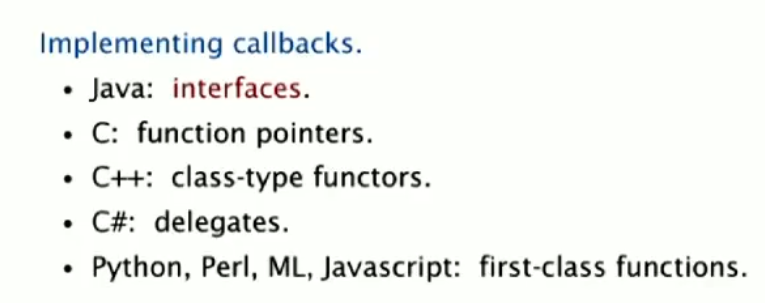
**Introduction**

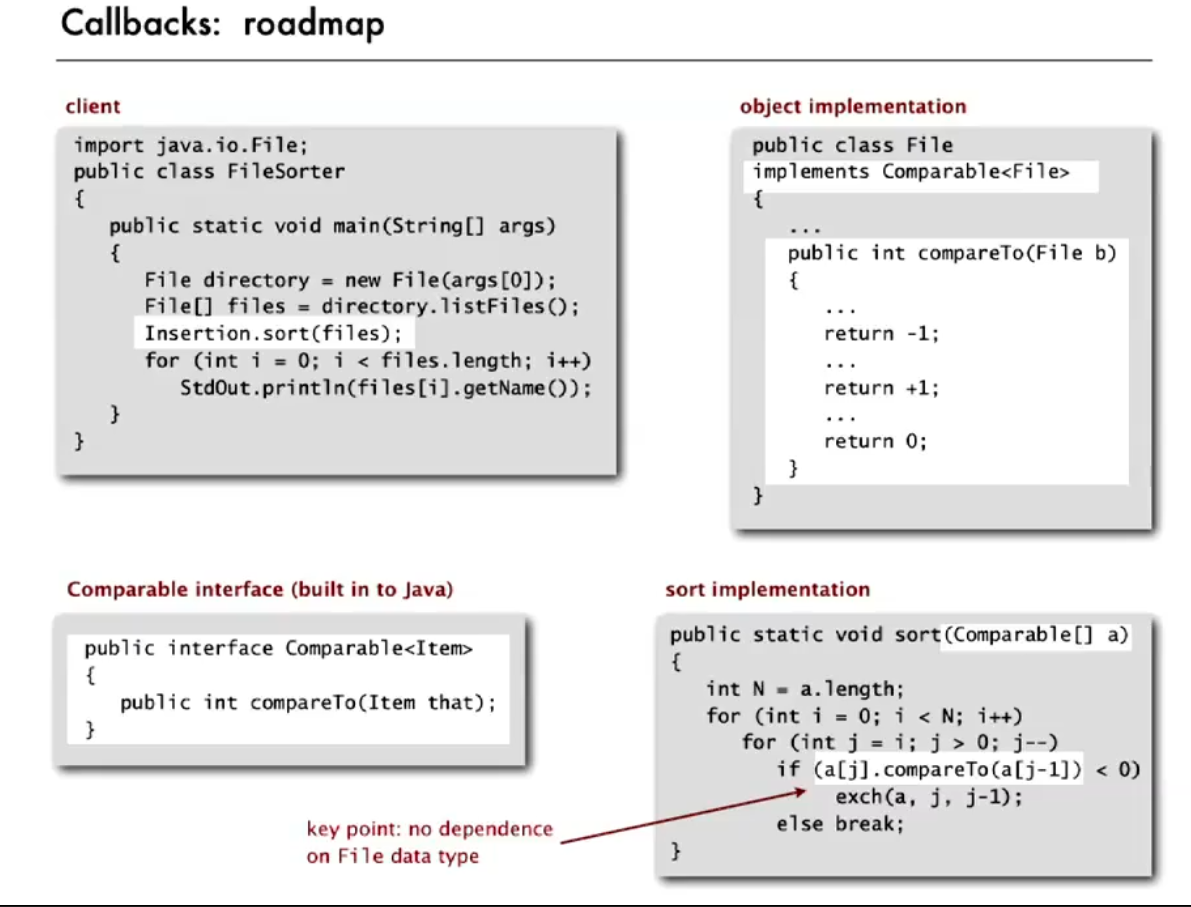
**Call-Back Function:** Basically, when an executable code(functions) is passed as a function parameter it is called call-back function. 

