must be sorted,
- we pick a pivot value from it,

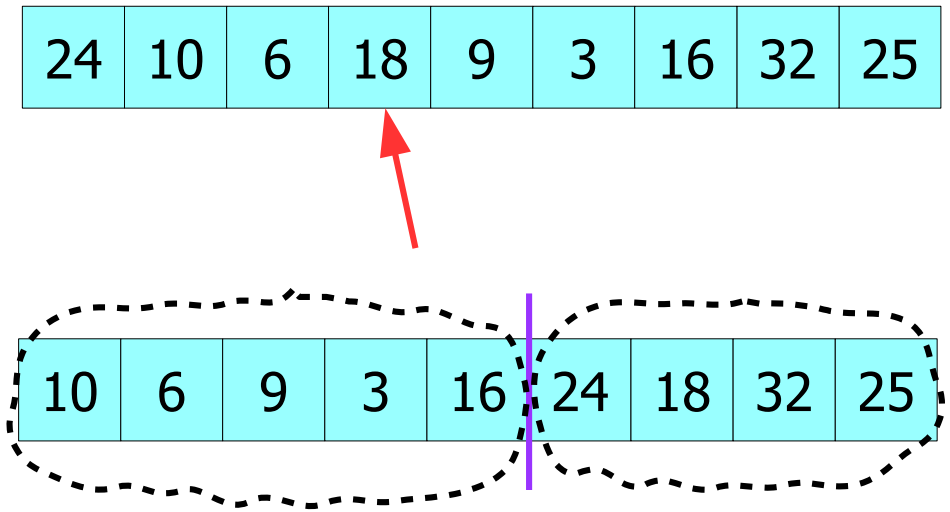
QuickHull



Let's recall how QuickSort works:

- given a sequence which must be sorted,
- we pick a pivot value from it,
- rearrange it in **2** parts, upon pivot,

QuickHull



Let's recall how QuickSort works:

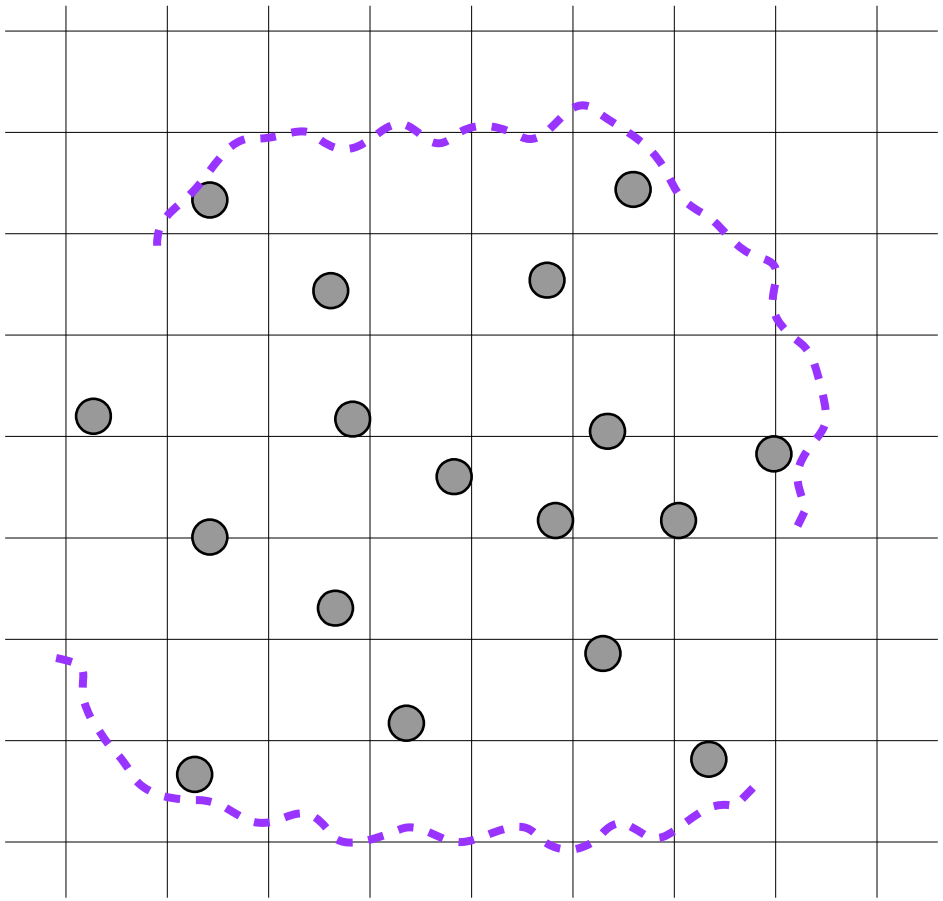
- given a sequence which must be sorted,
- we pick a pivot value from it,
- rearrange it in **2** parts, upon pivot,
- and recursively sort the **2** parts in independent way.

QuickHull

Can we figure out a similar scheme,

... where the work will be split into parts, and recursive calls will be made at the very end?

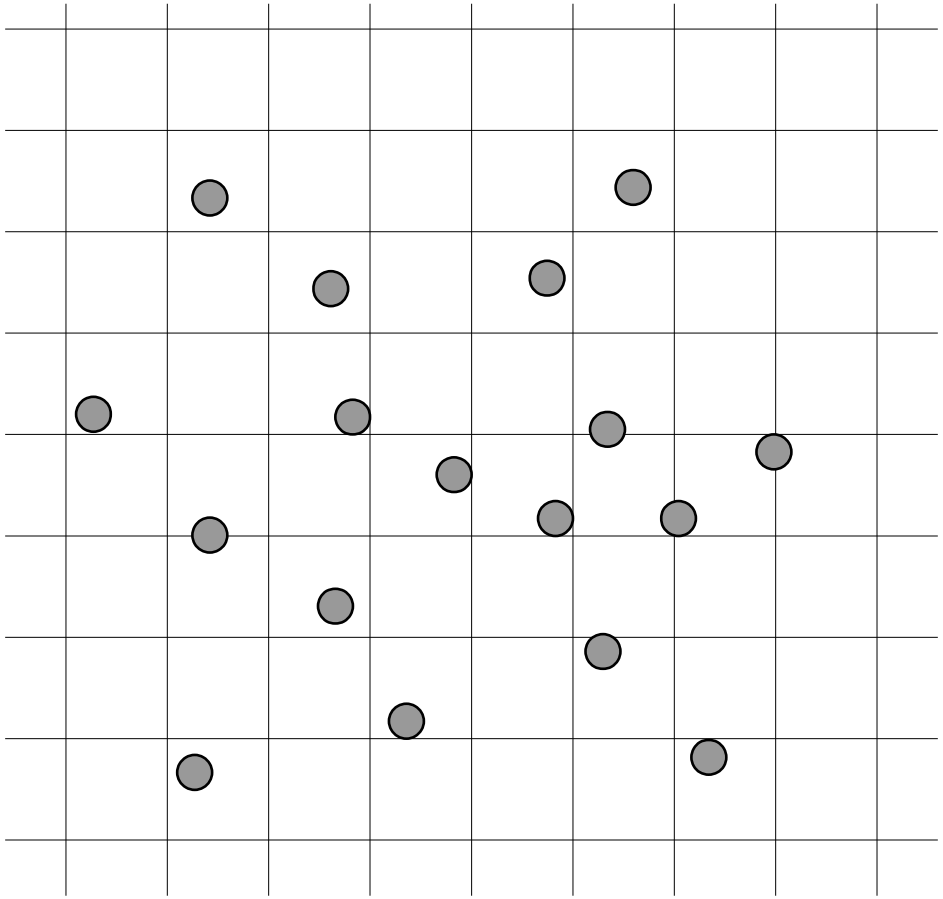
QuickHull



In order to do that, we must split the work in such a way...

...that it can be continued in **2** directions independently.

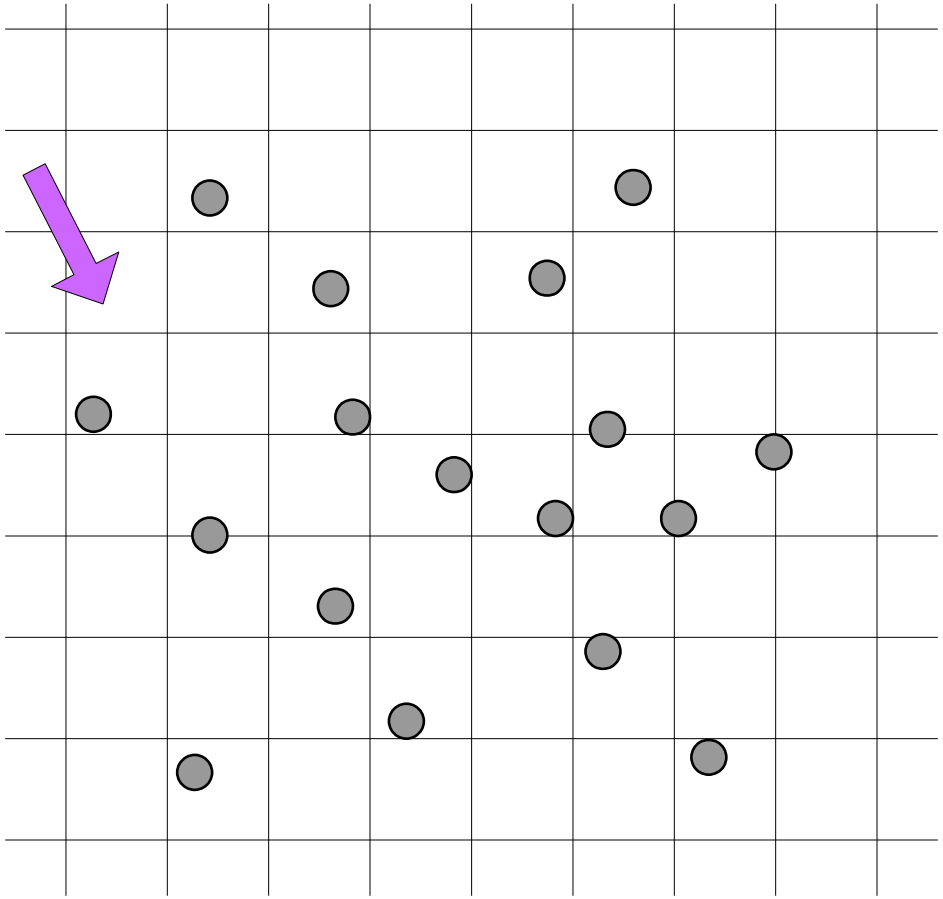
QuickHull
(farthest point)



Before moving forward, let's note an important property:

- having a set of points,

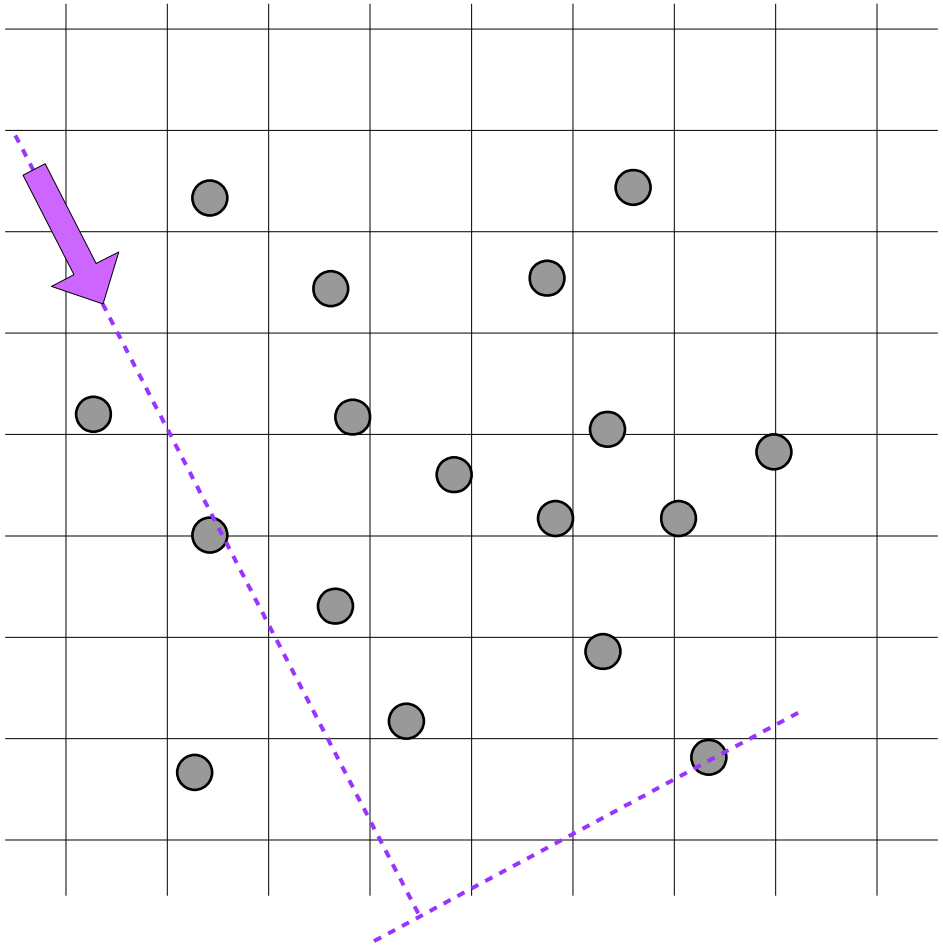
QuickHull
(farthest point)



Before moving forward, let's note an important property:

