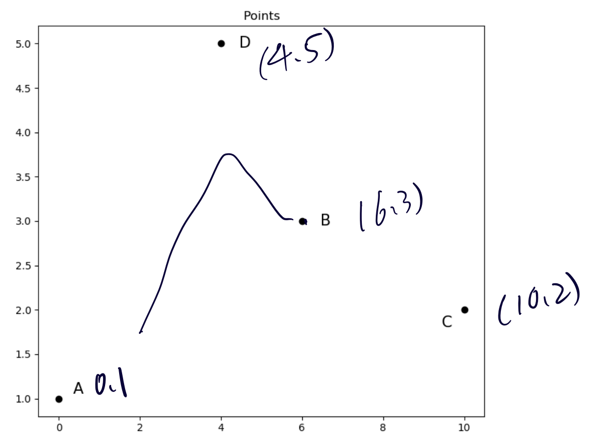
Lab Session (week 2)

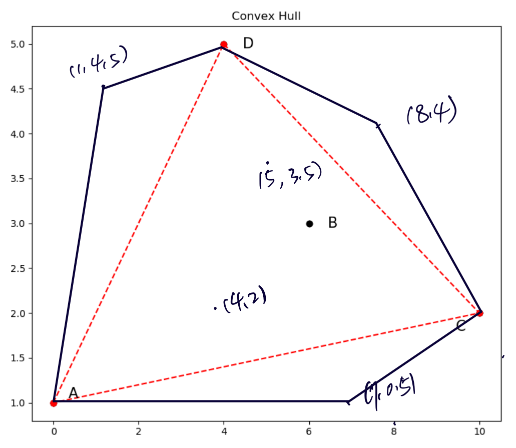
AY 2021/2022

For this lab session, you will have to program a **2-Dimension Convex Hull solver**, using two different methods: gift wrapping and divide-and-conquer. You can download on NTU Learn the template file `convex_hull.py` which contains the code that generates the points and displays the convex hull. You have to write yourself the Python code for three functions, see below.

We refer to the lecture for the notion of Convex Hull (i.e. the smallest convex polygon containing the points). The solver will take as input a cloud of points `pts`, stored as a list of two-entry sublists. For example, for four points $A(0,1)$, $B(6,3)$, $C(10,2)$ and $D(4,5)$, the list `pts` would be:

```
>>> pts = [[0,1], [6,3], [10,2], [4,5]]
```





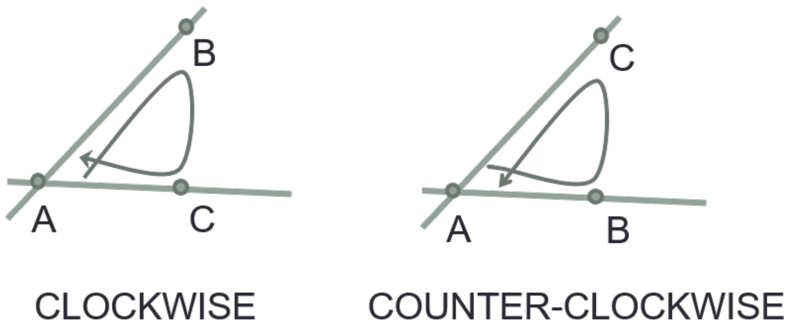
Note that we assume here that the points are randomly distributed and that no three points lie on a common line. The output of the solver will be a list of points, again stored as a list of two-entry sublists, that one encounters when drawing the convex hulls clockwise (the starting point of the list has no importance). With the example above we could have an output such as:

```
>>> hull = [[4,5], [10,2], [0,1]]
```

1 Clockwise Ordering

Listing three points (assumed unaligned) in the plane, we have two different possibilities: either the points are listed in the list in clockwise order or counter-clockwise order. Consider three points $A(ax, ay)$, $B(bx, by)$

and $C(cx, cy)$, in that order. If $(cy - ay) * (bx - ax) < (by - ay) * (cx - ax)$ then (A, B, C) are in clockwise order, otherwise they are in counter-clockwise order[1].



Your first assignment is to implement the function `is_clockwise` that will take as input three points A, B and C and return True if (A, B, C) is in clockwise ordering, False otherwise (see template file `convex_hull.py`).