* How do sort() function works for every generic type in java?

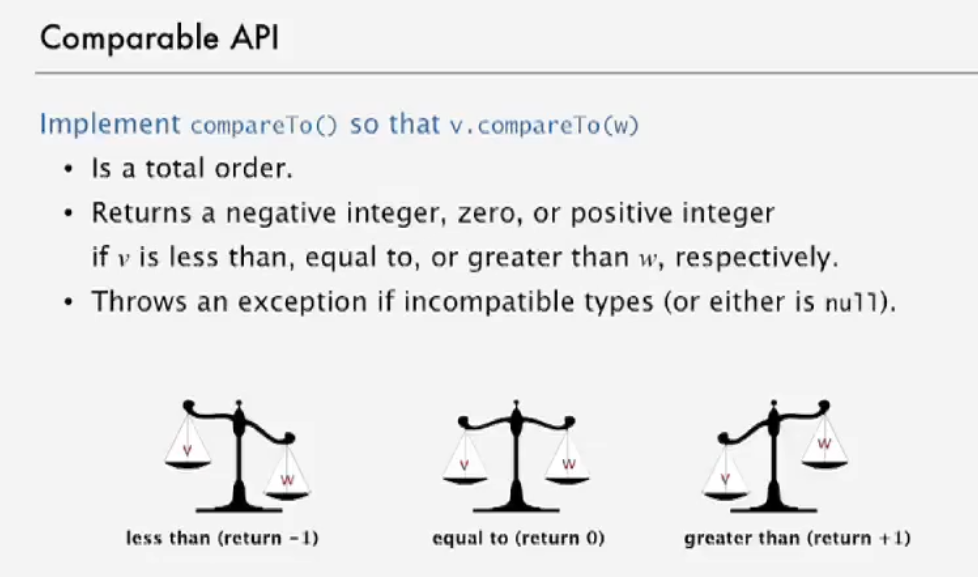
Client passes an array of object to the sort function. Then the sort function in turn calls back the object’s compareTo() method.

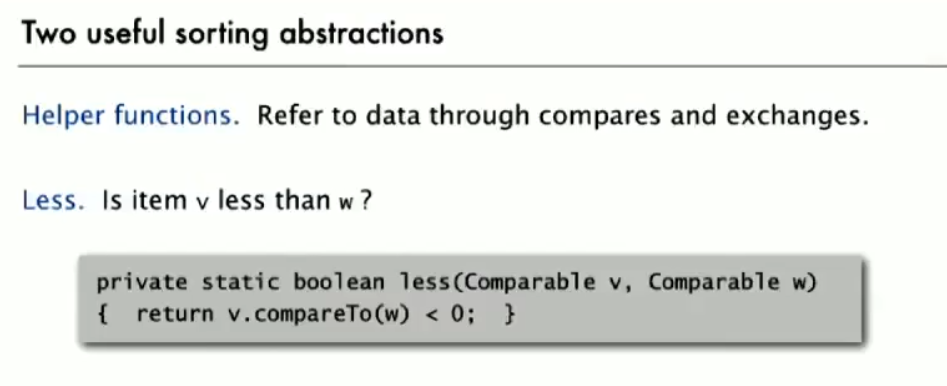
The solution to the callback implementation in java is the **Comparable** interface. It has a method named **compareTo**. Any class that has the sort method, will implement the Comparable interface (in turn overriding the compareTo method for the object and details of the comparison).