- having a set of points,
- and an arbitrary direction,

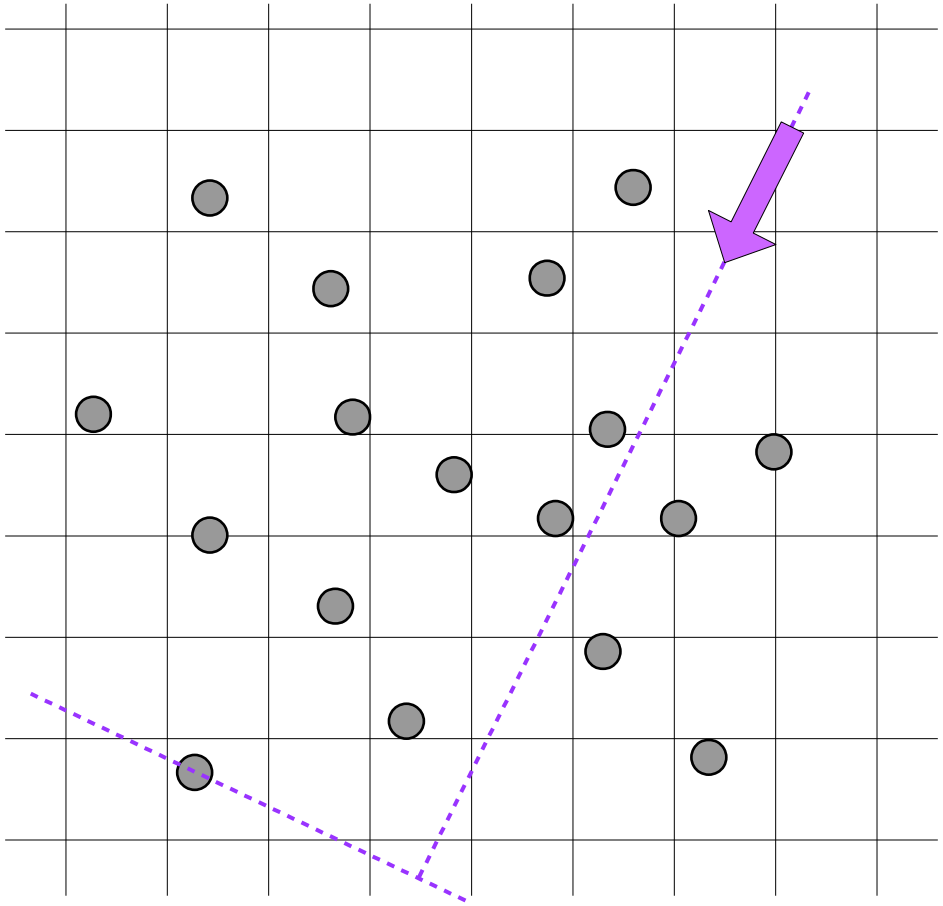
QuickHull
(farthest point)



Before moving forward, let's note an important property:

- having a set of points,
- and an arbitrary direction,
- The point which is farthest upon that direction will necessarily participate in the hull.

QuickHull
(farthest point)

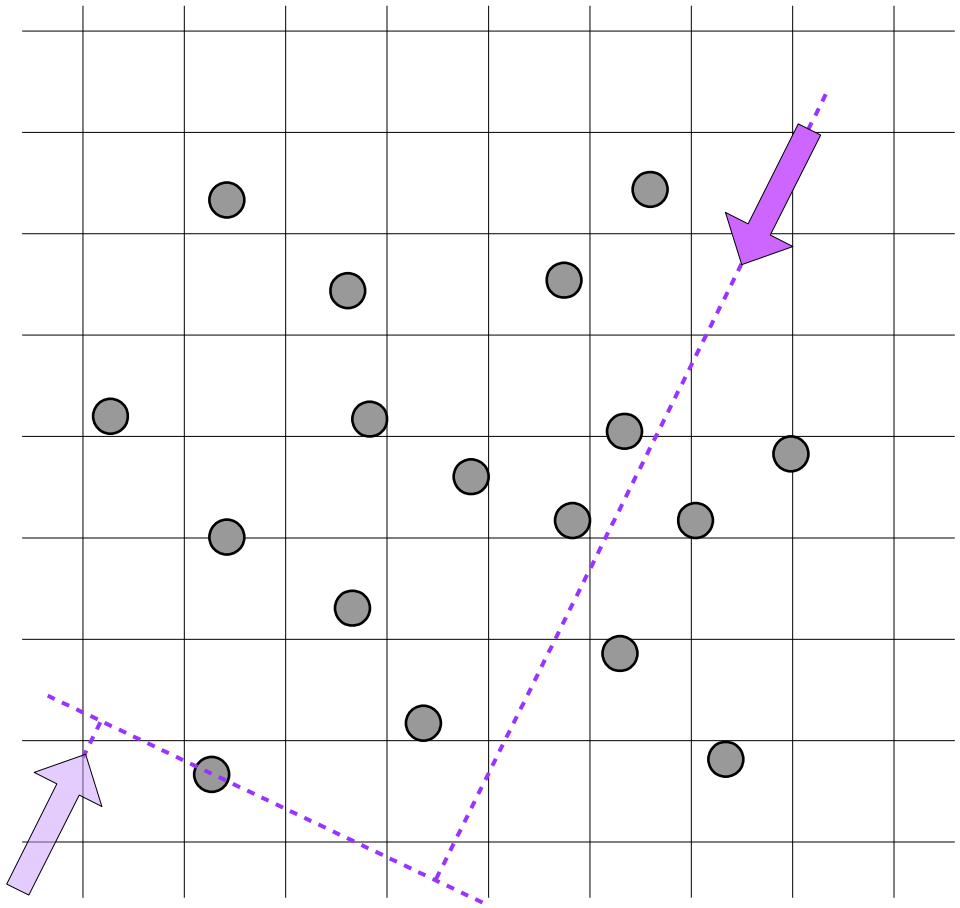


Before moving forward, let's note an important property:

- having a set of points,
- and an arbitrary direction,
- The point which is farthest upon that direction will necessarily participate in the hull.

... one more example.

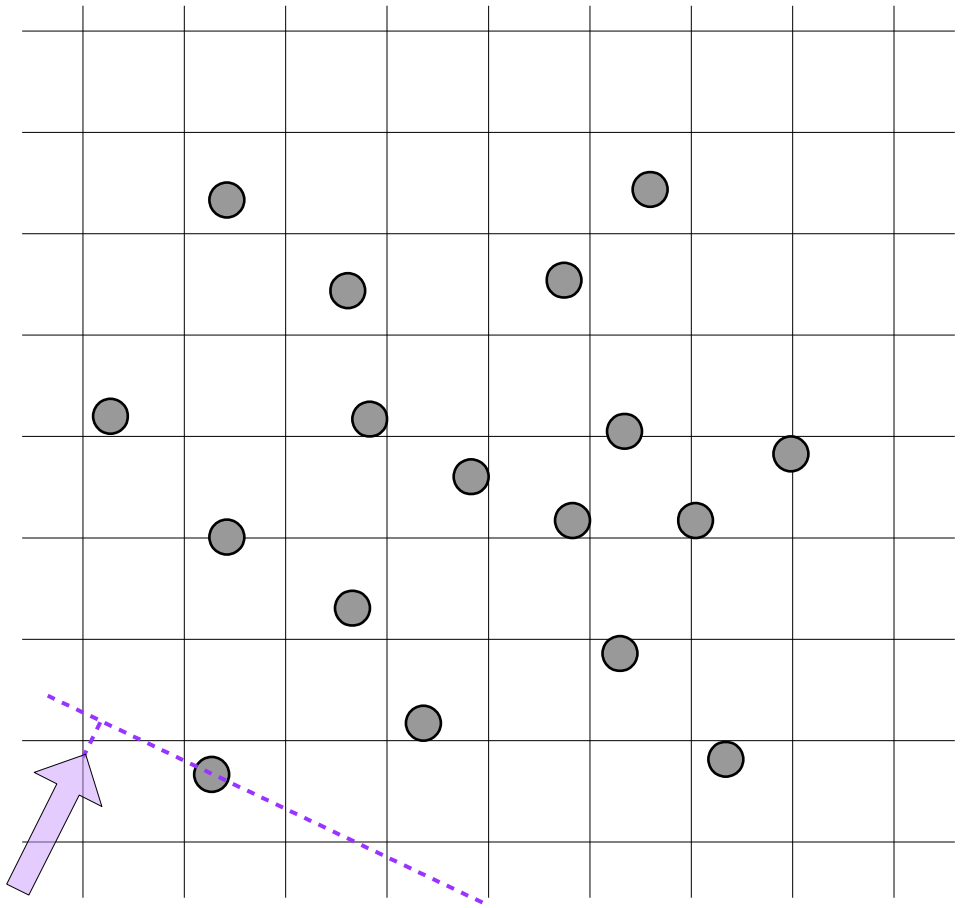
QuickHull
(farthest point)



Why is it that way?

- the point which is farthest upon given direction is also the closest upon the opposite direction,

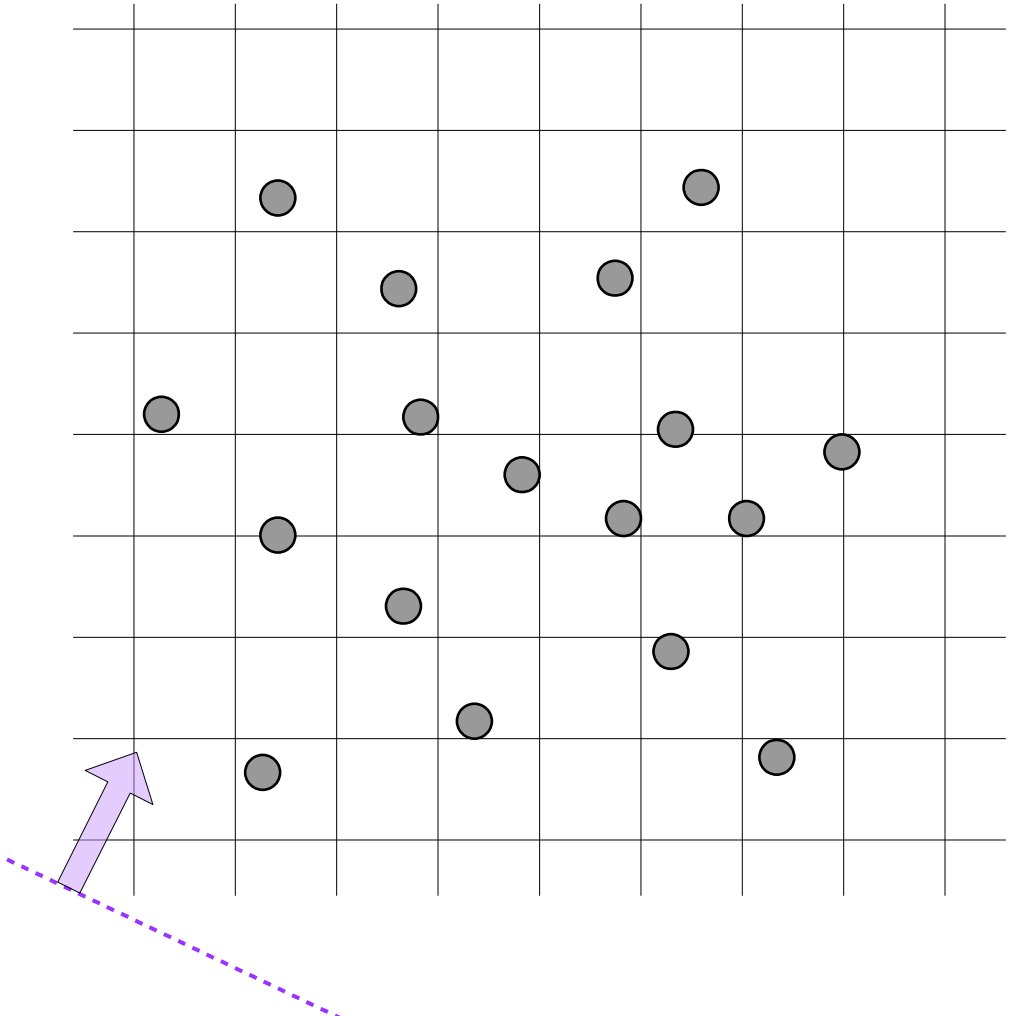
QuickHull
(farthest point)



Why is it that way?

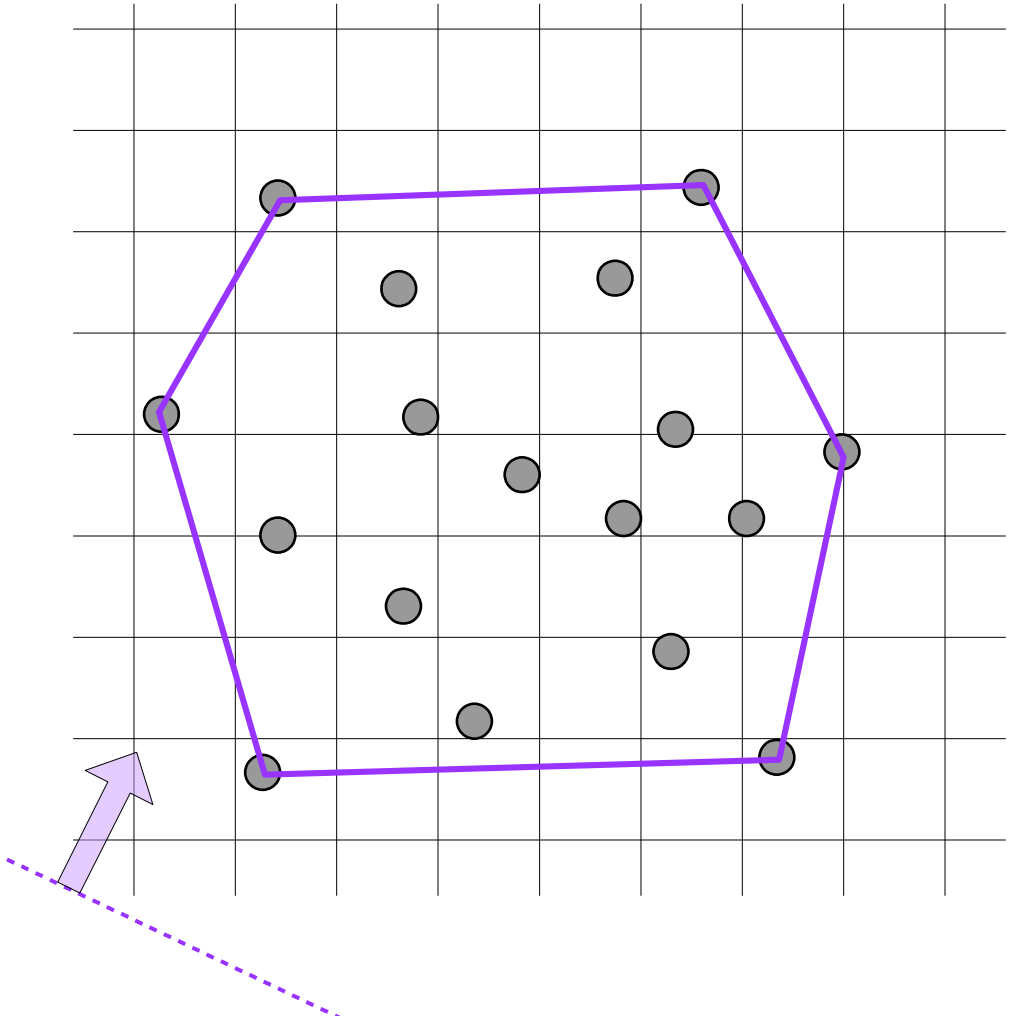
- the point which is farthest upon given direction is also the closest upon the opposite direction,
- so the question becomes – why the closest point upon any direction will participate in convex hull?

