CAD & CG: Practical work 6

Convex hull of a set of points

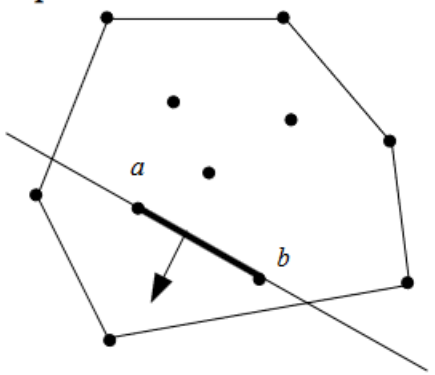
Two algorithms:

- **Slow algorithm** $O(n^3)$, which is given
- **Graham algorithm** $O(n \log n)$, which is the object of this tutorial

Convex hull of a set of points

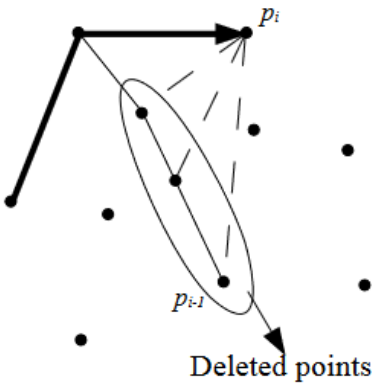
- Slow algorithm $O(n^3)$

ConvexHullSlow(P, L)
Intput : a set of points P in the euclidean plane
Output : an ordered list of points L of the vertices defining the polygon CH(P) in a clockwise order
{
$E = \emptyset$ // set of edges
For every pair of points $(a, b) \in P \times P$ with $a \neq b$
{
valid=true
For every point of $p \in P$, $p \neq a$ et $p \neq b$
{
If p is on the left of ab then set *valid=false*, and exit the loop.
}
If (*valid=true*) then apend (a,b) to E.
}
Build an ordered list L of vertices from the unordered set of edges E.
}



Convex hull of a set of points

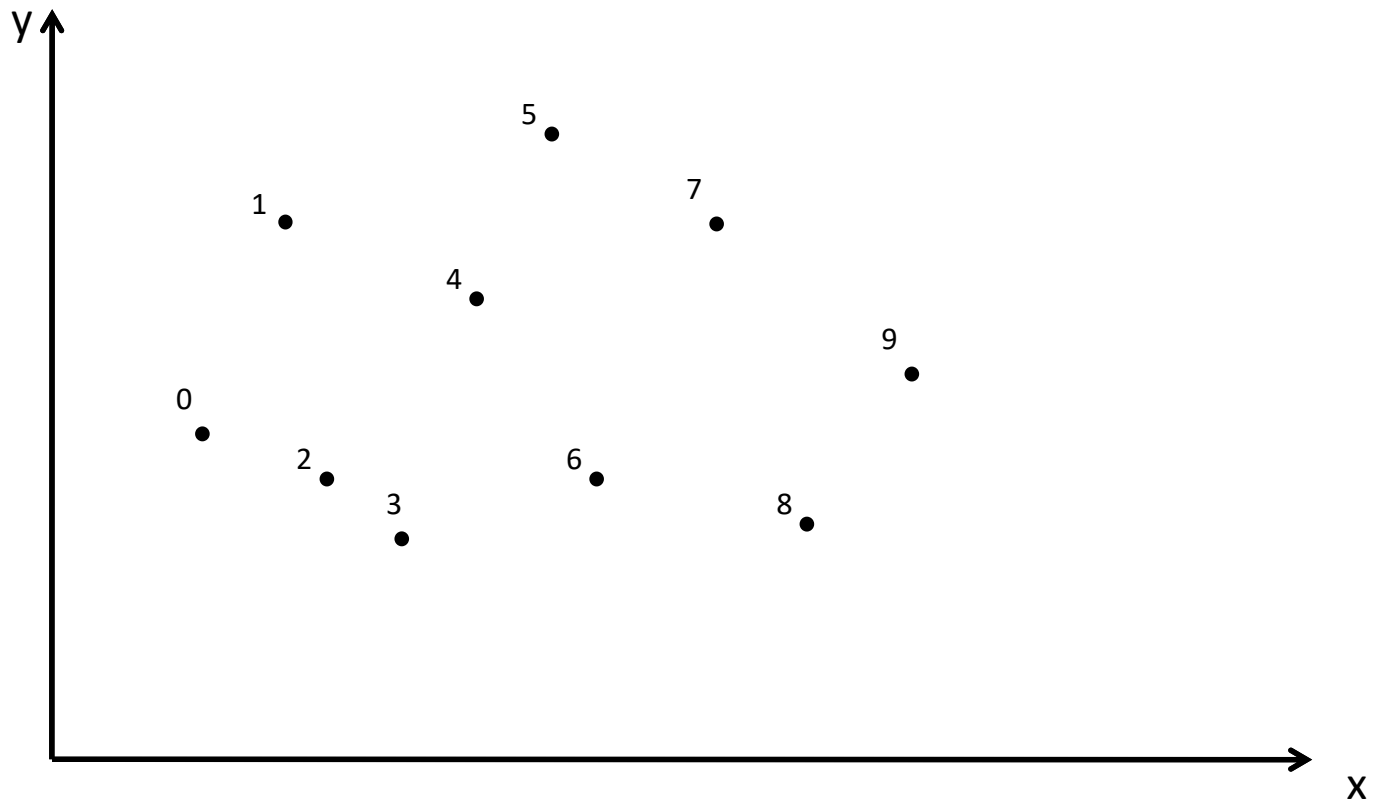
- Graham algorithm O(n log n)
  - The delicate step is the update of the convex hull after each new point insertion p_i.
    - From $p_1, \ldots, p_{i-1}$ one wants $p_1, \ldots, p_i$
    - There is one hint: when "walkin" a convex polygon in clowise fashion, every turn is a "right turn" at each vertex. Therefore :



  - p_i est obviously the last point of L_{sup}, hence it is inserted
  - One checks the 3 last points of L_{sup},
    - If the turn is a "right turn", end here.
    - Otherwise, one has to delete the beforelast point of L_{sup}, and recheck the three last points – until there is a right turn (or only two points in L_{sup})
  - Same idea for L_{inf}