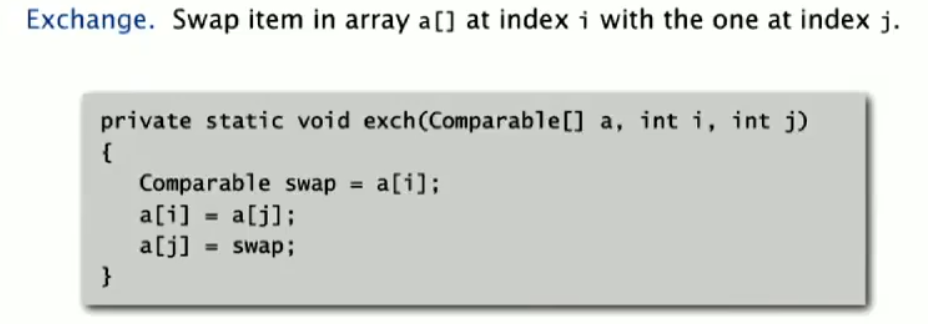
So, the object (for example array, linked list etc) that has sort method, also implements the Comparable interface. So that makes it possible for the static sort method to have Comparable array of object, since it only need that compareTo method. The implementations example is shown below.

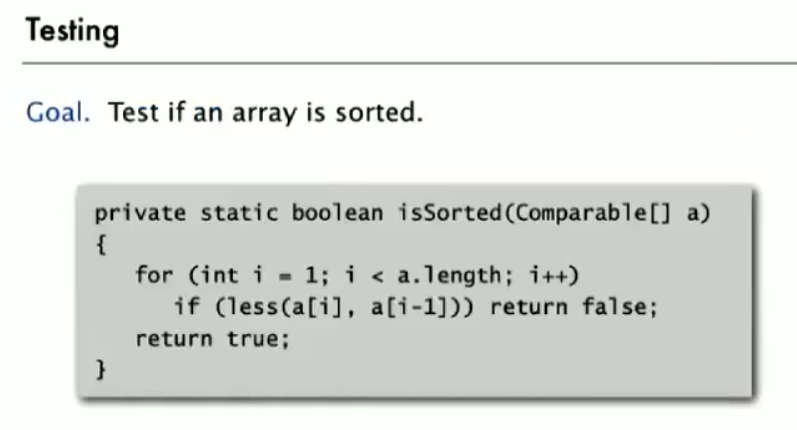


We can implement our own type for the comparable API.