QuickHull
(farthest point)



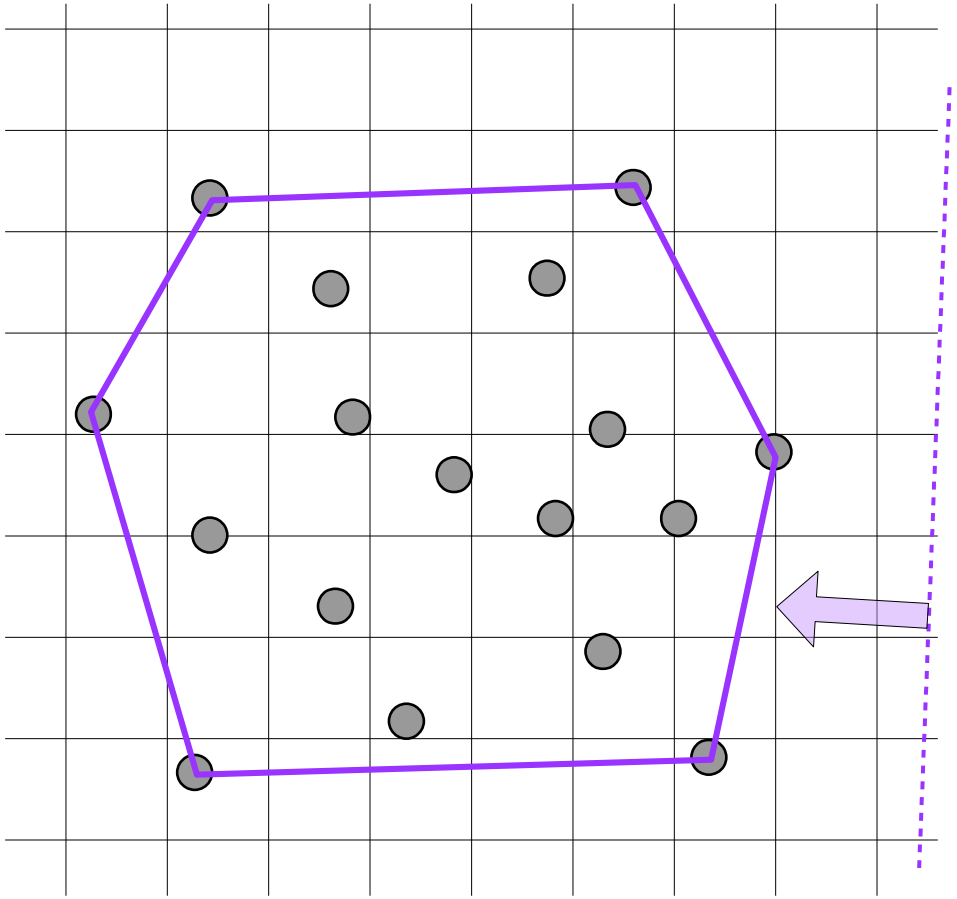
- Let's assume a line, coming by the direction,
- It will first hit the closest point, upon that direction,

QuickHull
(farthest point)



- Let's assume a line, coming by the direction,
- It will first hit the closest point, upon that direction,
- But first hit means also that we are hitting the convex hull,

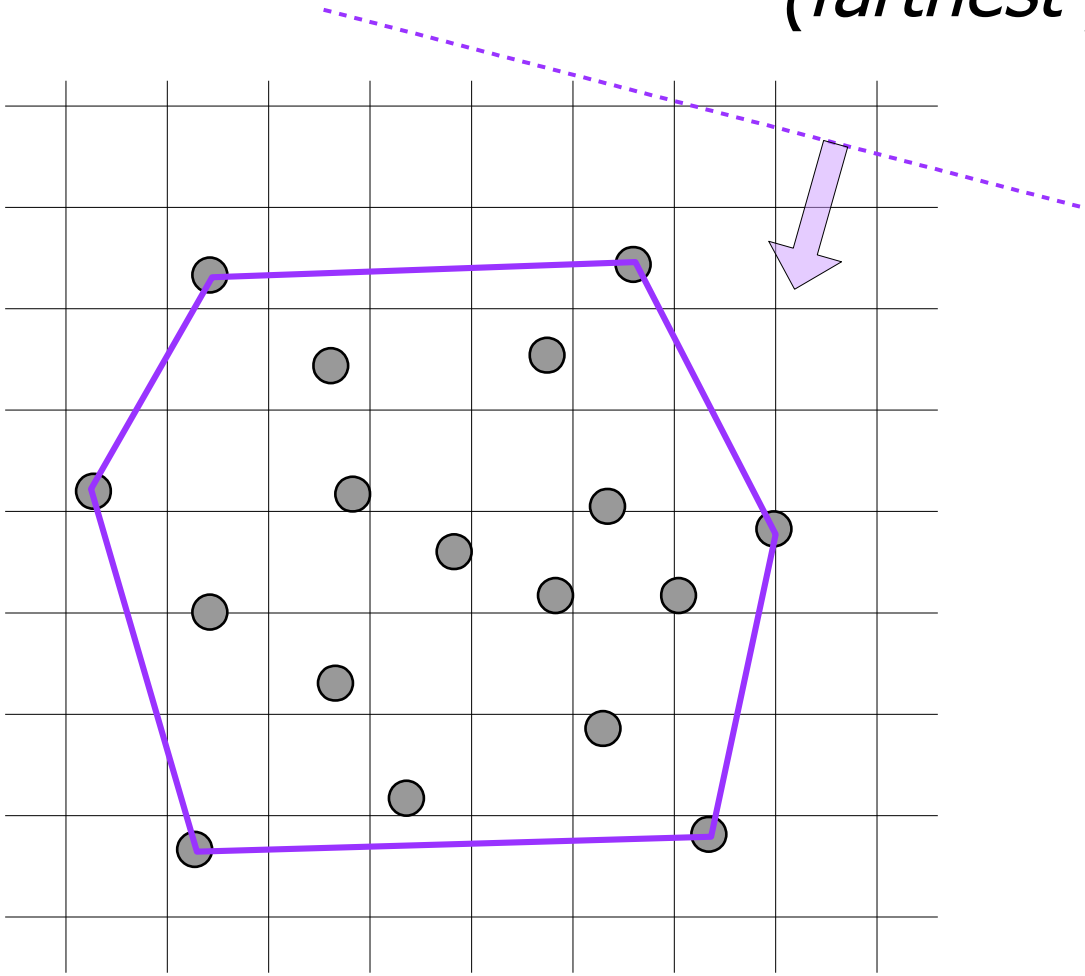
QuickHull
(farthest point)



- Let's assume a line, coming by the direction,
- It will first hit the closest point, upon that direction,
- But first hit means also that we are hitting the convex hull,

Another example...

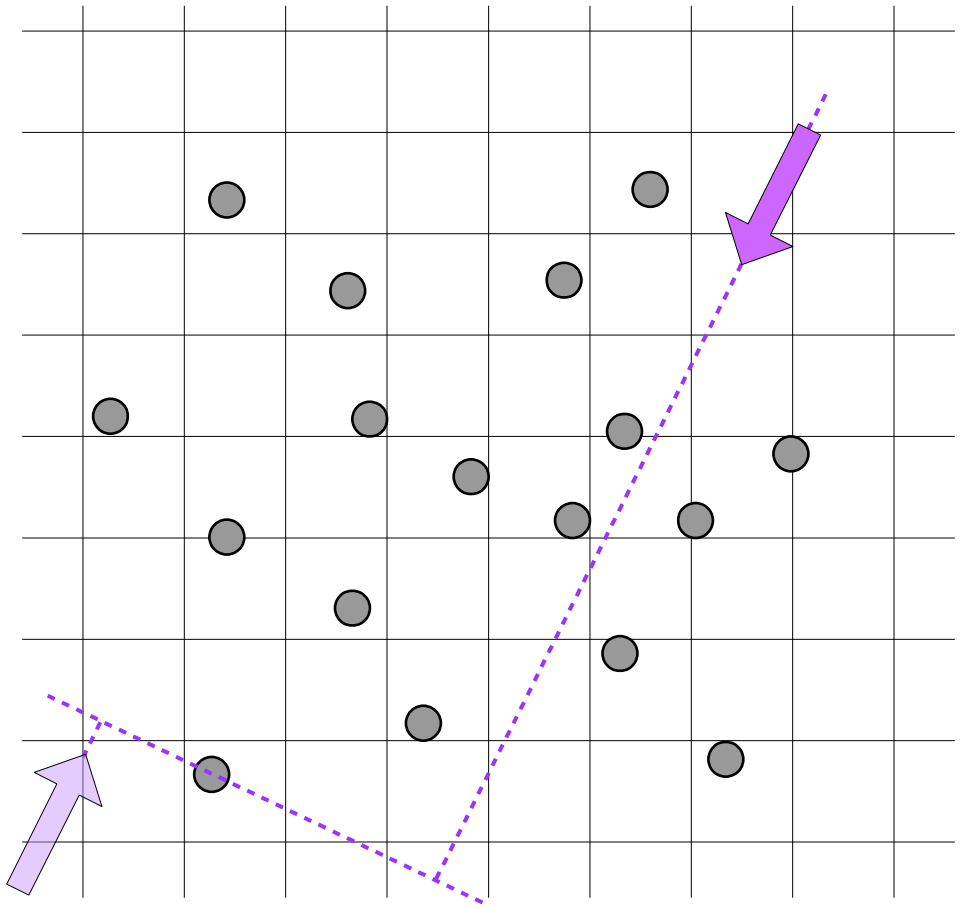
QuickHull
(farthest point)



- Let's assume a line, coming by the direction,
- It will first hit the closest point, upon that direction,
- But first hit means also that we are hitting the convex hull,

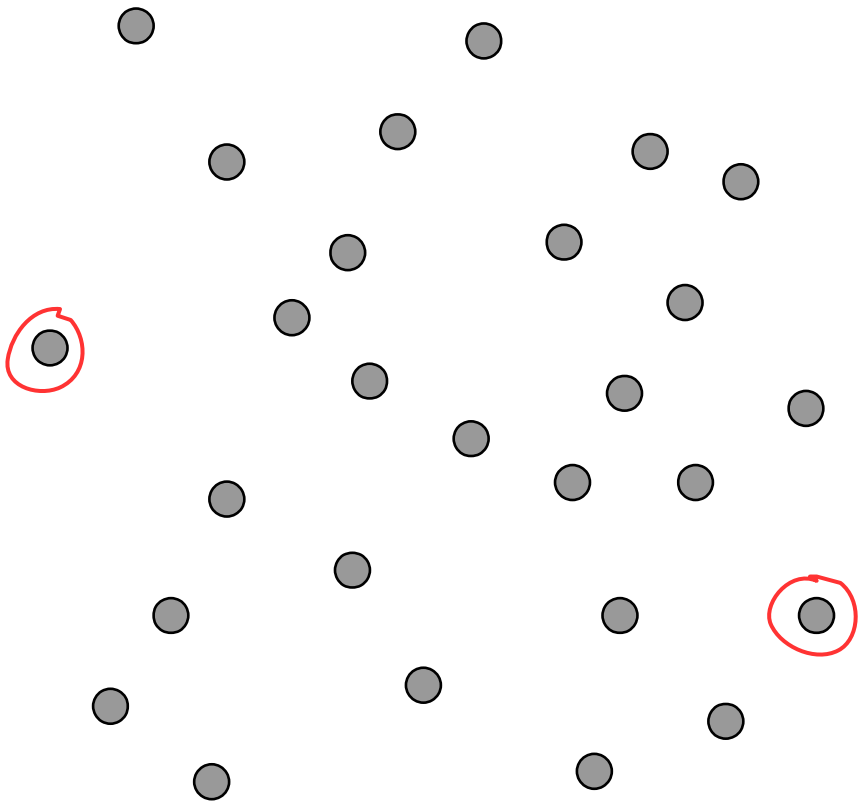
One more example...

QuickHull
(farthest point)



- Which is why the closest point upon any direction will participate in convex hull,
- As well as the farthest point will.