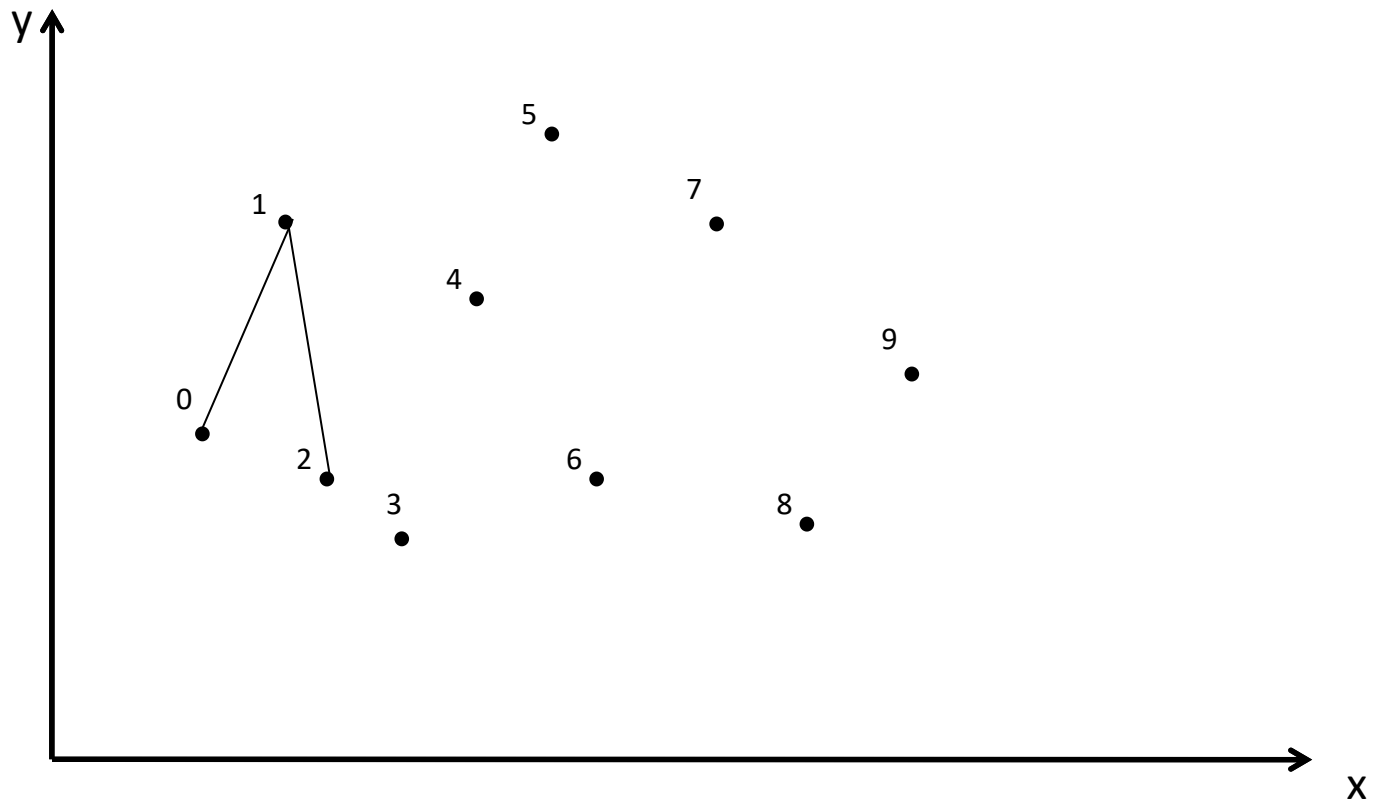
Convex hull of a set of points

- Sort points along x (and y) *std::sort*



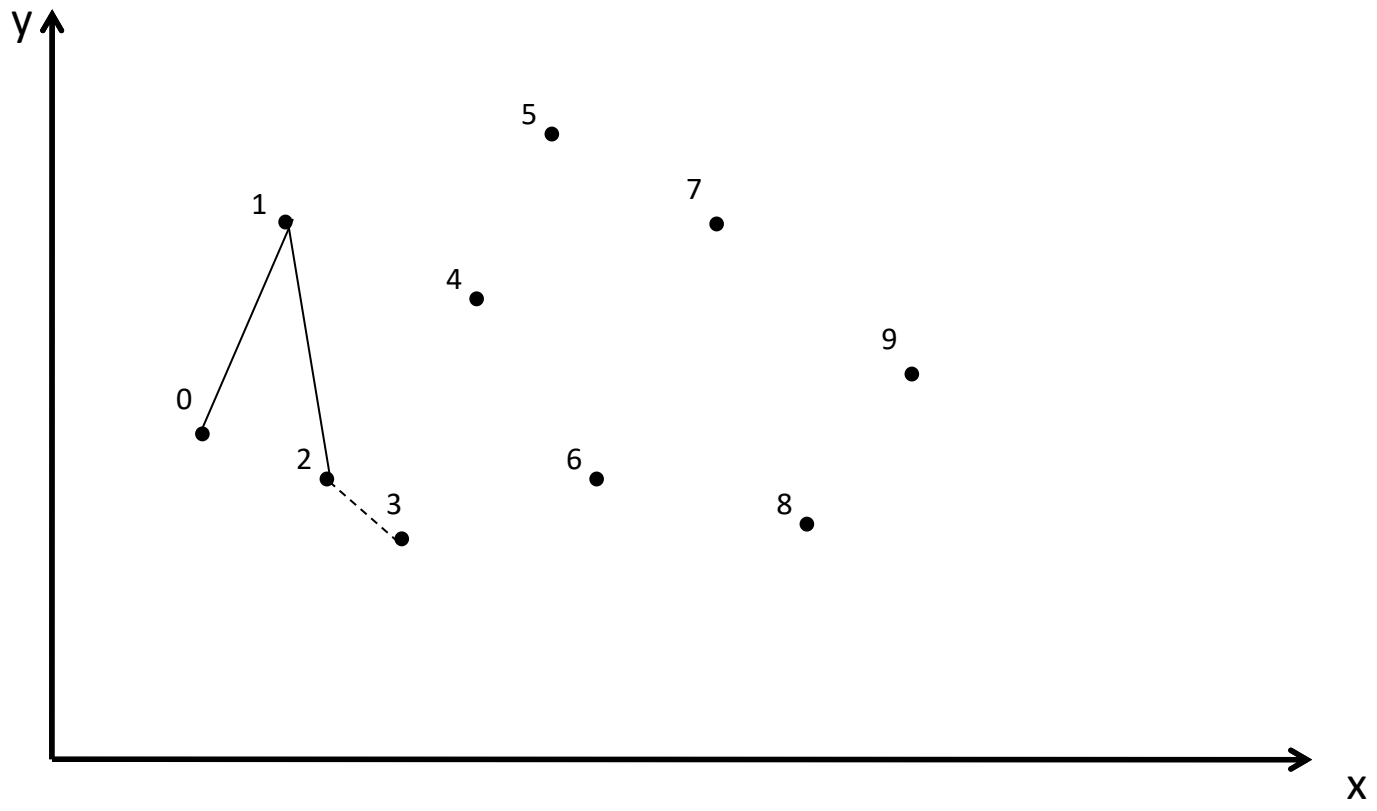
Convex hull of a set of points

- Take the first three points