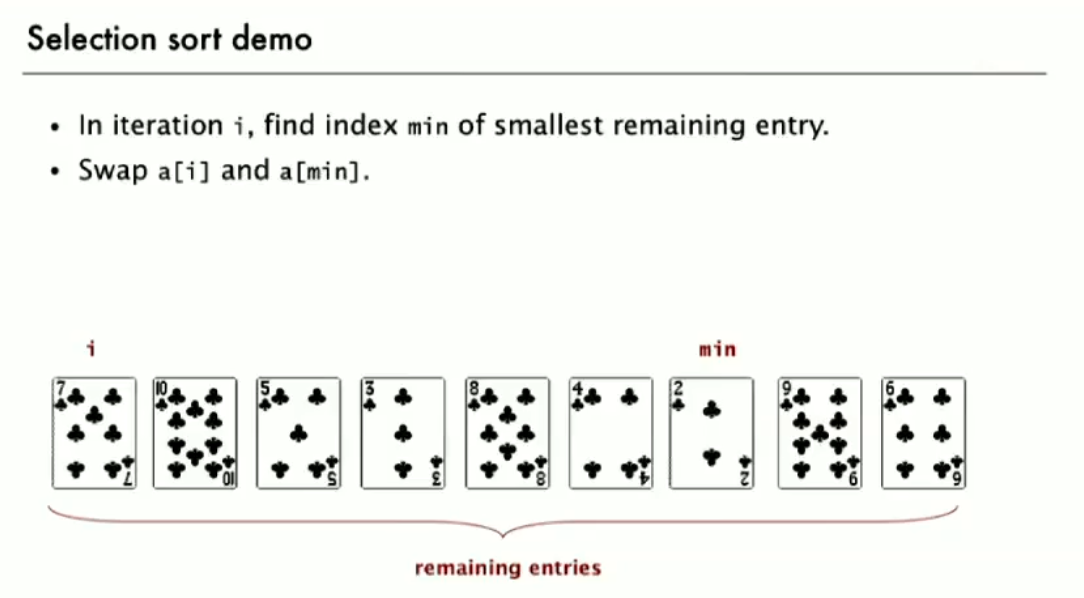


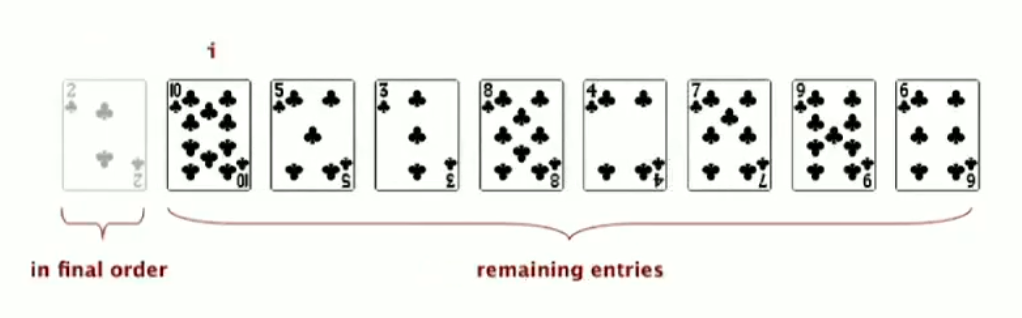


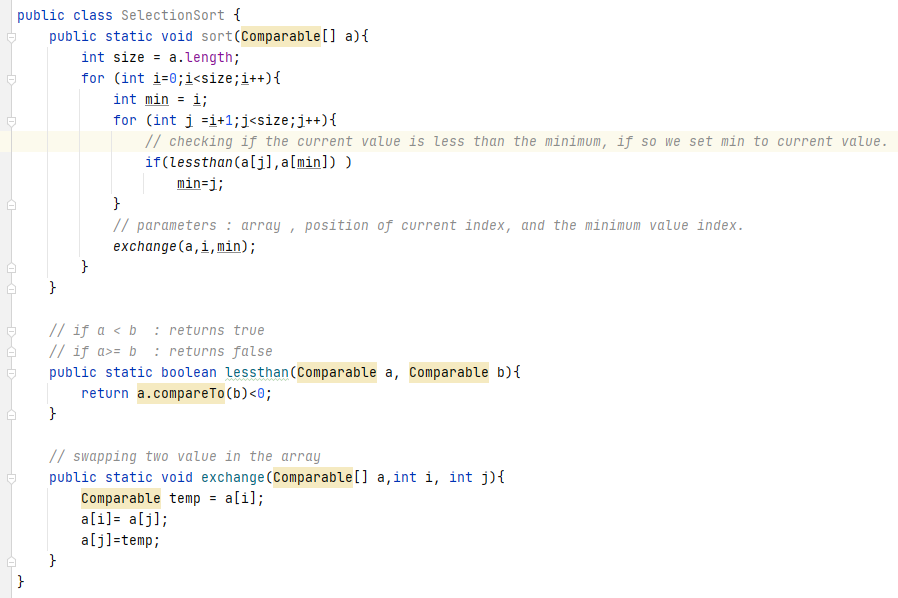




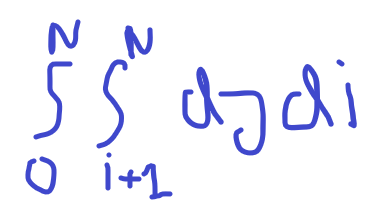
**Selection Sort**







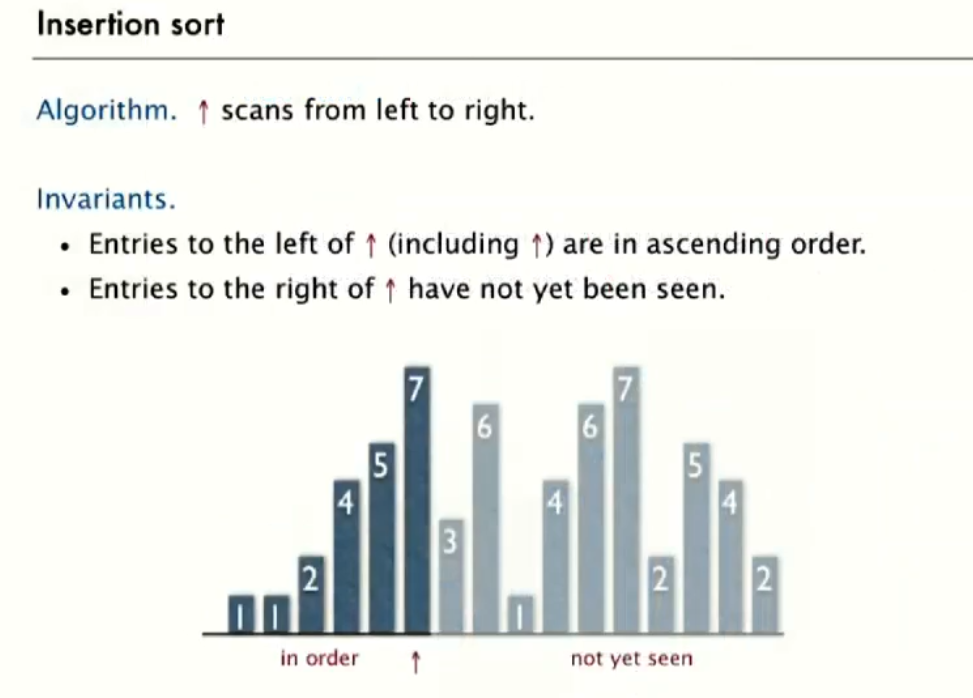
Its time complexity is N^2/2.



**Insertion Sort**

Here we follow a different path than selection sort.