QuickHull

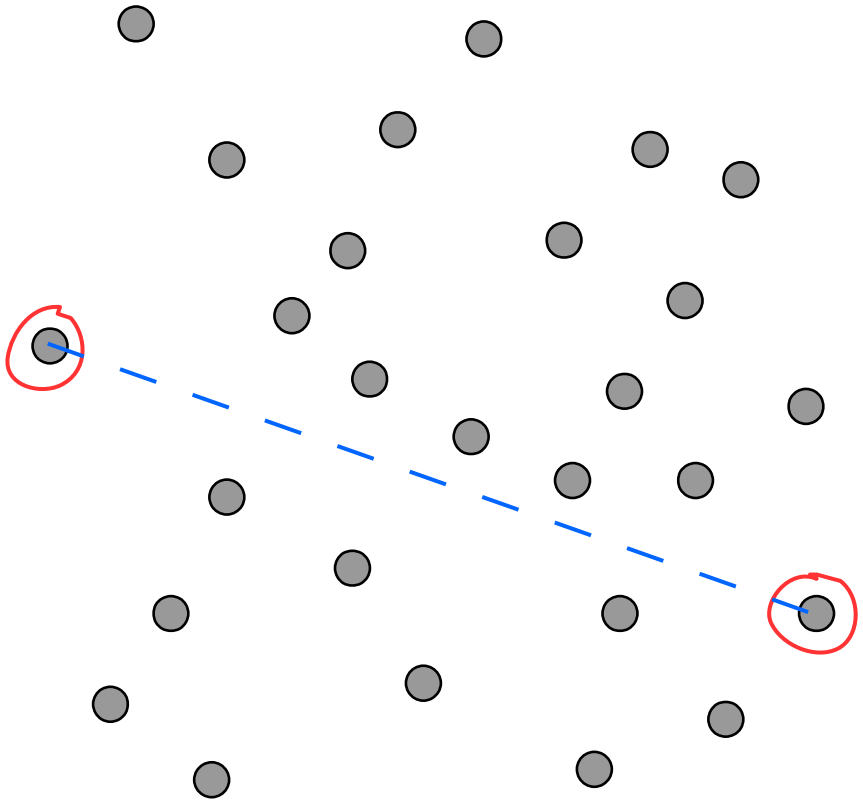


Now let's address the algorithm:

At first we will find leftmost and rightmost points,

... those **2** will participate in the hull.

QuickHull



Now let's address the algorithm:

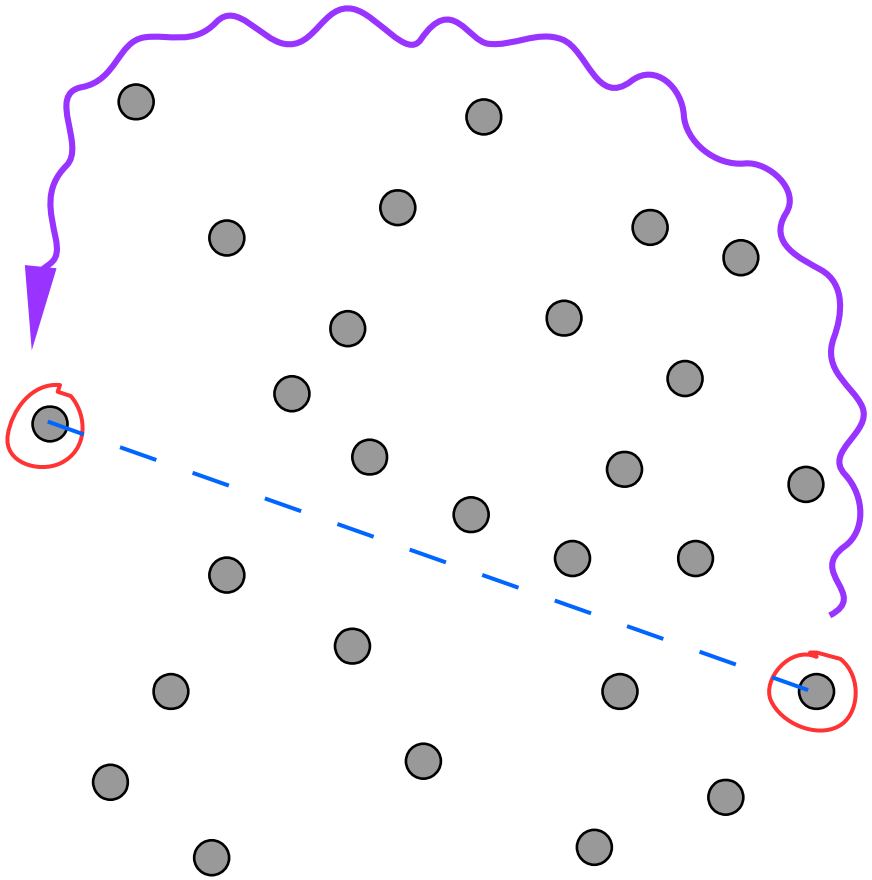
At first we will find leftmost and rightmost points,

... those **2** will participate in the hull.

Let's connect them by a segment.

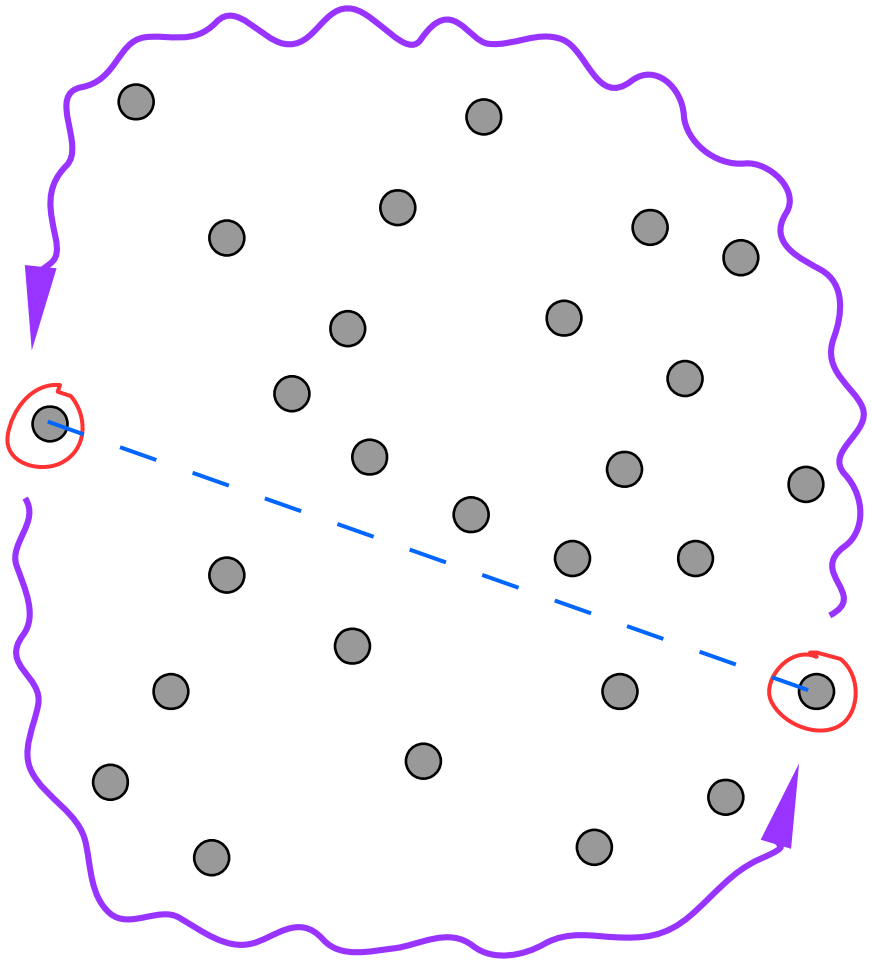
... this will result in **2** disjoint sets of points.

QuickHull



From upper set we will construct one half of the convex hull,

QuickHull



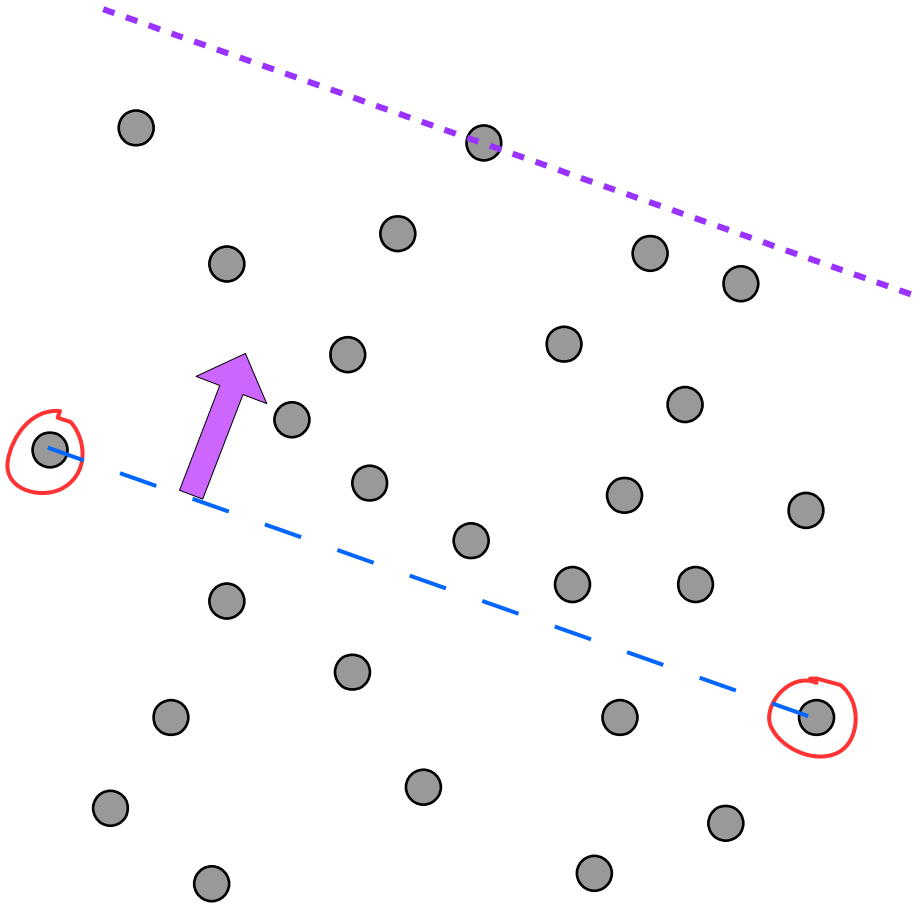
From upper set we will construct one half of the convex hull,

From lower set we will construct the other half.

Note, we already managed to split the task in two parts.

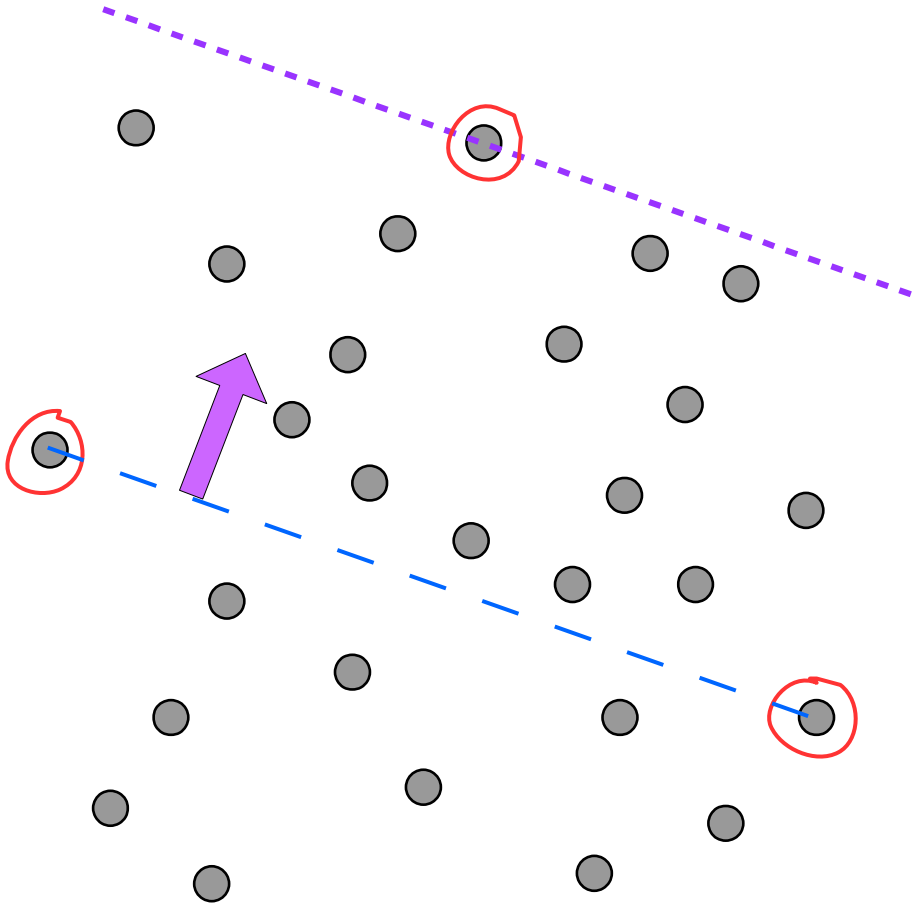
... this subtasks can even run in parallel.