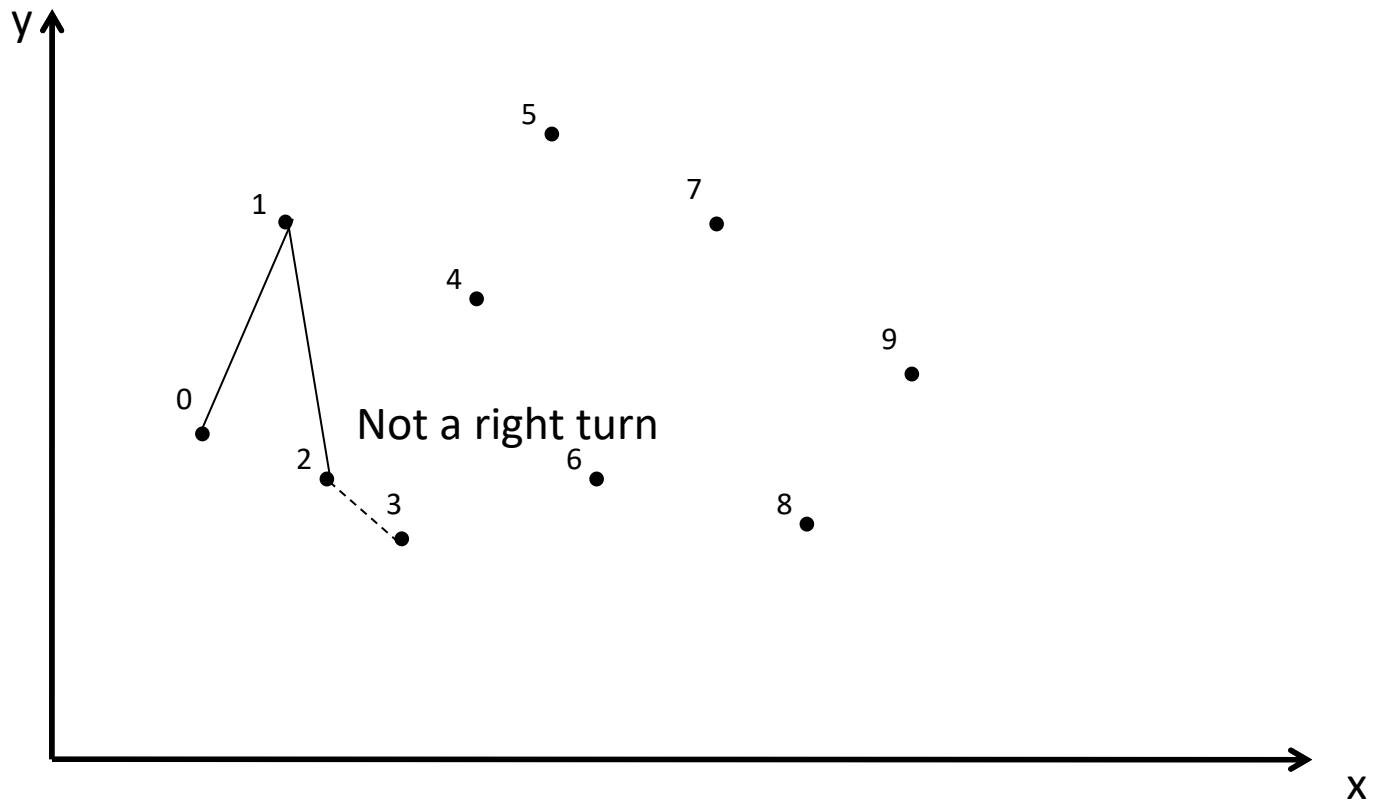
Convex hull of a set of points

- Test the next point



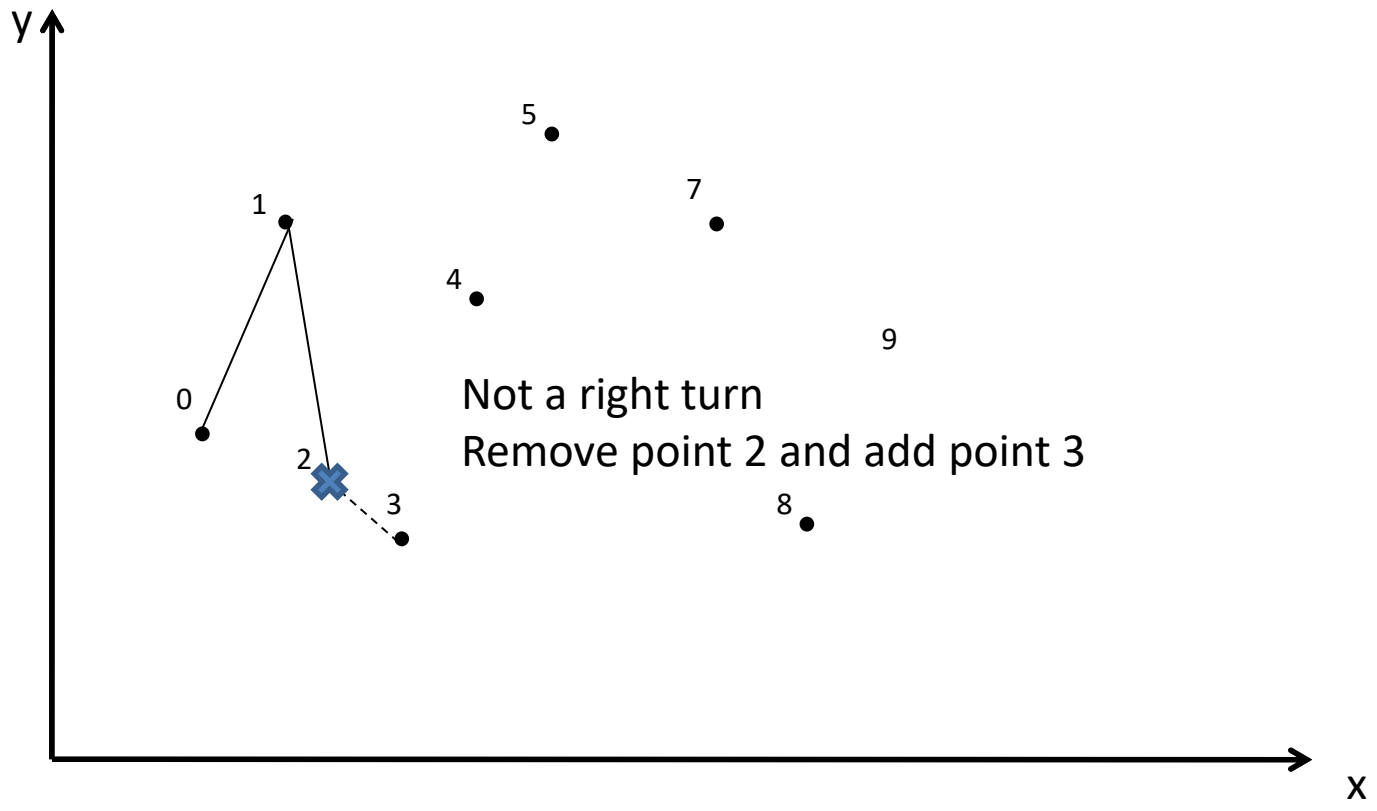
Convex hull of a set of points

- Test the next point



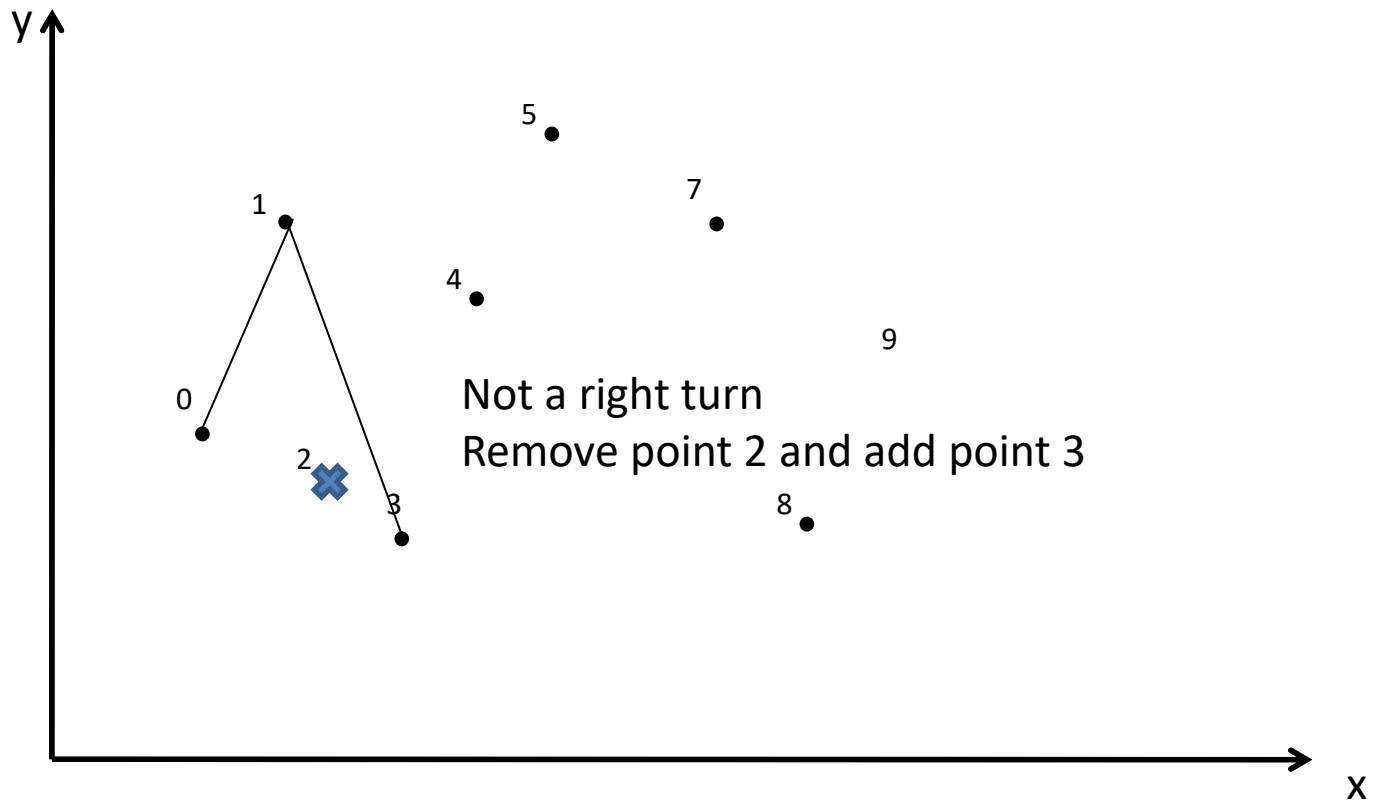