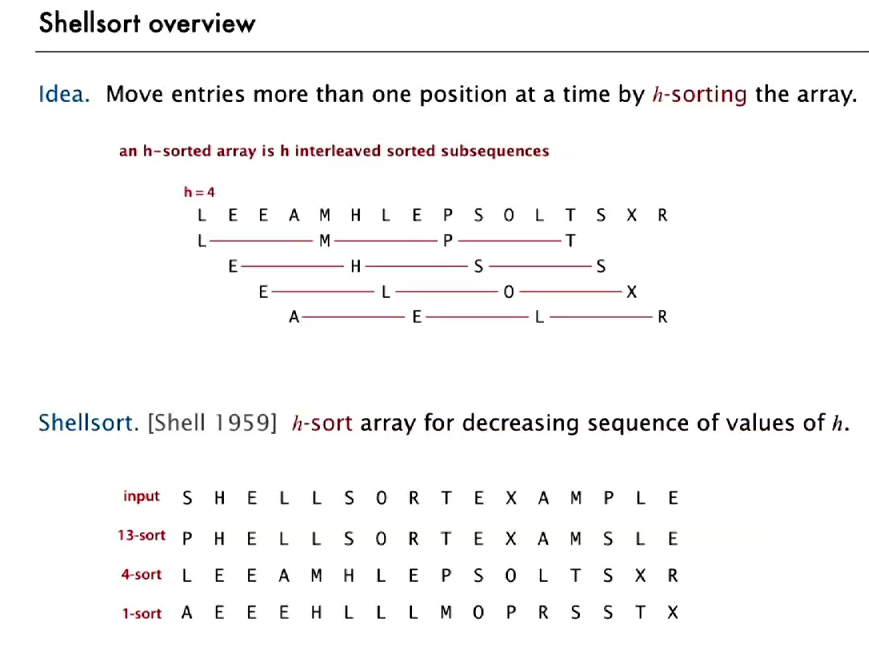
* We assume that in each pass, the entries on the left are in sorted order, we don’t need to do anything about it.
* The entries we process, we compare it to its immediately left entry, if **current < left** we swap current and left, if not then we assume the sorting is complete till this and we move to next entry on the right.