QuickHull



Next, considering current subset, let's find the farthest point upon direction of the perpendicular,

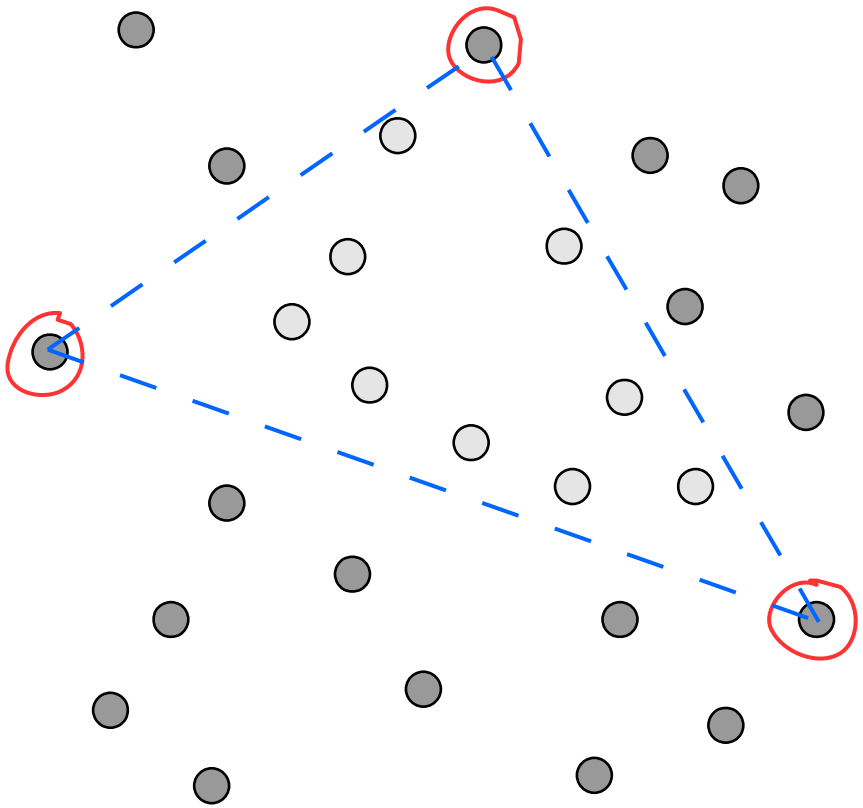
QuickHull



Next, considering current subset, let's find the farthest point upon direction of the perpendicular,

... recall, it will definitely participate in convex hull.

QuickHull



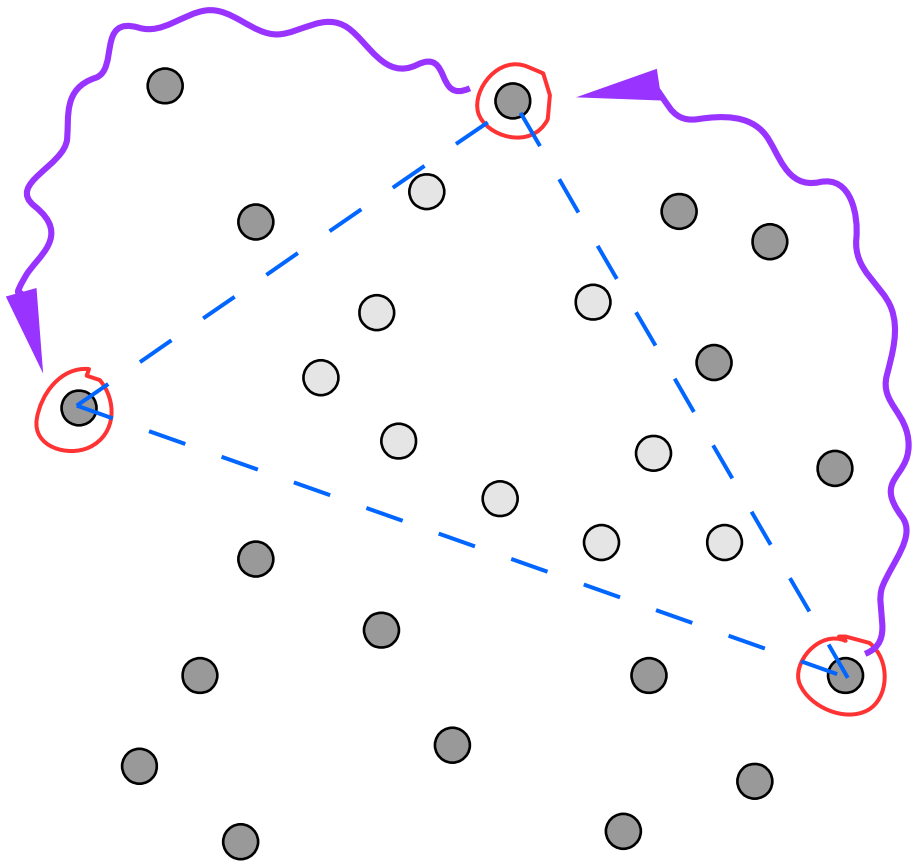
Next, considering current subset, let's find the farthest point upon direction of the perpendicular,

... recall, it will definitely participate in convex hull.

Now we have a triangle, so we can safely discard all the points inside,

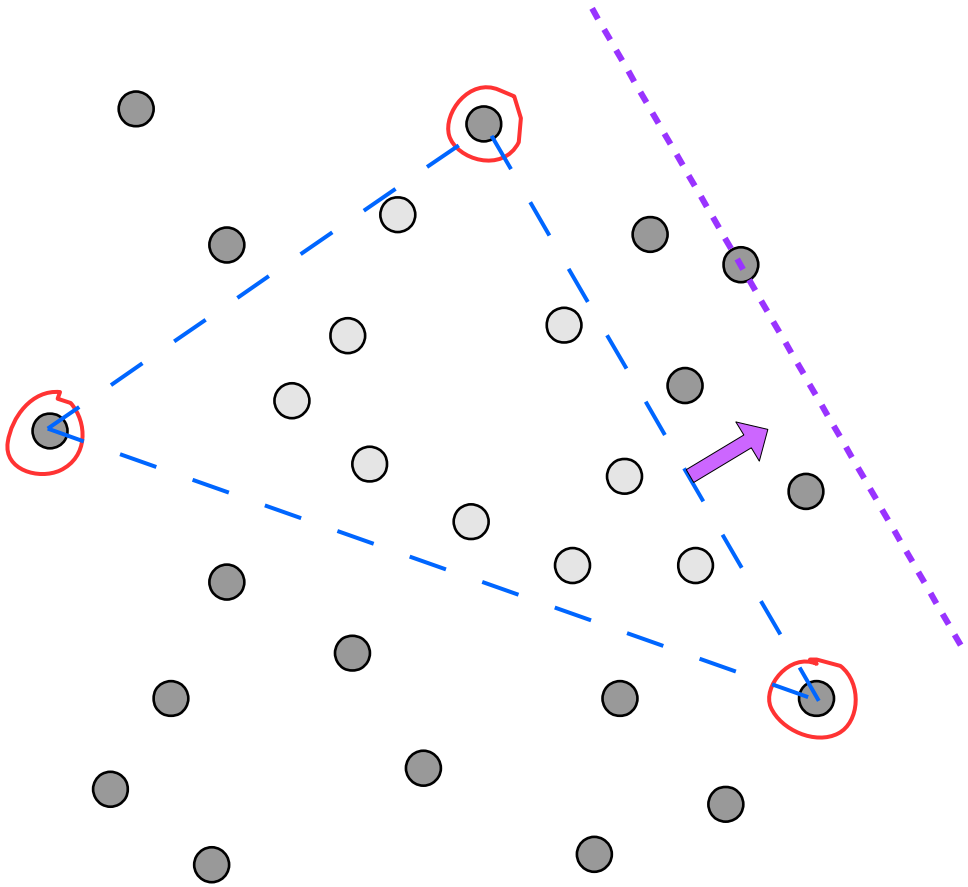
... they can't participate in the hull.

QuickHull



At the same time, we split our subtask into **2** even smaller subtasks,

QuickHull

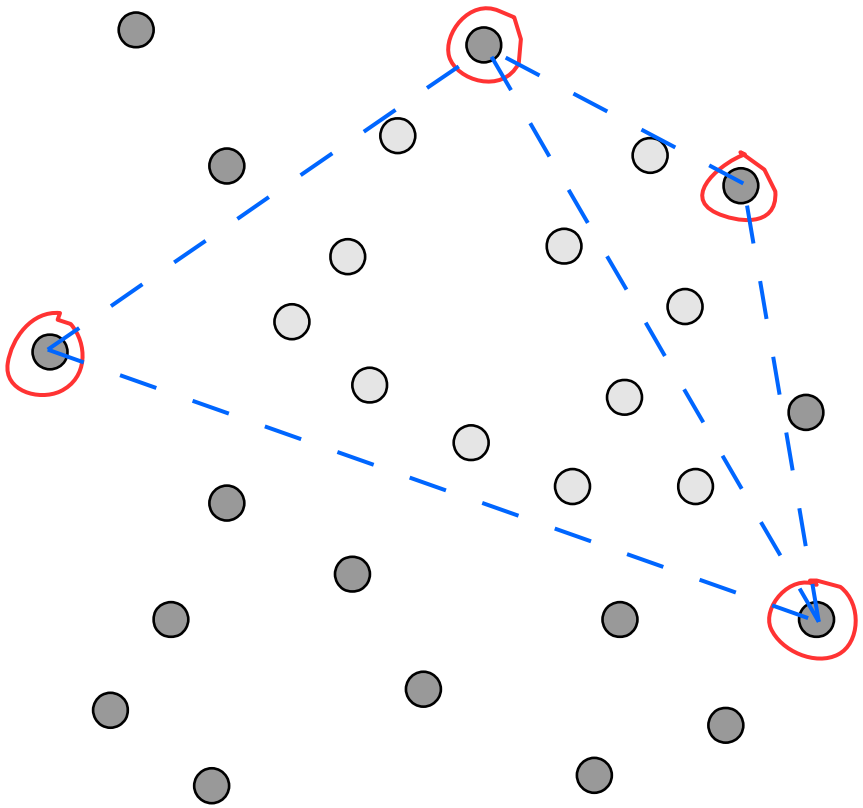


At the same time, we split our subtask into **2** even smaller subtasks,

... so we can continue recursively with every part.

... finding farthest point,

QuickHull



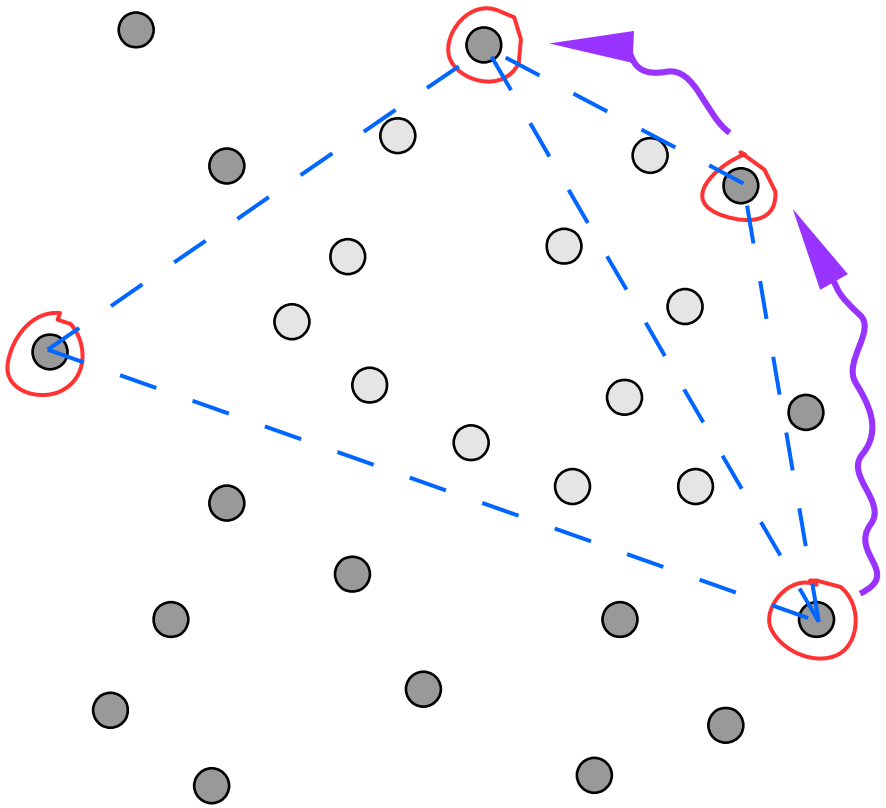
At the same time, we split our subtask into **2** even smaller subtasks,

... so we can continue recursively with every part.

... finding farthest point,

... discarding points inside triangle,

QuickHull



At the same time, we split our subtask into **2** even smaller subtasks,

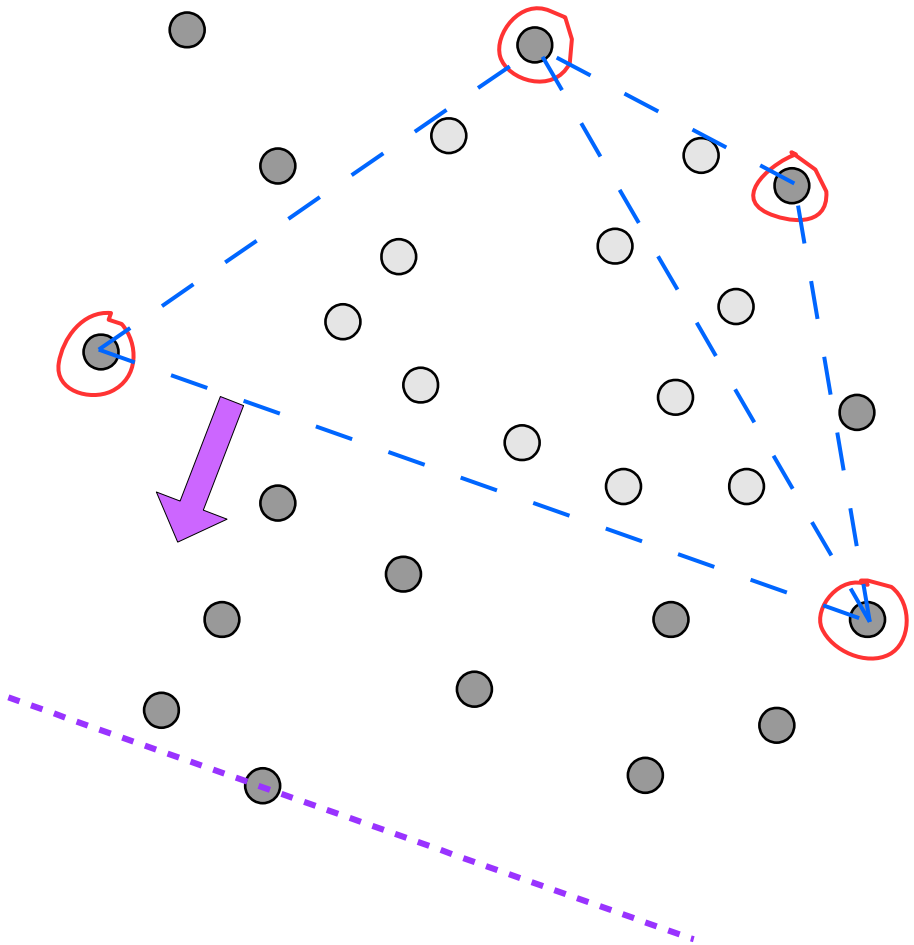
... so we can continue recursively with every part.