Convex hull of a set of points

- Test the next point



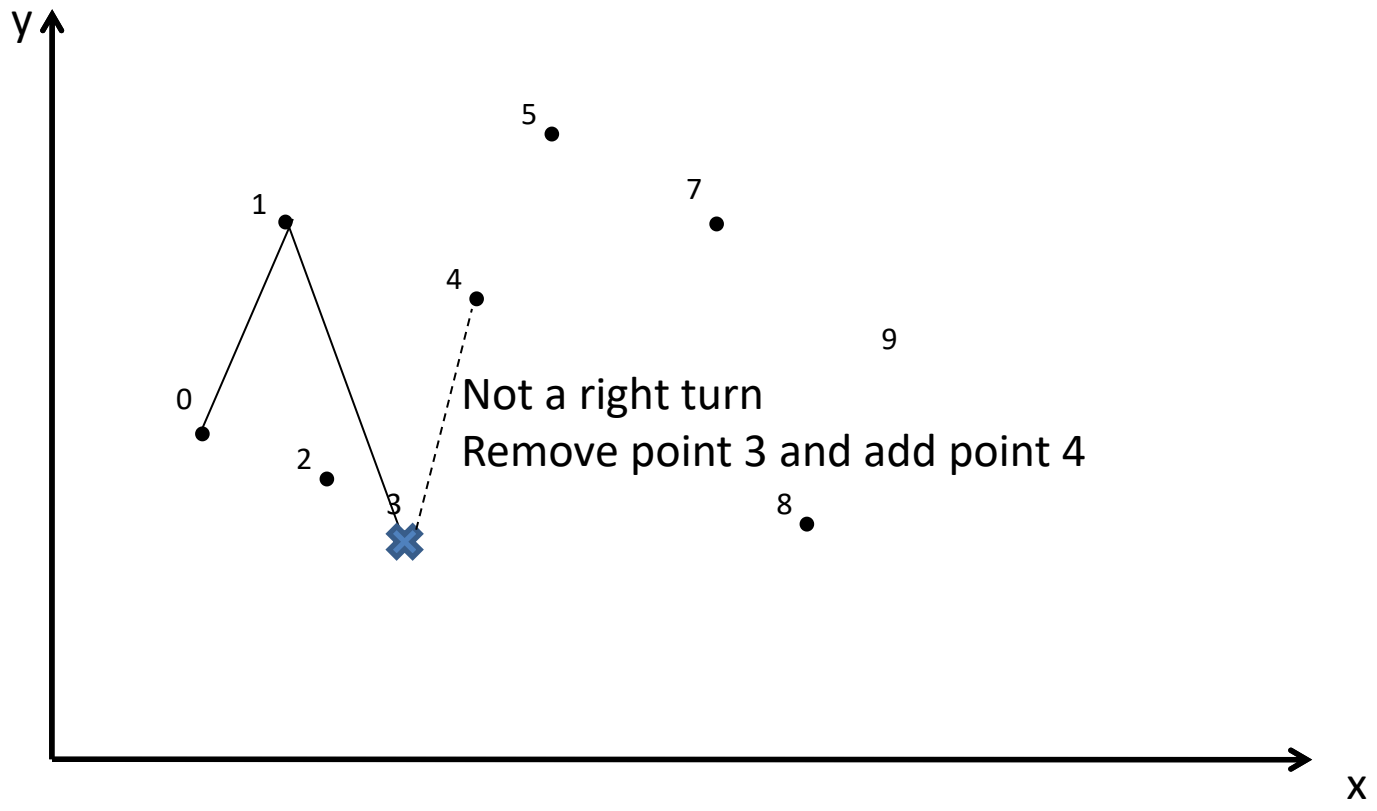
Convex hull of a set of points

- Test the next point



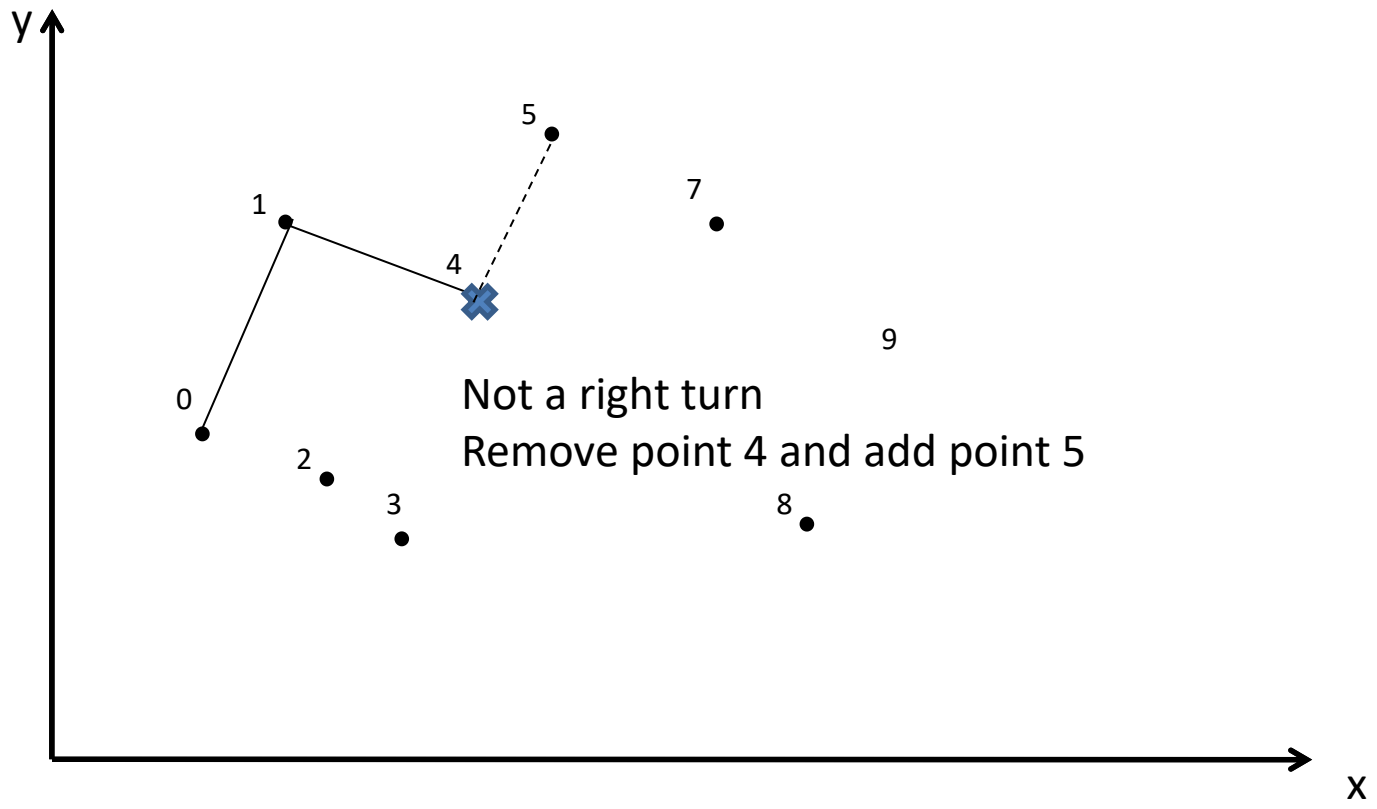
Convex hull of a set of points

- Test the next point