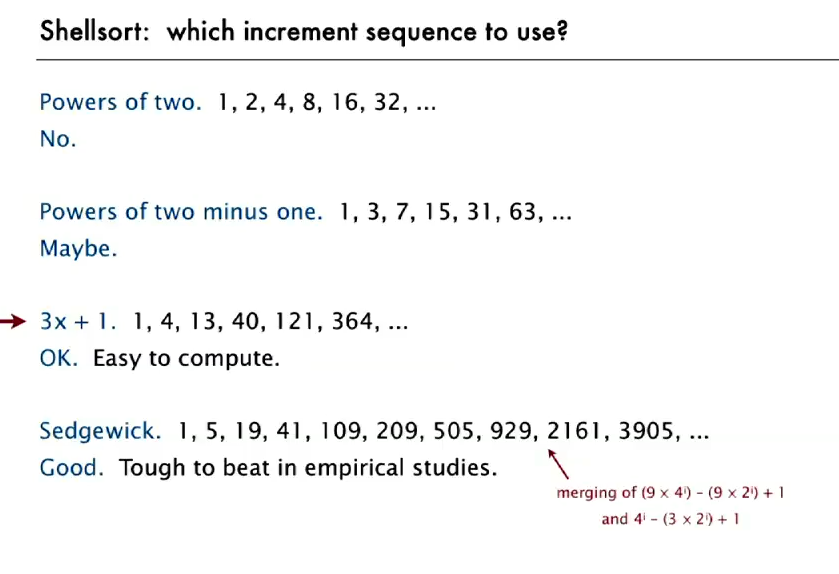
1. Its best case is if the array is already sorted. All it does is it validates the sorting N-1 times.
2. Its worst case is if the array is in descending order and no duplicates insertion sort makes ½ N^2 compare and same number of exchanges.
3. For partially sorted arrays, Insertion sort runs in nearly linear time.

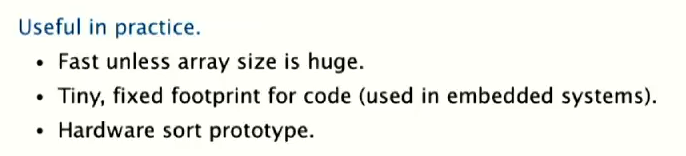
**Shell Sort**

* It’s an improvement over insertion sort. The reason is, in insertion sort the entry only move one place at a time even if it has to travel all the way back. The idea behind shell sort is to move it more than one position at a time by **h-sorting** the array.



* And h-sorted array is h different interleaved sorted subsequence. Basically, instead of sorting each member, we sort each member after h place. For example, a 7-sort array would be when we sort 0th element to its 7th element and so on.
* To perform on h-sort, we actually use insertion sort, but instead of going 1 step back, we go h step back here.
* In this sense, insertion sort is basically 1-sorted array.
* The advantage shell sort is, when the increament is big(h is big), any sorting will work well since its small sub-array. And when we get to smaller increament, because we sorted for bigger increament already array will be nearly sorted. In that case, insertion sort works close to linear time.