... finding farthest point,

... discarding points inside triangle,

... continue recursively,

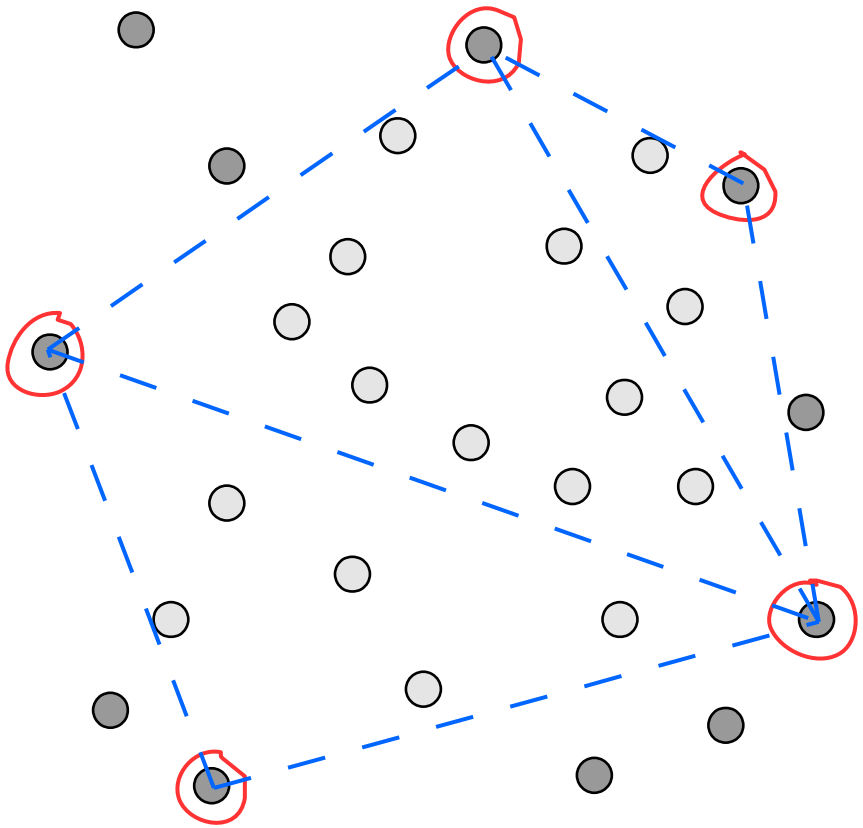
QuickHull



Let's observe one more step:

... finding farthest point,

QuickHull

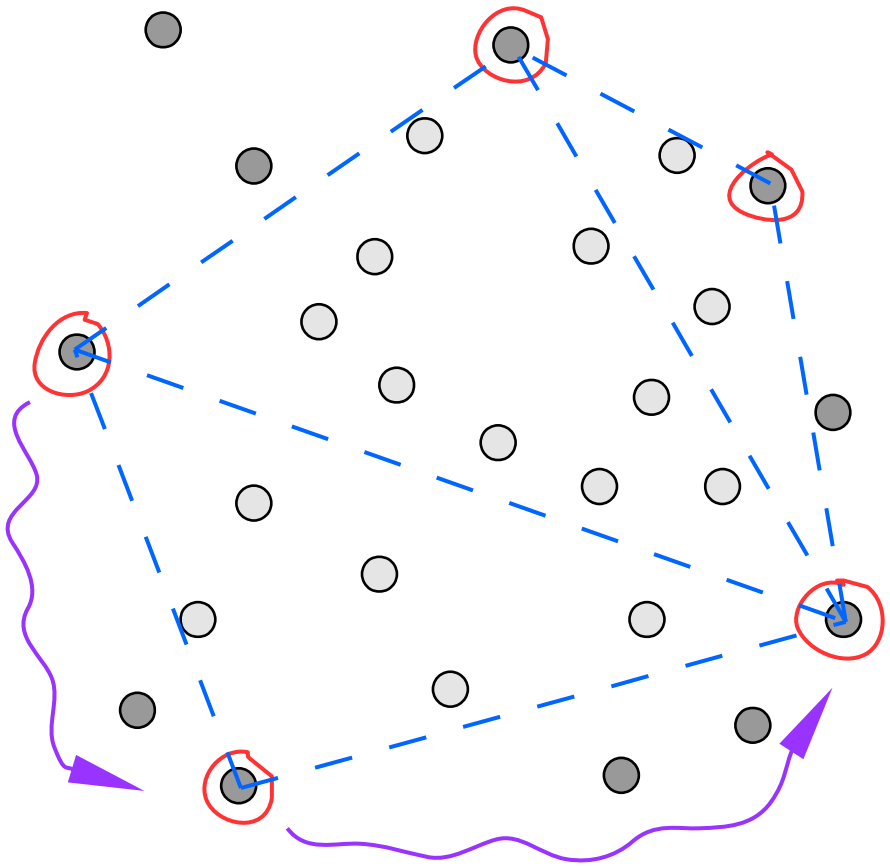


Let's observe one more step:

... finding farthest point,

... discarding points inside triangle,

QuickHull



Let's observe one more step:

... finding farthest point,

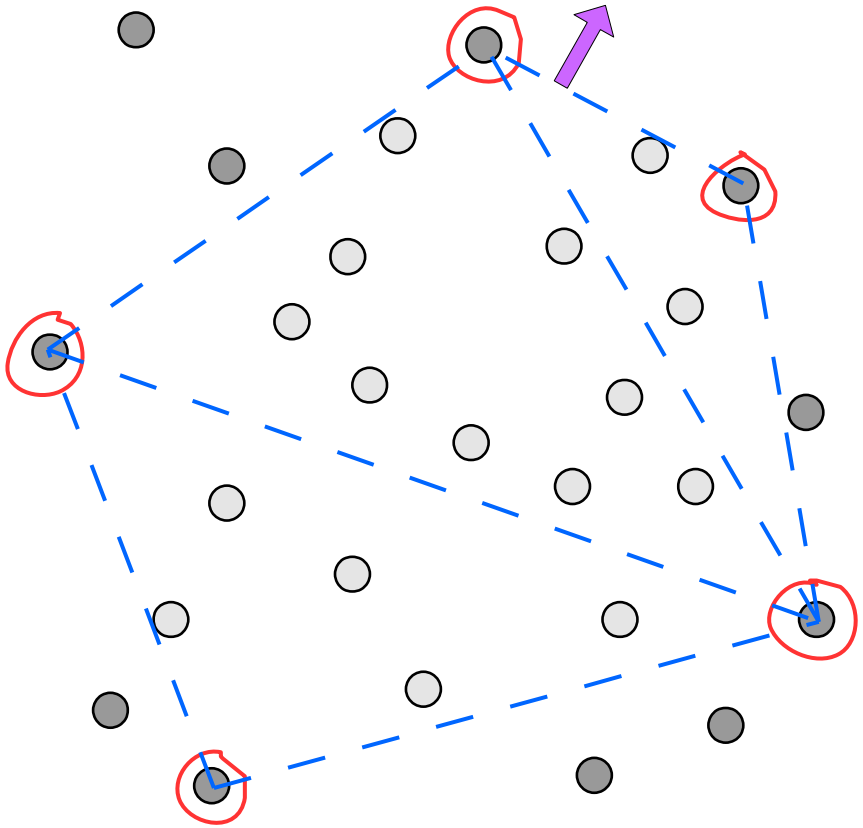
... discarding points inside triangle,

... continue recursively.

QuickHull

Presented algorithm is called QuickHull, in the analogy of QuickSort algorithm.

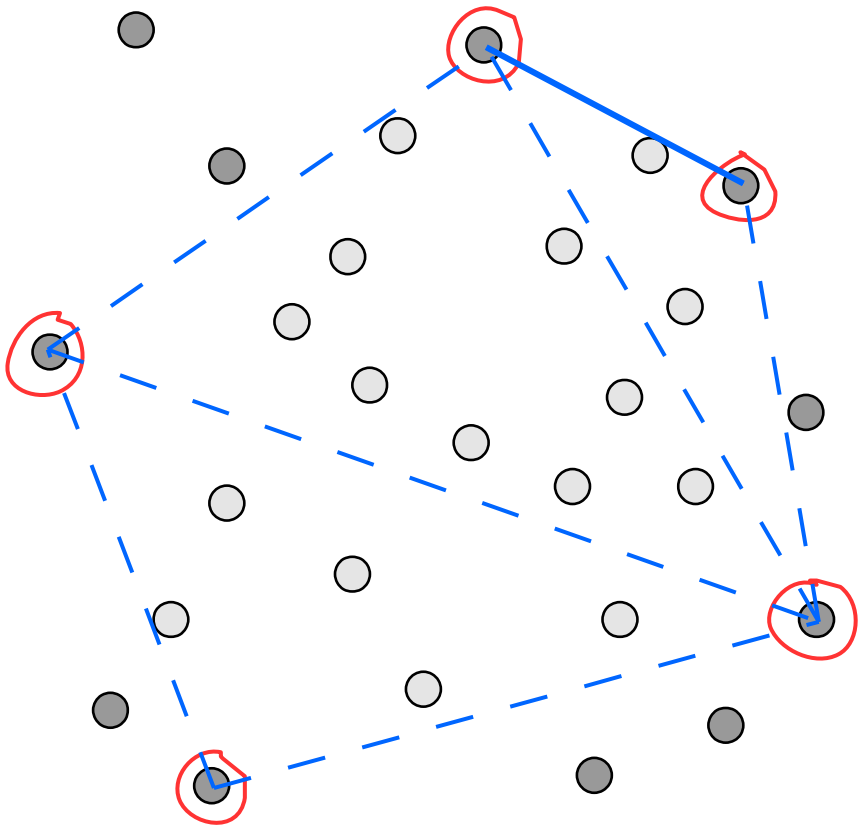
QuickHull



Recursion will terminate when there are no more points out of the line,

... so triangle can't be constructed.

QuickHull

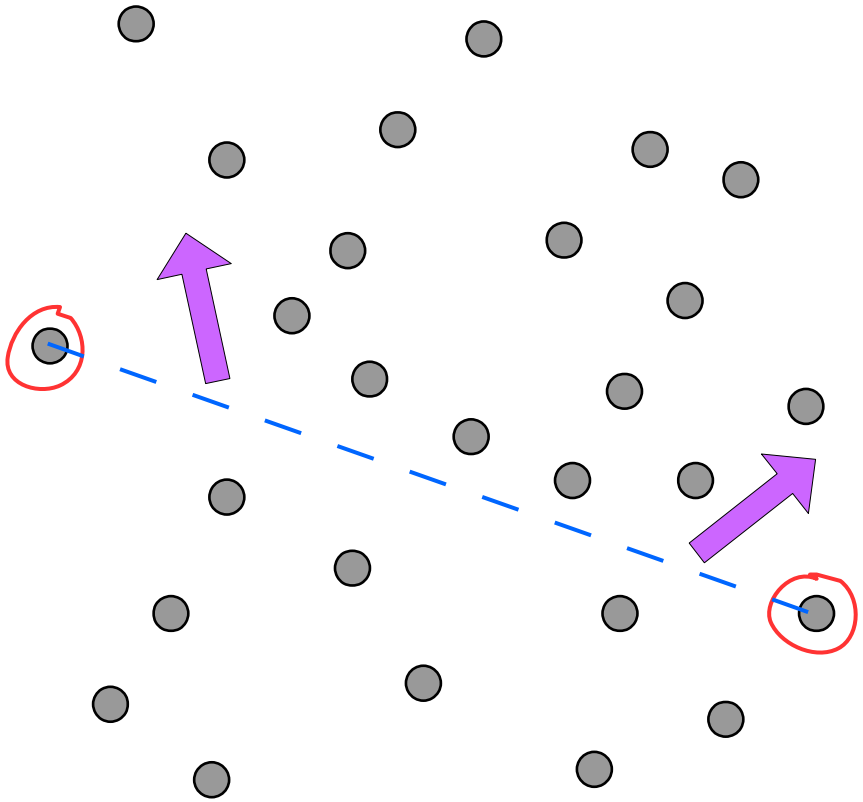


Recursion will terminate when there are no more points out of the line,

... so triangle can't be constructed.