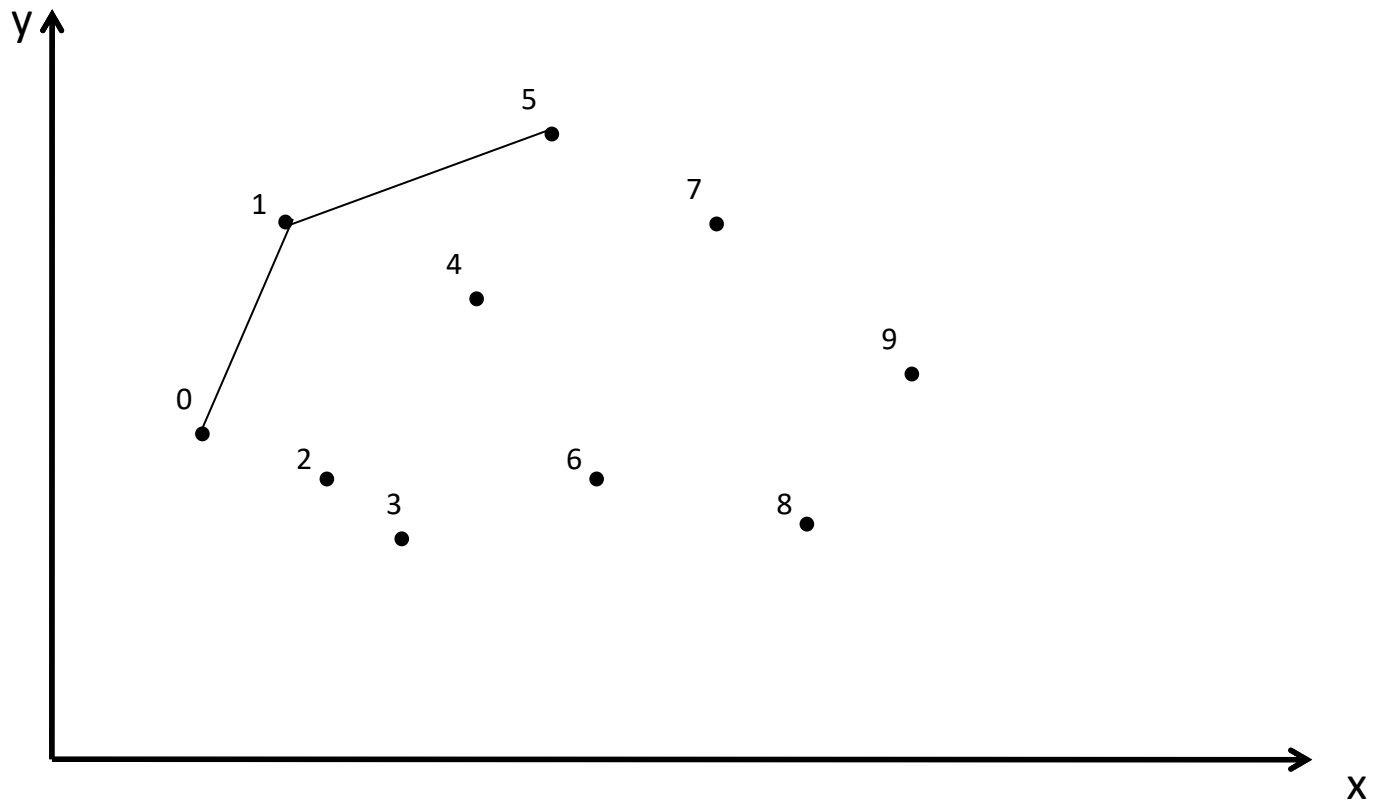
Convex hull of a set of points

- Test the next point



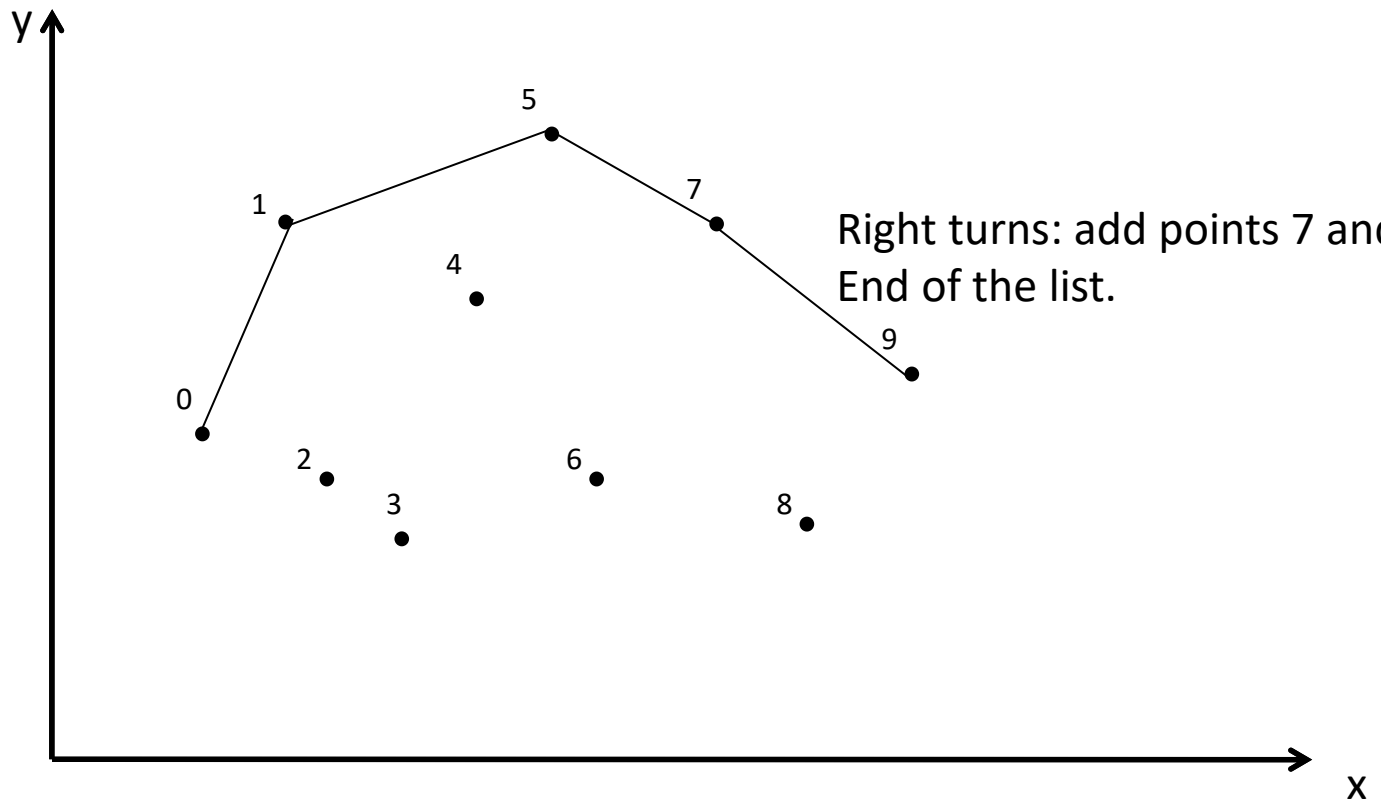
Convex hull of a set of points

- Test the next point



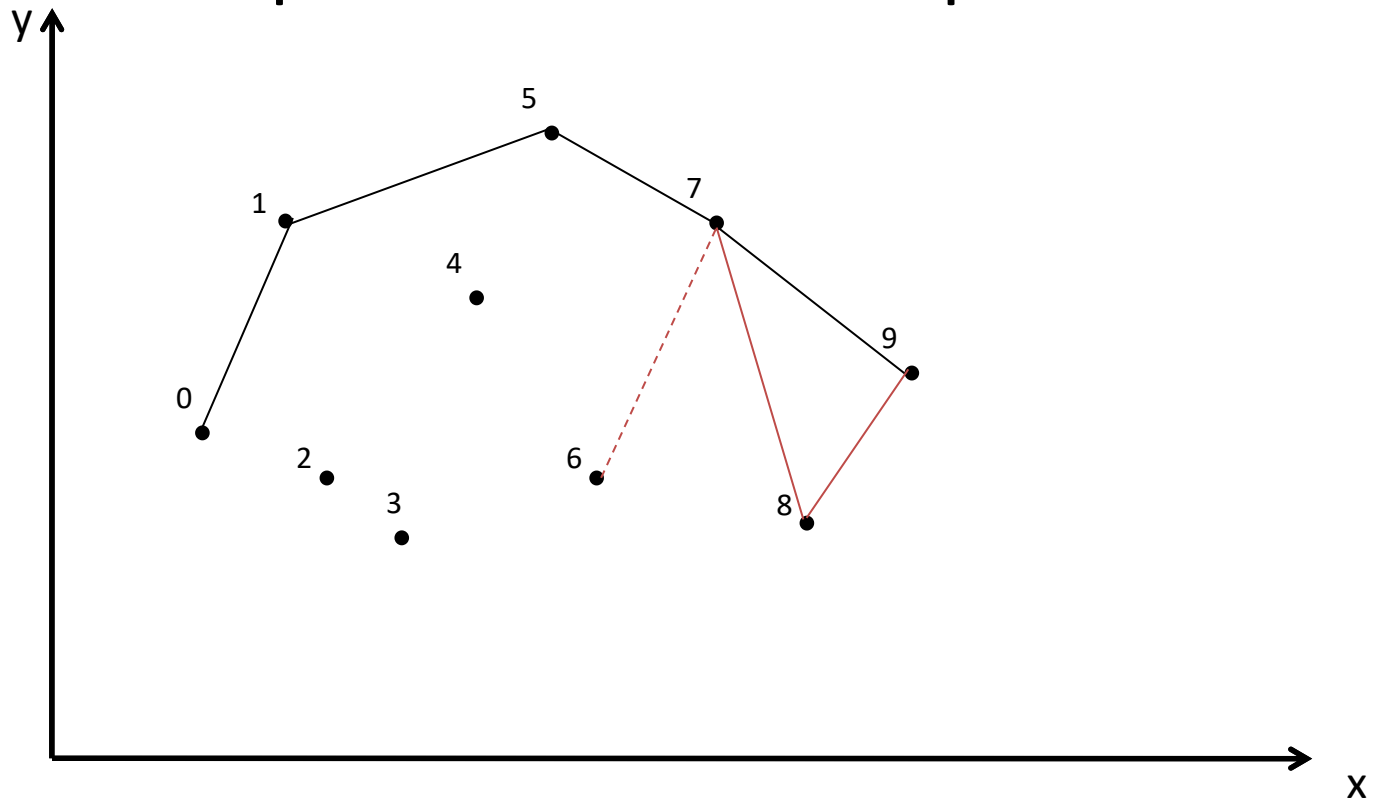