2 Gift Wrapping Algorithm

A simple algorithm to solve the convex hull problem is the Gift Wrapping strategy: identify a point on the convex hull and from it wrap the convex hull around this point, just like you would wrap a gift. More precisely, identifying a point on the convex hull is easy: you can take for example the leftmost point l (which is necessarily part of the convex hull). Then, in order to find the next point p of the convex hull that comes after l, wrapping clockwise, you simply go through all the points and check which one is such that the line (l, p) leaves all the other points on its right hand side. You then repeat this process starting from p. etc. until you reach point l again. In order to check if a point p' is located on the right hand side of the line (l, p), you can test if the triplet (l, p, p') is clockwise.

Check this animation from Wikipedia that depicts the Gift Wrapping algorithm in action: `www.wikipedia.org/wiki/File:Animation_depicting_the_gift_wrapping_algorithm.gif`

Your second assignment is to implement the function `convex_hull_2d_gift_wrapping` that will compute the convex hull using the Gift Wrapping algorithm. It takes as input a cloud of points and returns its convex hull (see template file `convex_hull.py`). In order to help you, a plot of your cloud of points and your computed convex hull is automatically generated.

What is the complexity of this algorithm, according to the number of points n and the number h of edges in the final convex hull ?

3 Divide-and-Conquer Algorithm

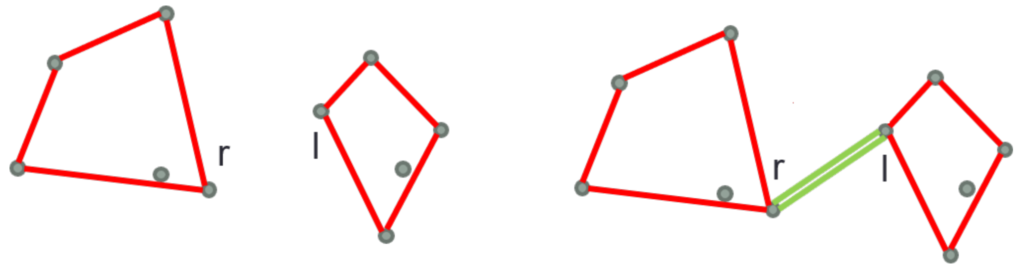
Another efficient algorithm to find the Convex Hull is the Divide-and-Conquer strategy: divide the problem into two smaller problems of equal size and solve them recursively. Once obtained the convex hulls of the

[1]Sketch of proof: assume that A is the leftmost point: $ax < bx$ and $ax < cx$. Then, (A, B, C) are in clockwise order if the line (A, B) has a larger slope than line (A, C), i.e. when $(by - ay)/(bx - ax) > (cy - ay)/(cx - ax)$. Since both $(bx - ax)$ and $(cx - ax)$ are positive, this translates into: if $(cy - ay) * (bx - ax) < (by - ay) * (cx - ax)$ then (A, B, C) are in clockwise order. The reasoning is exactly the same if A is the rightmost point. However, when A is located in the middle of B and C with regards to x-axis, then (A, B, C) are in clockwise order if the line (A, B) has a smaller slope than line (A, C). Yet, this still leads to $(cy - ay) * (bx - ax) < (by - ay) * (cx - ax)$ when (A, B, C) are in clockwise order, since one of $(bx - ax)$ or $(cx - ax)$ is negative, thus flipping the inequality direction.

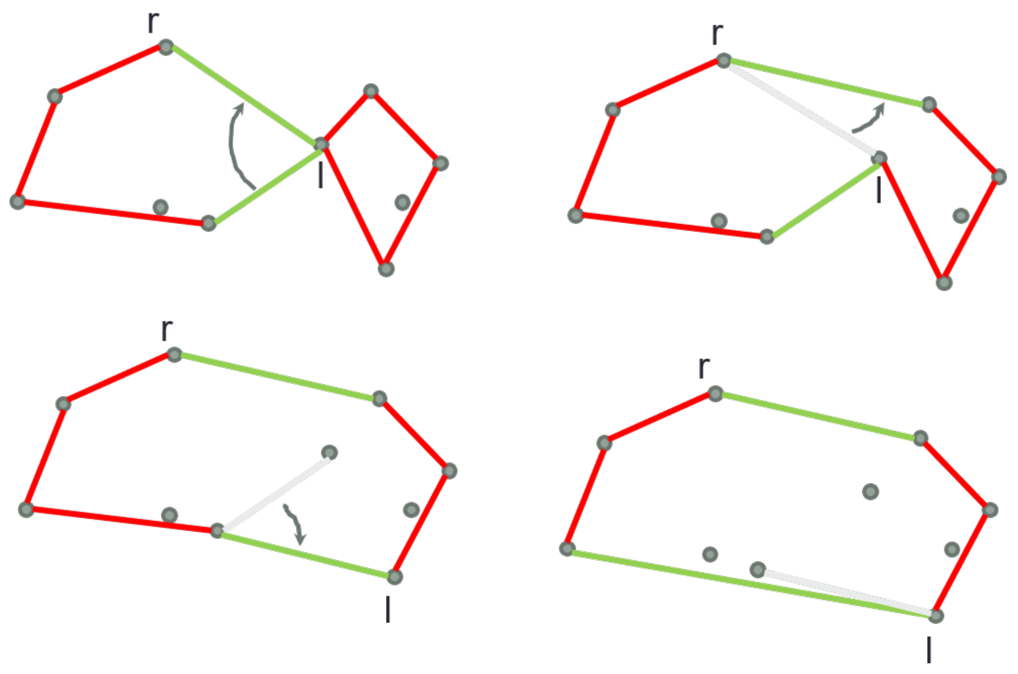
two sub-problems, merge them together to compute the convex hull of the original problem. We refer to the lecture for its general description.