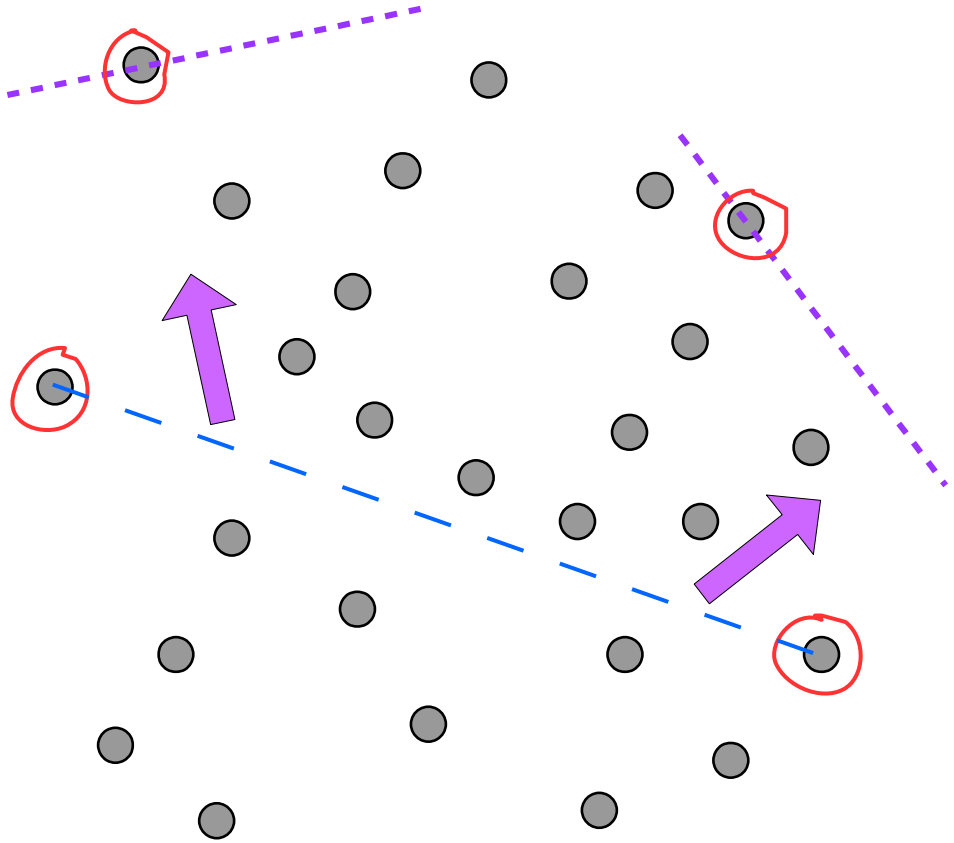
Reaching exit branch in the recursion means that those **2** points are adjacent in the hull.

QuickHull



Question: Can we split remaining set of points not in **2** outer parts, but in **3**?

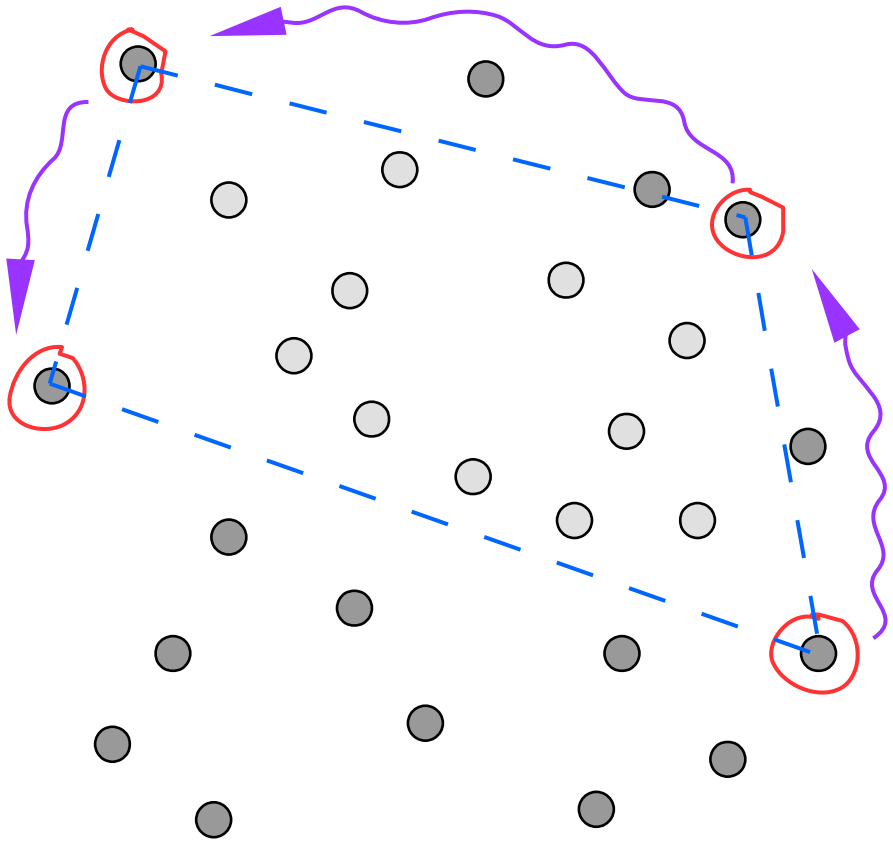
QuickHull



Question: Can we split remaining set of points not in **2** outer parts, but in **3**?

Answer: Yes, as every point farthest by any direction will participate in convex hull,

QuickHull



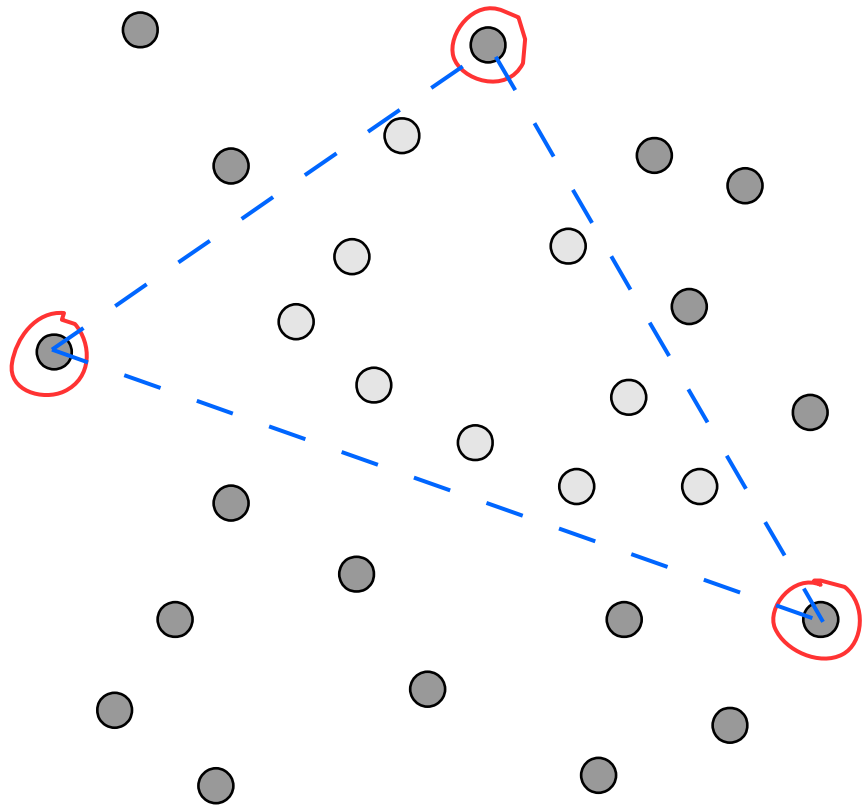
Question: Can we split remaining set of points not in **2** outer parts, but in **3**?

Answer: Yes, as every point farthest by any direction will participate in convex hull,

... so here we will need to do **3** recursive calls.

QuickHull

QuickHull is quite efficient algorithm, as it constantly discards lots of points.

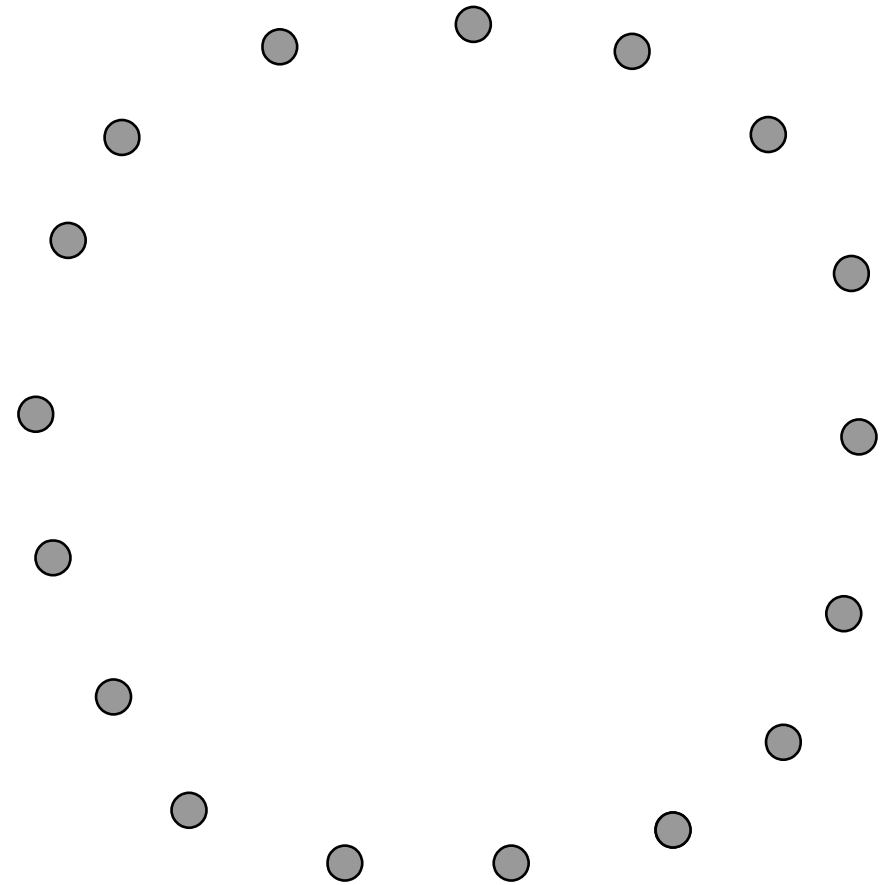


QuickHull

QuickHull is quite efficient algorithm, as it constantly discards lots of points.

Slow behavior will happen when there are no points to discard,

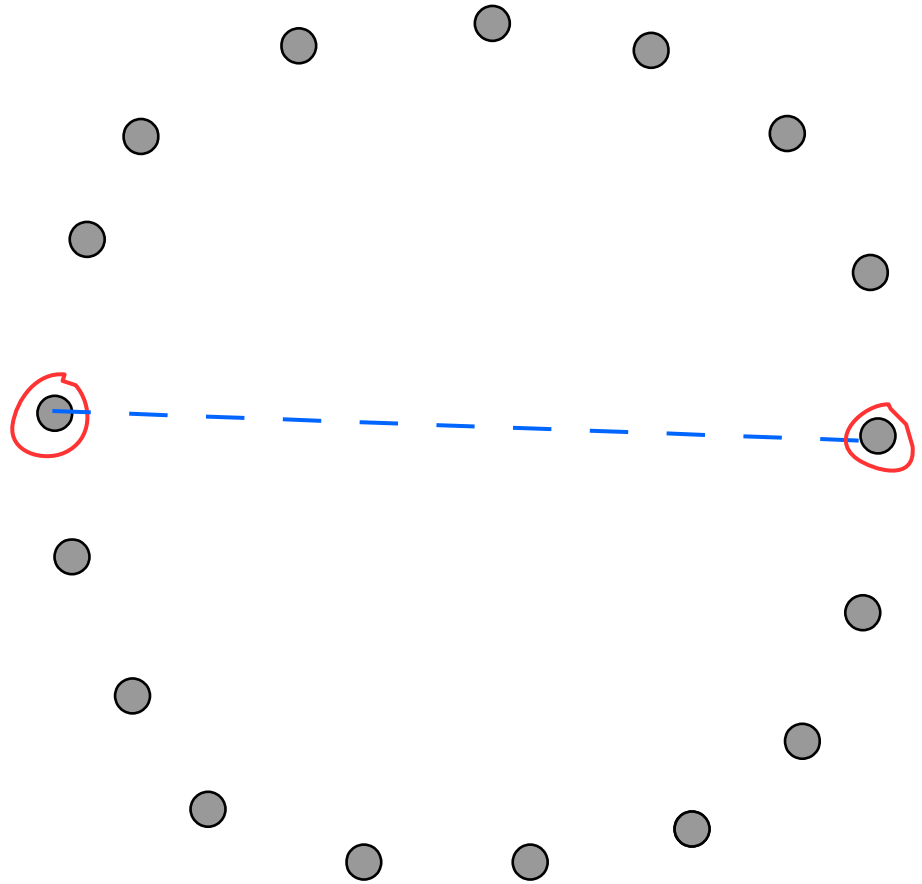
... in other words, if all **N** input points lie on the convex hull.