Convex hull of a set of points

- Test the next point and end



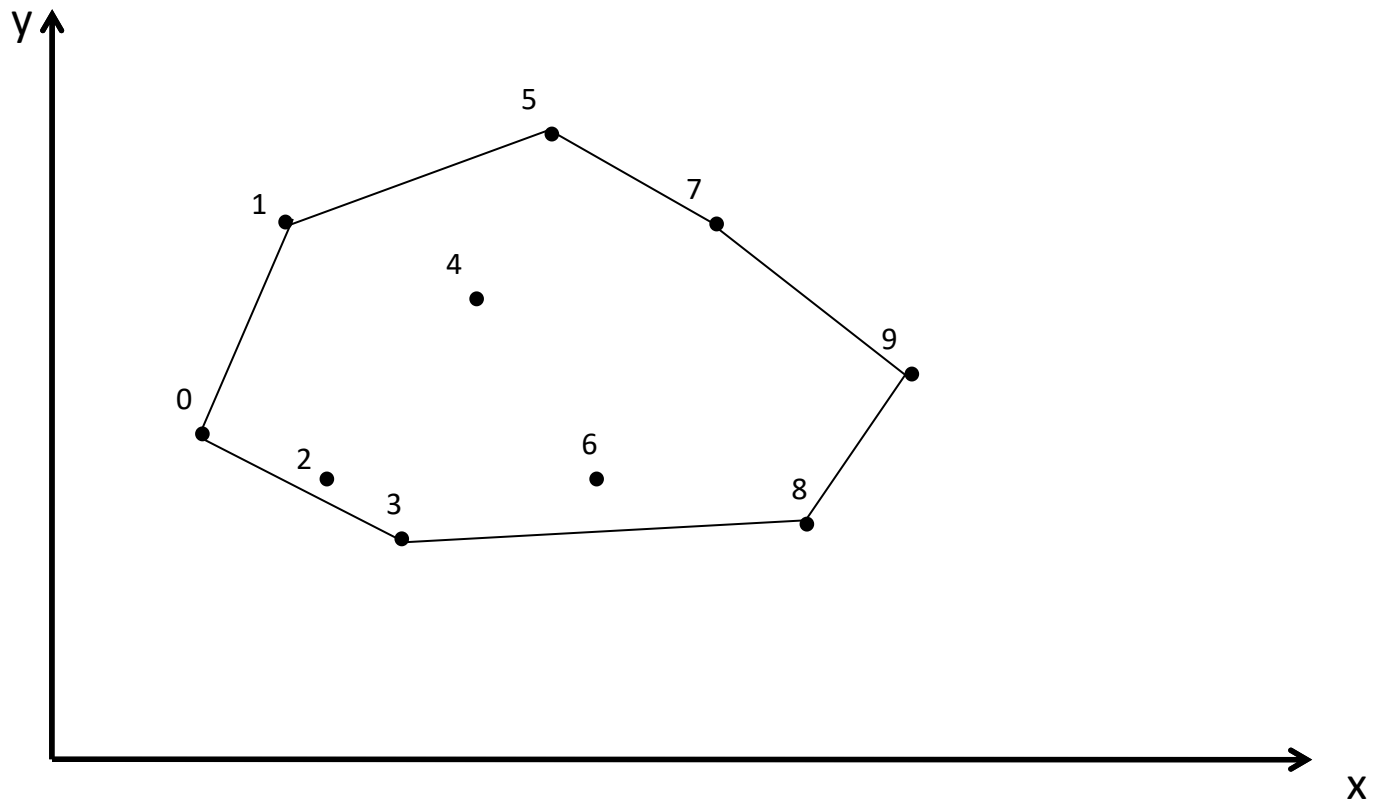
Convex hull of a set of points

- Lower part: same procedure, but start from the last points and test the previous ones.