The division into two sub-problems can be done according to the x coordinates of the points. For the base case of the divide-and-conquer recursion (for example as soon as the problem size is smaller or equal to 4 points), you can simply reuse the function `convex_hull_2d_gift_wrapping` to output the convex hull of these small instances.

The merging phase is slightly technical: once obtained the left and right convex hulls L and R, we connect them together to create a bigger hull by linking the rightmost point r of L with the leftmost point l of R (actually connect them twice to create a proper hull). Remark that at this precise moment this new hull is convex, except potentially on the points l and r.



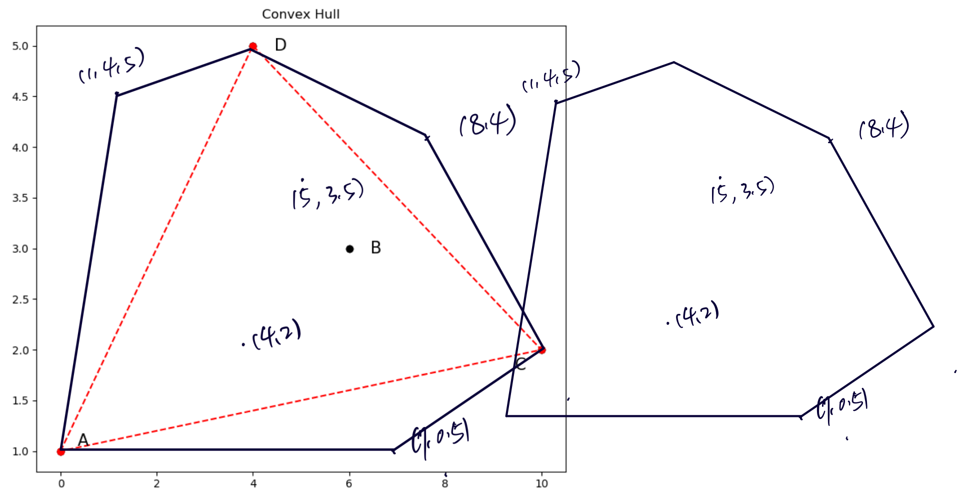
Therefore, perhaps corrections have to be applied: from l, we check if replacing its direct right neighbour p with the next point p' in the hull would lead to a more convex hull (again, this can be checked with verifying the clockwise property of (l, p, p')). If so, we remove p from the hull, so that l and p' are directly connected. We do the same for the left neighbours of l and also for r and its left/right neighbours. We repeat this entire process until no more of such modifications can be done.



Your third and last assignment is to implement the function `convex_hull_2d_divide_conquer` that will compute the convex hull using the Divide-and-Conquer strategy. It takes as input a cloud of points and returns its convex hull (see template file `convex_hull.py`).

What is the complexity of this algorithm, according to the number of points n ?

Once all your functions are properly implemented, you can test your program with much more points (by modifying the variable `NUMBER_OF_POINTS`). For example, you can see that for large enough value, the Gift Wrapping technique will not be able to solve the problem instances.


Convex Hull
D
B
A
C
(1,4.5)
(8,4)
(5,3.5)
(4,2)
(7,0.5)
(1,4.5)
(8,4)
(5,3.5)
(4,2)
(7,0.5)