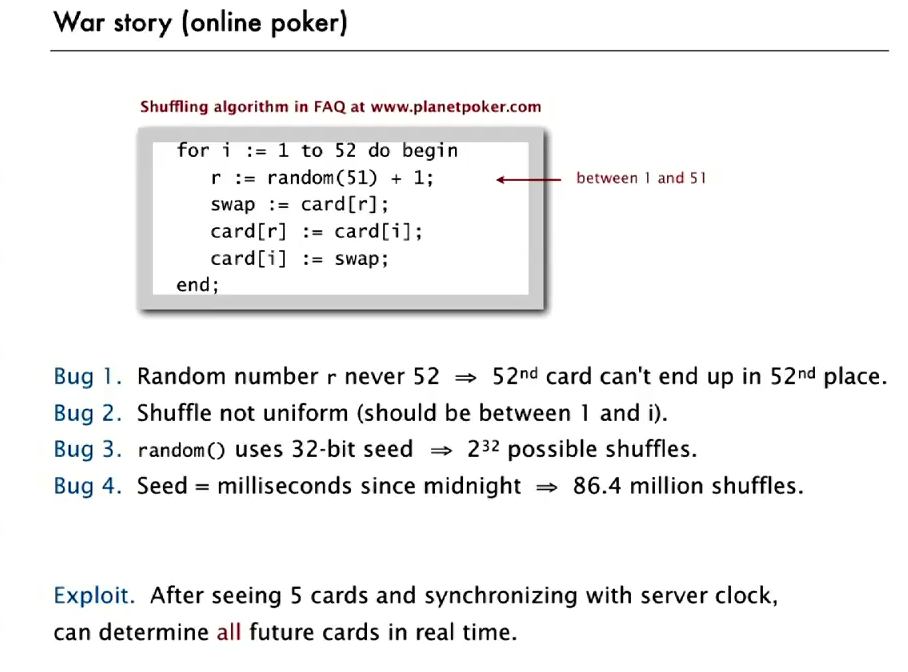




* 3x+1 approach has worst time complexity of N^3/2
* It works perfectly for medium sized array.

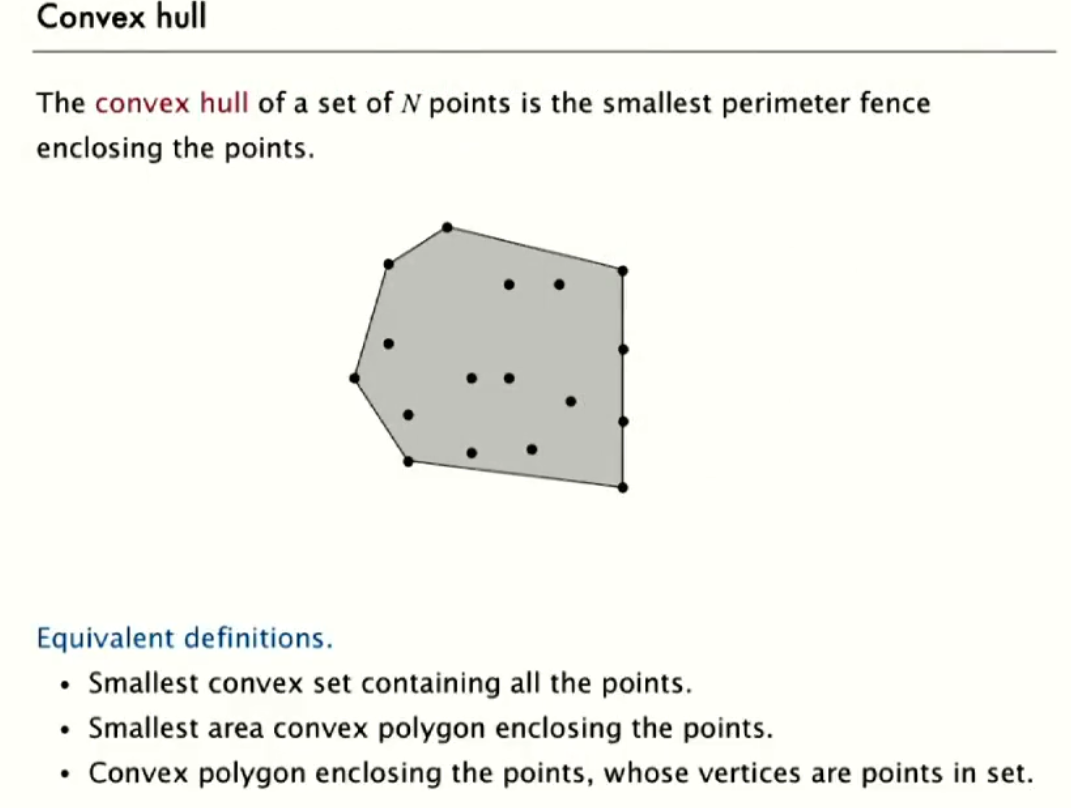
**Shuffle Sort**

* It is basically shuffling an array in random order. To achieve this, we can assign a random number to each entry of the array and sort it according to that numbers and get a shuffled array. But as this requires sorting it is a lot of work
* To solve this problem, shuffle sort works wonders. It’s almost linear time complexity. How it works is, it loops through entire array, and each time swaps the current entry with a random entry on its left.
* The key is to pick a uniform random number between **0** and **i.** There is a common bug here that programmers choose to pick a random number from whole array, but that won’t be uniform. Another variant can be choosing a random number between **i** and **(size-1).**