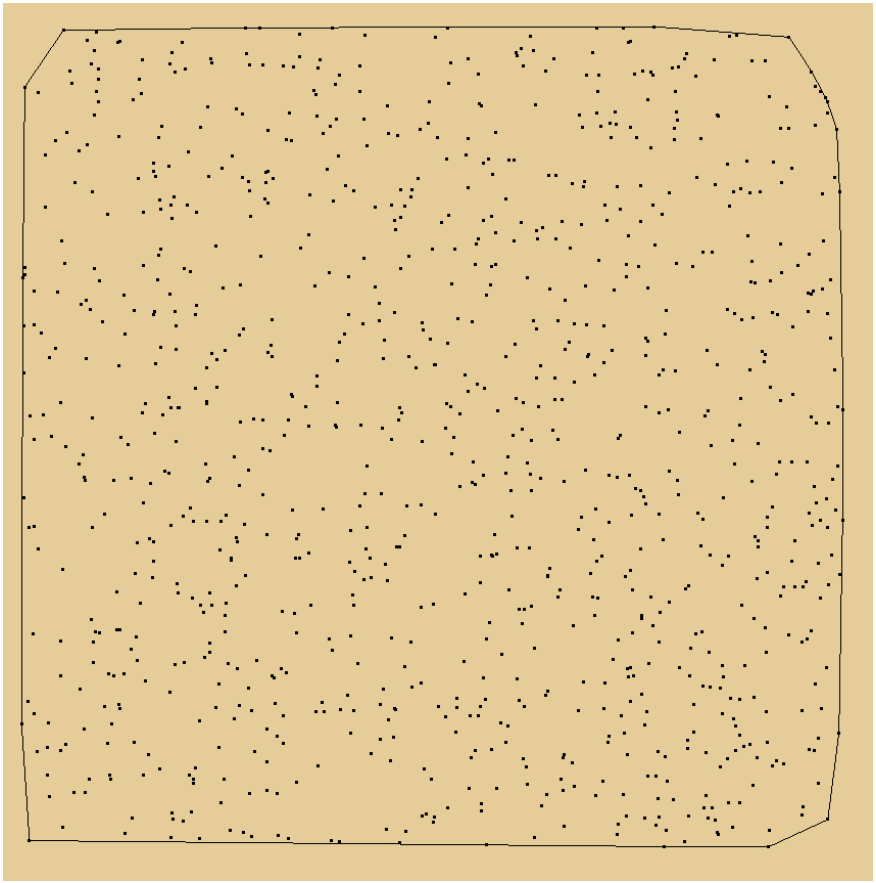
Convex hull of a set of points

- Convex hull: union of upper and lower parts



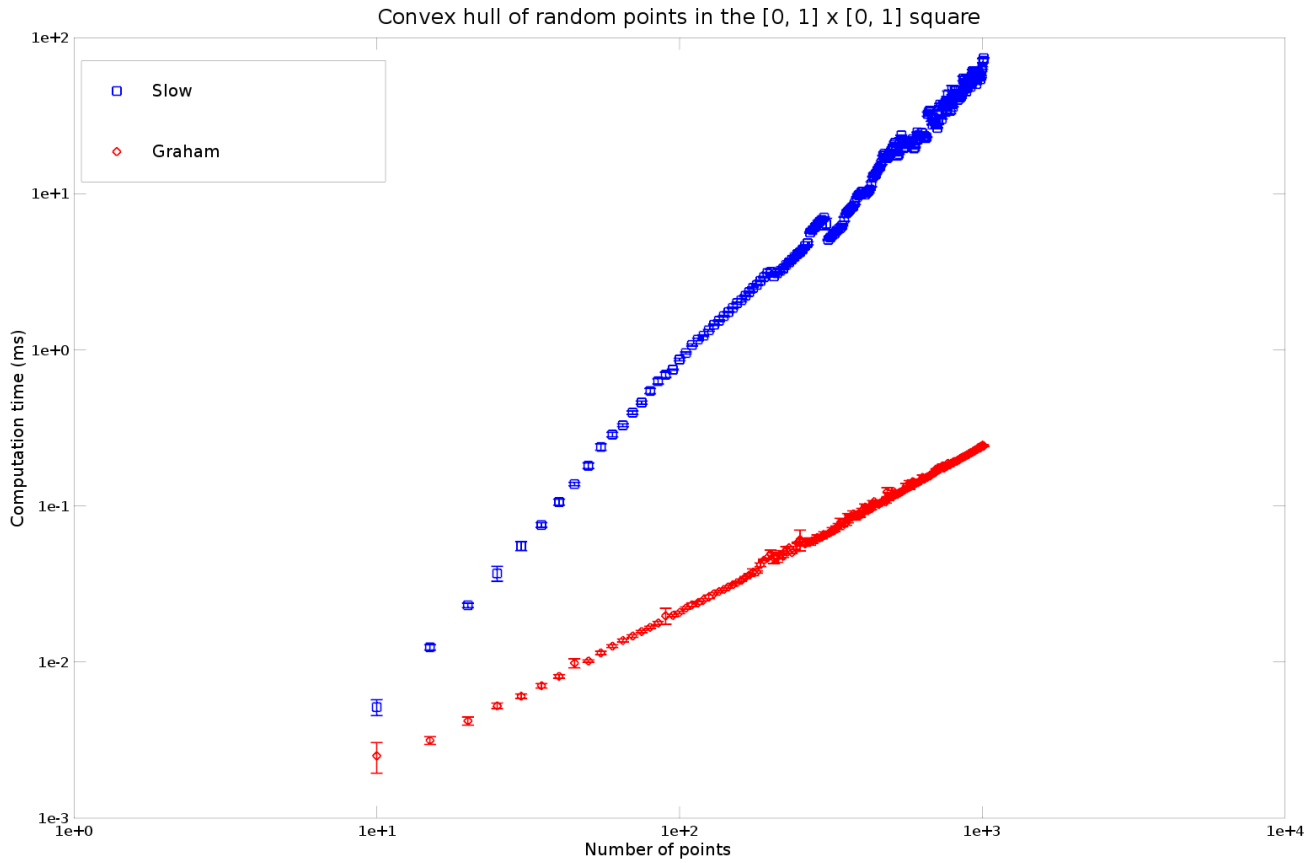
Convex hull of a set of points: complexity of both algorithms

- Test 1: random points in square [0, 1] x [0, 1]



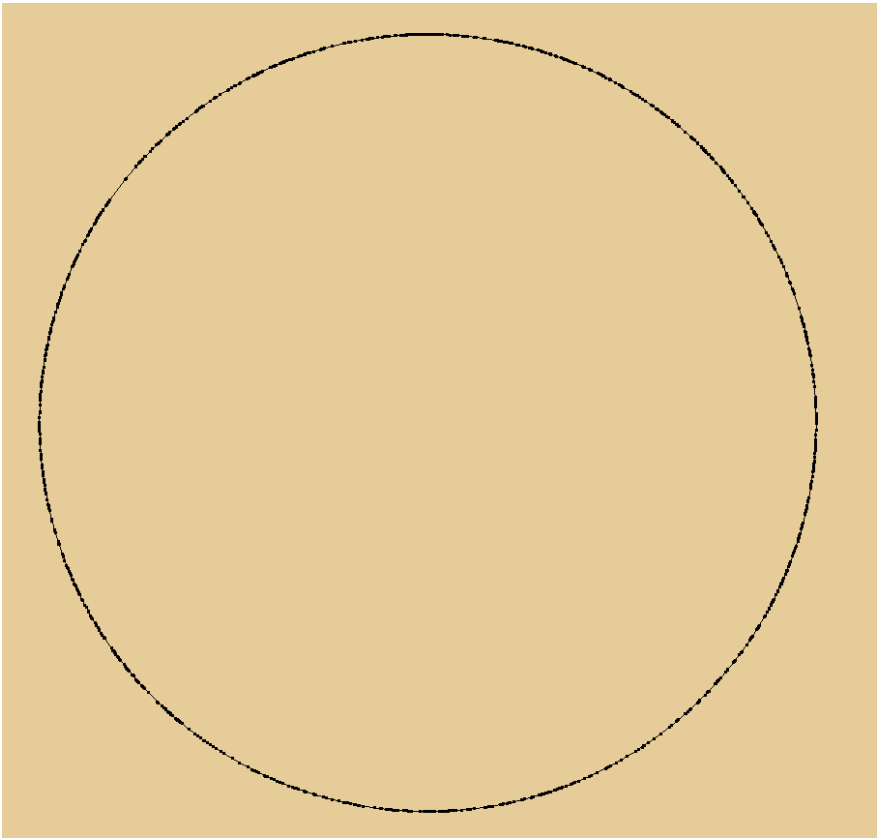
Convex hull of a set of points: complexity of both algorithms

- Test 1: random points in square [0, 1] x [0, 1]



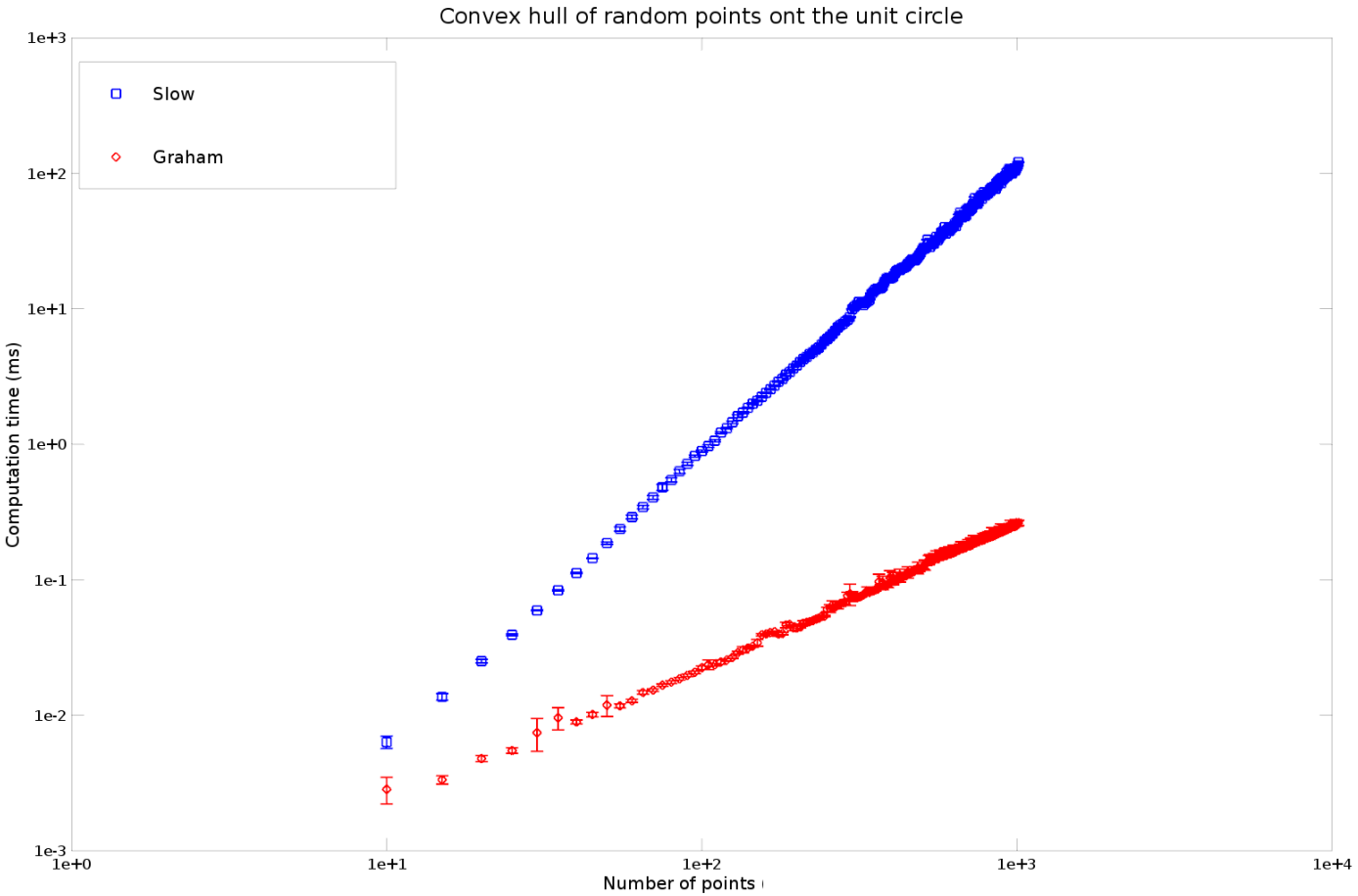
Convex hull of a set of points: complexity of both algorithms