QuickHull

For such case we will spend:

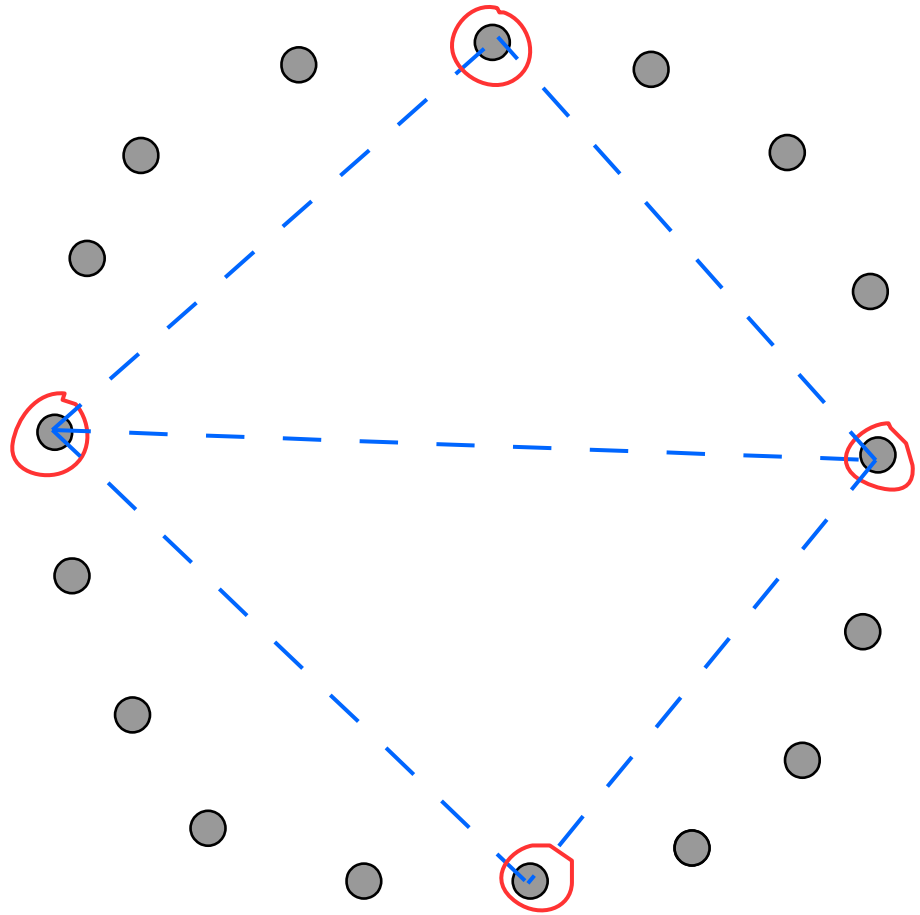
- **O(N)** time for finding leftmost and rightmost points,



QuickHull

For such case we will spend:

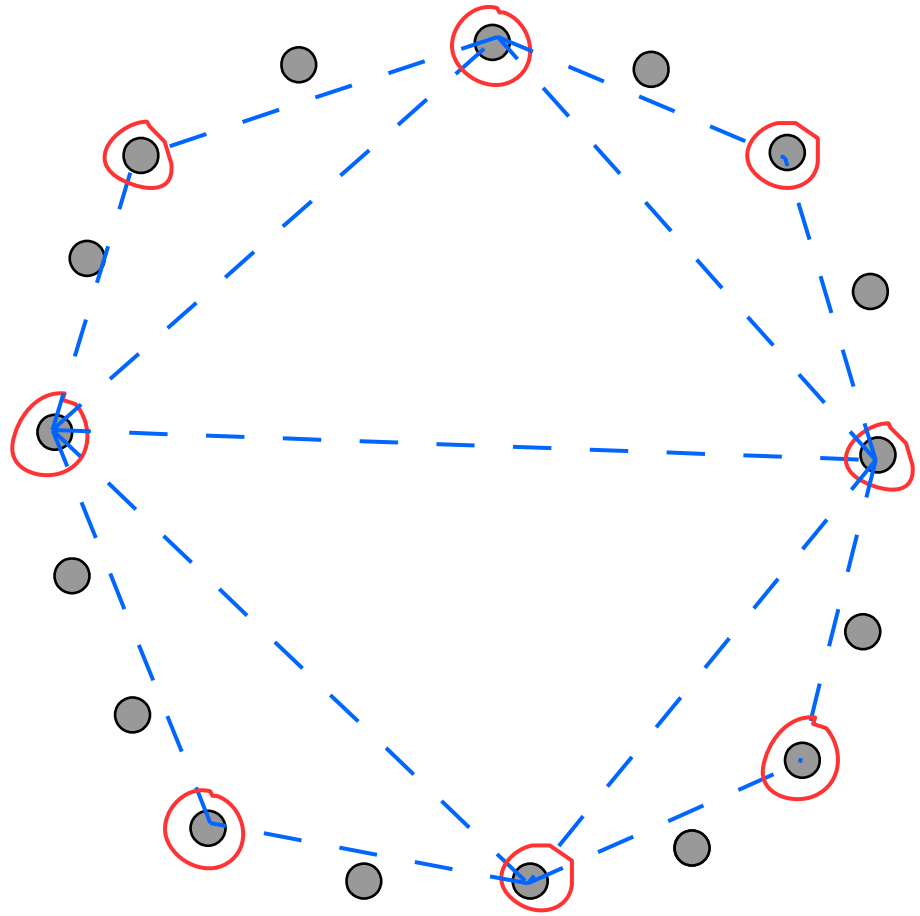
- **O(N)** time for finding leftmost and rightmost points,
- **O(2*(N/2))** time for finding farthest points in both directions,



QuickHull

For such case we will spend:

- **O(N)** time for finding leftmost and rightmost points,
- **O(2*(N/2))** time for finding farthest points in both directions,
- **O(4*(N/4))** time for finding farthest points in the **4** resulting directions,



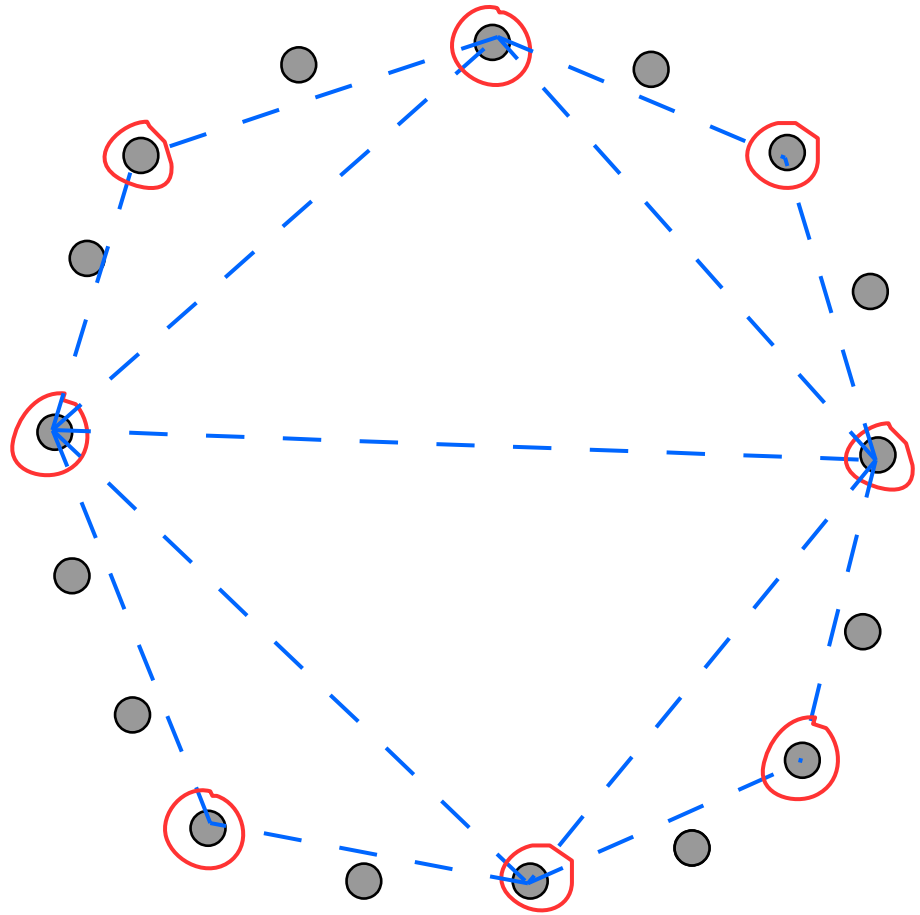
QuickHull

... which leads to:

N + 2*(N/2) + 4*(N/4) + ... + N =

= N*log$_2$N = O(N*logN).

So even in the case <u>when we discard no points</u>, performance is good.



QuickHull

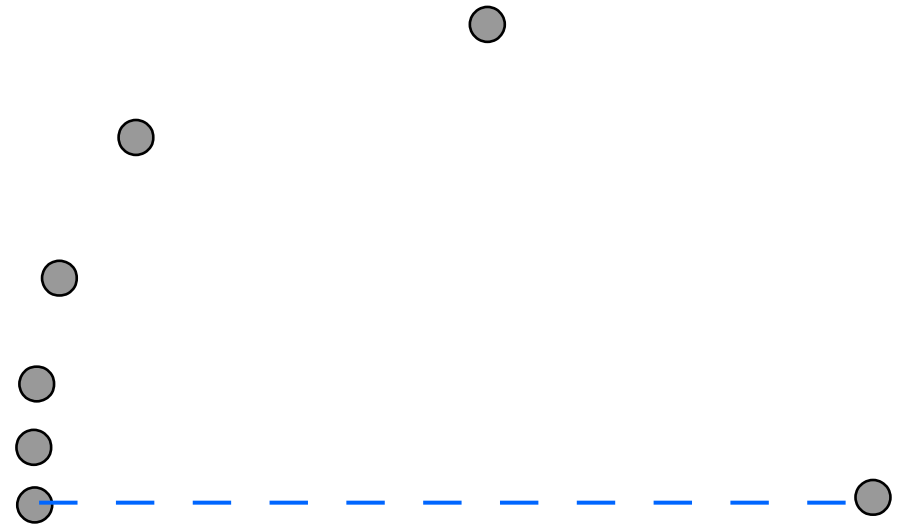
Question: Is there such an input, which will cause **O(N^2)** time complexity?

QuickHull

Question: Is there such an input, which will cause **O(N^2)** time complexity?

Answer: Yes there is...

... where points are arranged on a convex shape (circle), concentrating towards one point in geometrical progression,

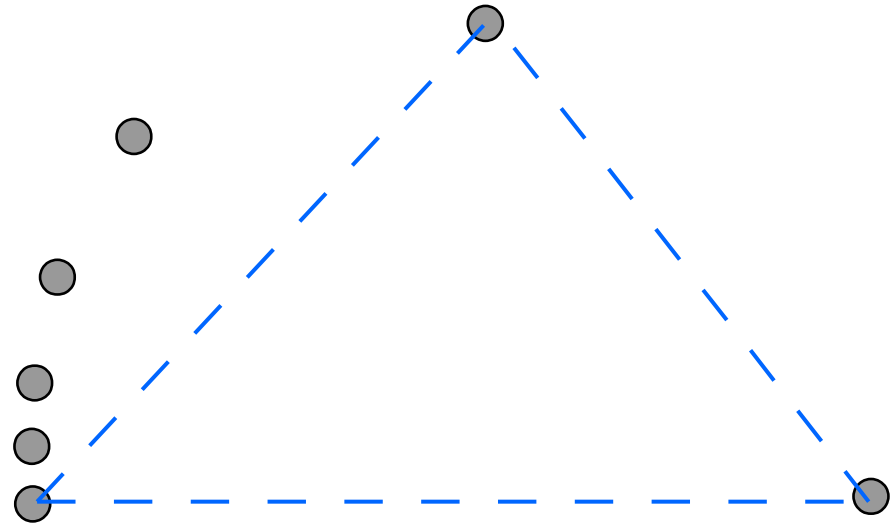


QuickHull

Question: Is there such an input, which will cause **O(N^2)** time complexity?

Answer: Yes there is...

... where points are arranged on a convex shape (circle), concentrating towards one point in geometrical progression,

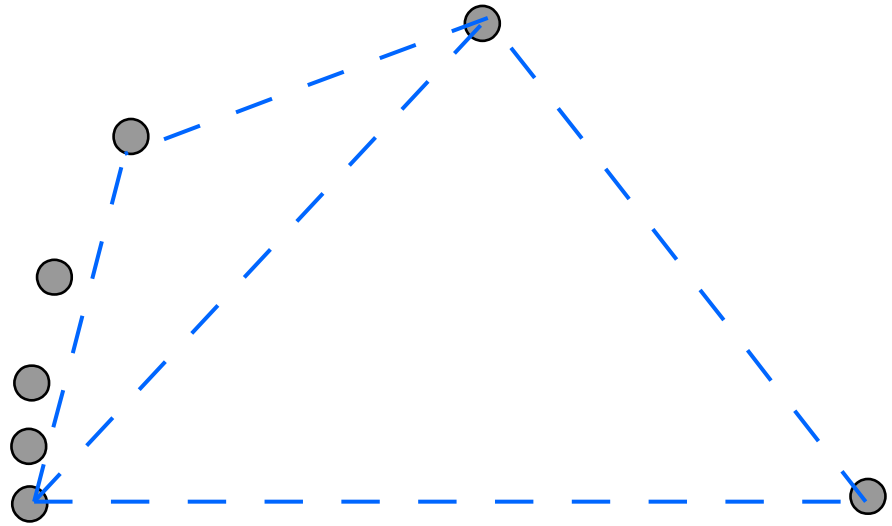


QuickHull

Question: Is there such an input, which will cause **O(N^2)** time complexity?

Answer: Yes there is...

... where points are arranged on a convex shape (circle), concentrating towards one point in geometrical progression,

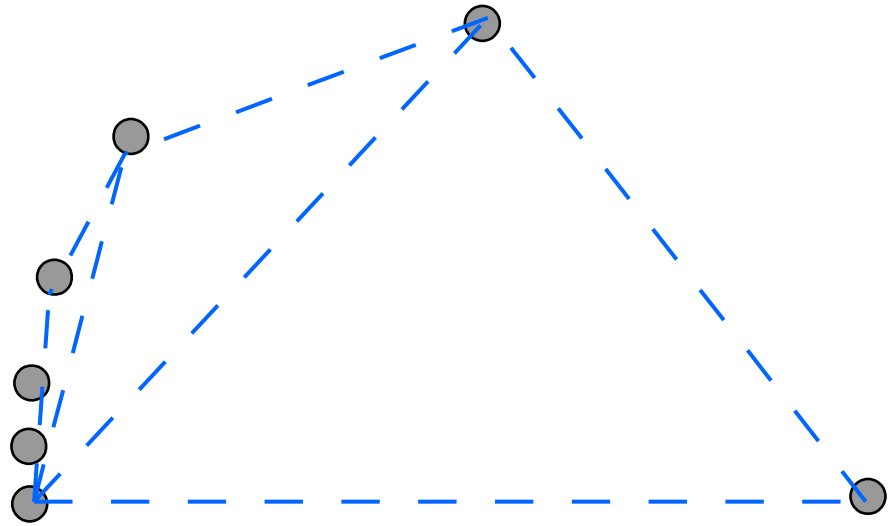


QuickHull

Question: Is there such an input, which will cause **O(N^2)** time complexity?

Answer: Yes there is...

... where points are arranged on a convex shape (circle), concentrating towards one point in geometrical progression,



... obviously, it is quite improbable.

Exercise

Draw around **25-30** points, and perform QuickHull algorithm on them.

Presentation writer: Tigran Hayrapetyan

Lecturer | Programmer | Researcher

www.linkedin.com/in/tigran-hayrapetyan-cs/

Thank you!

Convex hull
(QuickHull)