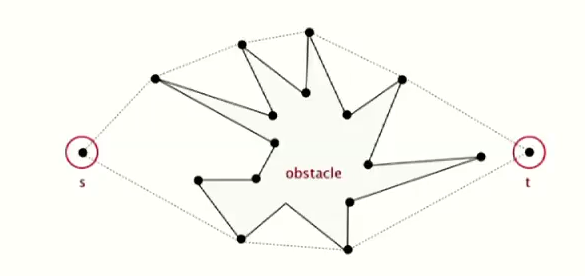


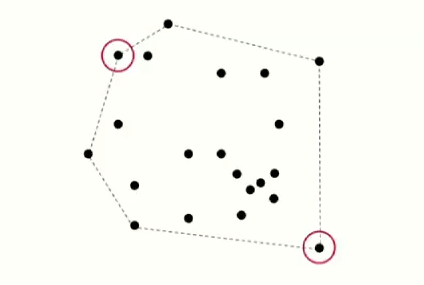
**Convex Hull**

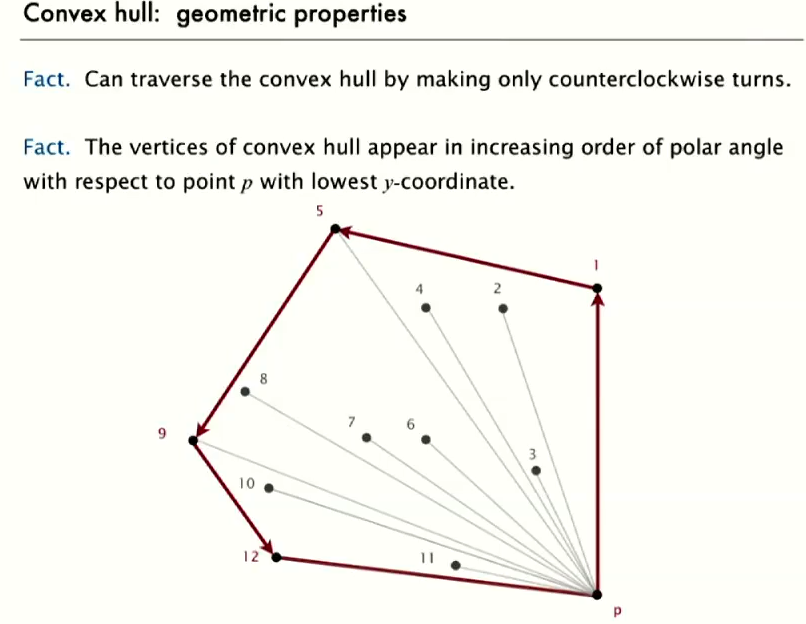
* If we are given a set of points, convex hull is the smallest perimeter that closes the points. The picture below is clearer about this.



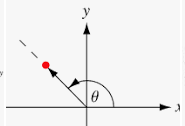
Some practical application:

* In case a robot wants to move from point a to b, and there is bunch of obstacles in between the points (obstacles can be taken as the point object). So, the shortest path would be one of the two polygonal chains of the convex hull. 
* For N number of given points, finding pair of points which are the furthest apart

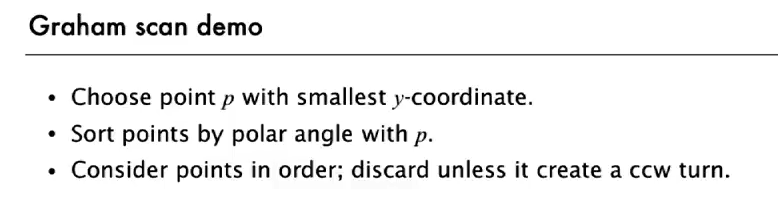
. 



Note: In the plane, the polar angle is the counterclockwise angle from the x-axis at which a point in the negative plane lies.



To Solve this problem, we will use the Graham Scan algorithm.



Note: for details for counter clockwise computation theory, check the slide.