- Test 2: random points on the unit circle



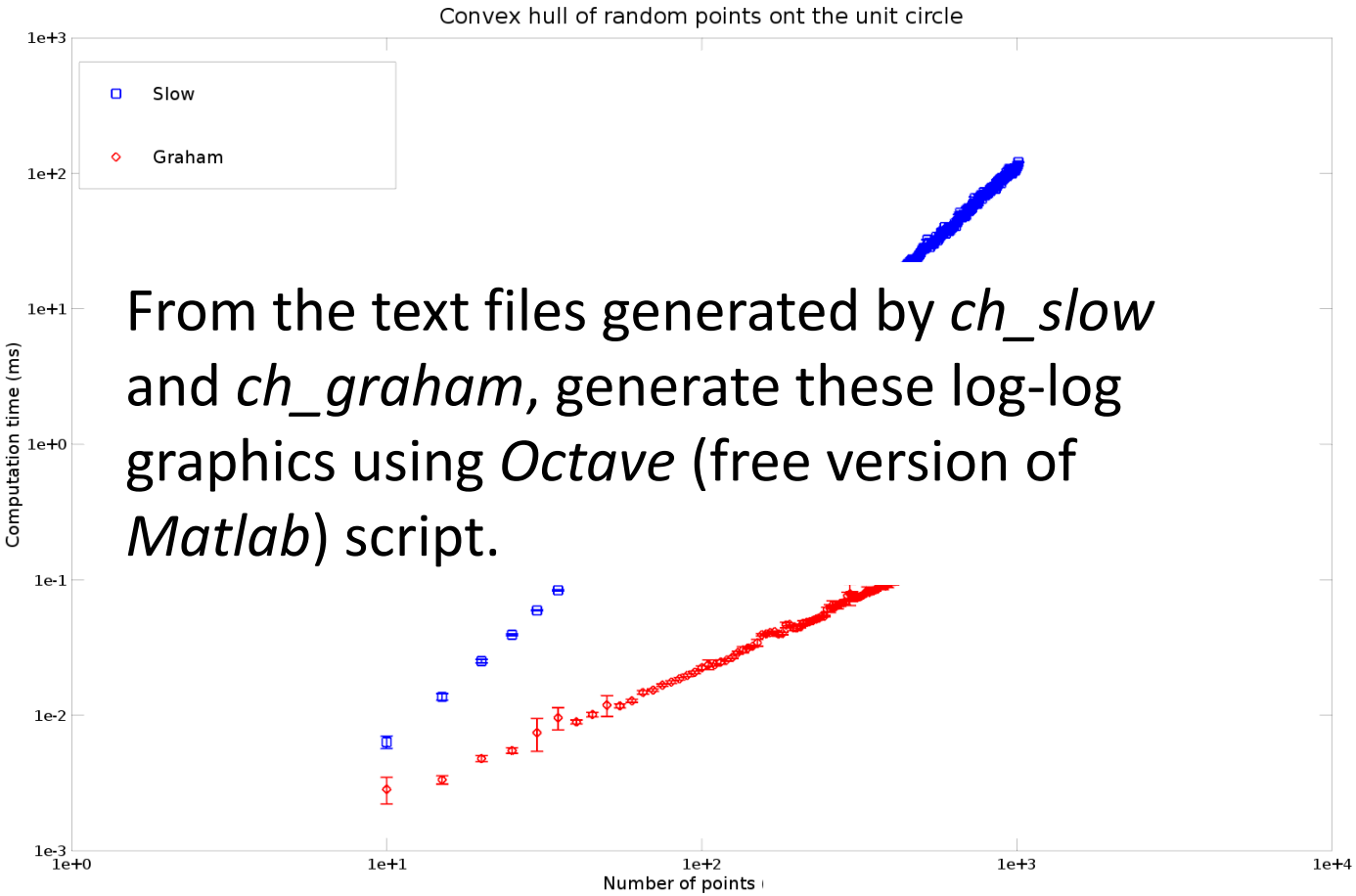
Convex hull of a set of points: complexity of both algorithms

- Test 2: random points on the unit circle



Convex hull of a set of points: complexity of both algorithms

- Test 2: random points on the unit circle



From the text files generated by *ch_slow* and *ch_graham*, generate these log-log graphics using *Octave* (free version of *Matlab*) script.

Convex hull of a set of points: complexity of both algorithms

- Activate tests in Code::Blocks:
  - Project→Set programs' arguments…
  - Select target (*ch_slow* of *ch_graham*)
  - Set the following arguments:

    test* 10 1000 5

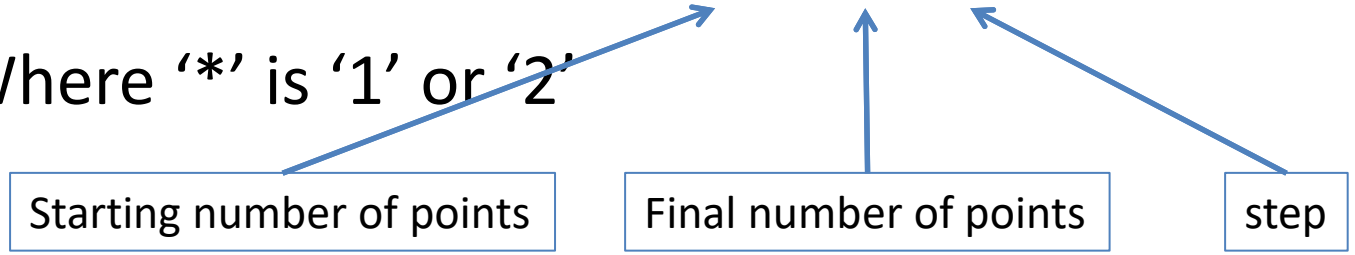
  Where '*' is '1' or '2'

Convex hull of a set of points: complexity of both algorithms

- Activate tests in Code::Blocks:
  - Project→Set programs' arguments…
  - Select target (*ch_slow* of *ch_graham*)
  - Set the following arguments:

test* 10 1000 5

Where '*' is '1' or '2'

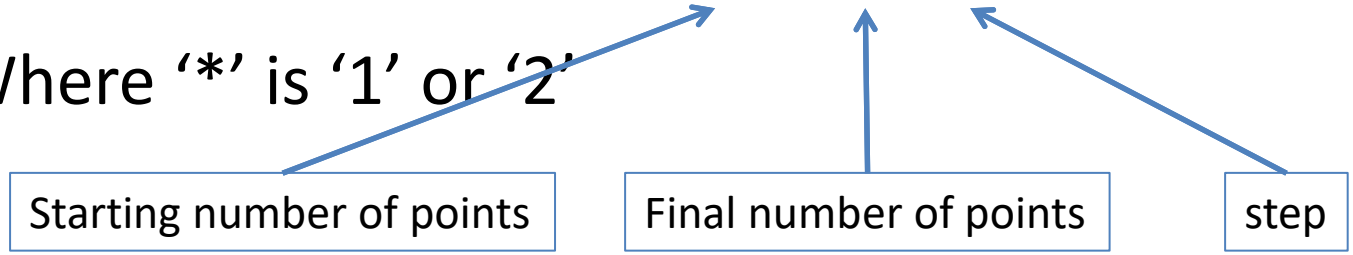


Convex hull of a set of points: complexity of both algorithms

- Activate tests in Code::Blocks:
  - Project→Set programs' arguments…
  - Select target (*ch_slow* of *ch_graham*)
  - Set the following arguments:

test* 10 1000 5

Where '*' is '1' or '2'



Perform test* (1 or 2) with 10, 15, 20, …, 1000 random points. (For each set of points, the convex hull is computed 20 times and the mean and std computational times are computed)

Convex hull of a set of points: complexity of both algorithms

- Output data (use the « *load* » function in Octave):