Advanced Algorithms

Geometric Algorithm

Jarvis's march Approach

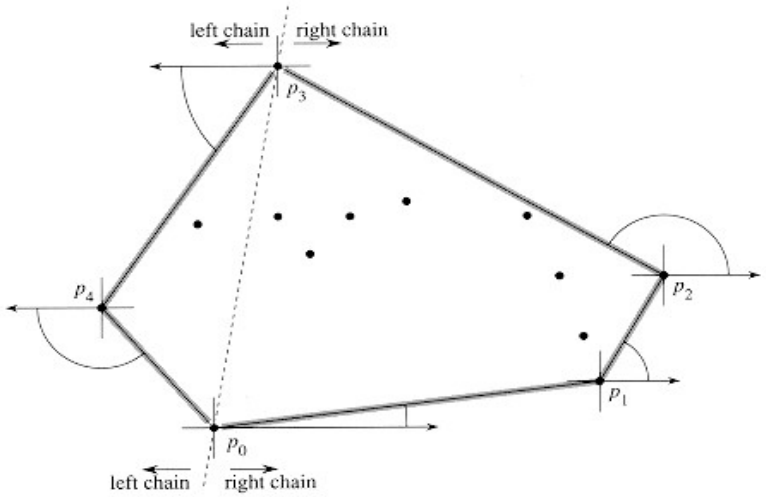
Jarvis's march builds a sequence $H = p_0, p_1, \ldots, p_h\text{-}1$ of the vertices of CH(Q).

We start with p_0. The next convex hull vertex p_1 has the least polar angle with respect to p_0. (In case of ties, we choose the point farthest from p_0.)

Similarly, p_2 has the least polar angle with respect to p_1, and so on. When we reach the highest vertex, say p_k (breaking ties by choosing the farthest such vertex), we have constructed the ***right chain*** of CH(Q).

To construct the ***left chain***, we start at p_k and choose p_k+1 as the point with the least polar angle with respect to p_k, but *from the negative x-axis*. We continue on, forming the left chain by taking polar angles from the negative x-axis, until we come back to our original vertex p_0.

Jarvis's march Approach



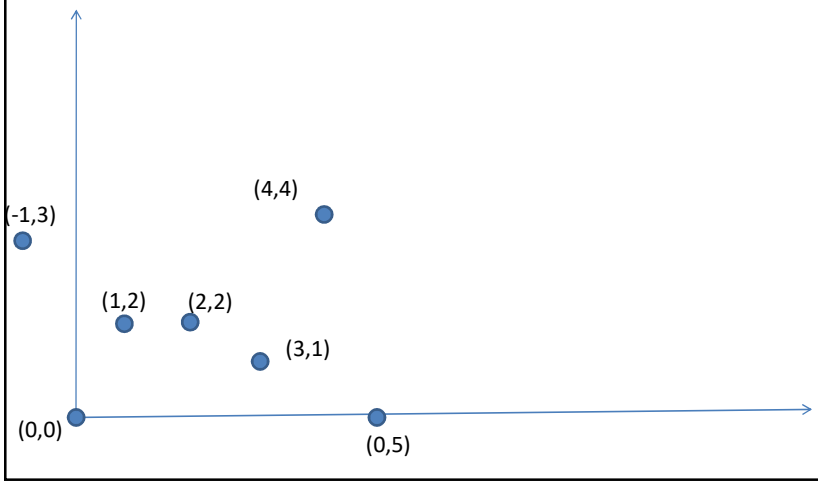
Jarvis's march Approach

Jarvis's march computes the convex hull of a set Q of points by a technique known as ***package wrapping*** (or ***gift wrapping***). The algorithm runs in time $O(nh)$, where h is the number of vertices of CH(Q).

When h is $o(\lg n)$, Jarvis's march is asymptotically faster than Graham's scan.

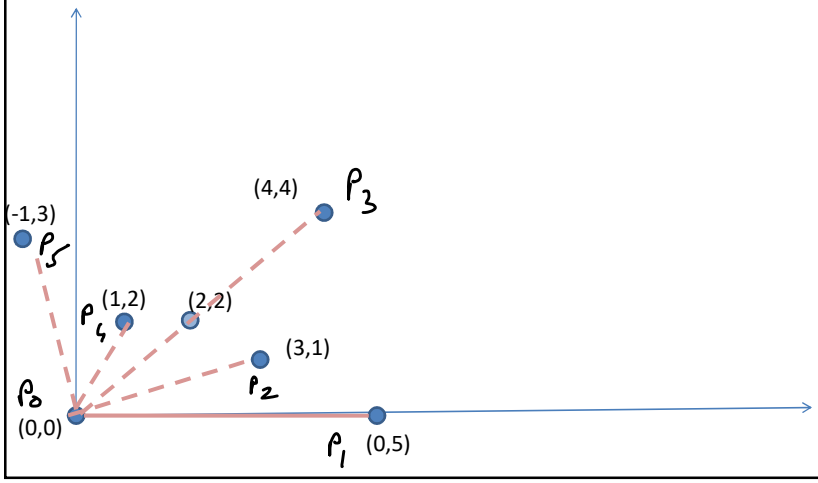
Jarvis's march Algorithm

{{0,0},{3,1},{4,4},{1,2},{0,5},{2,2},{-1,3}}



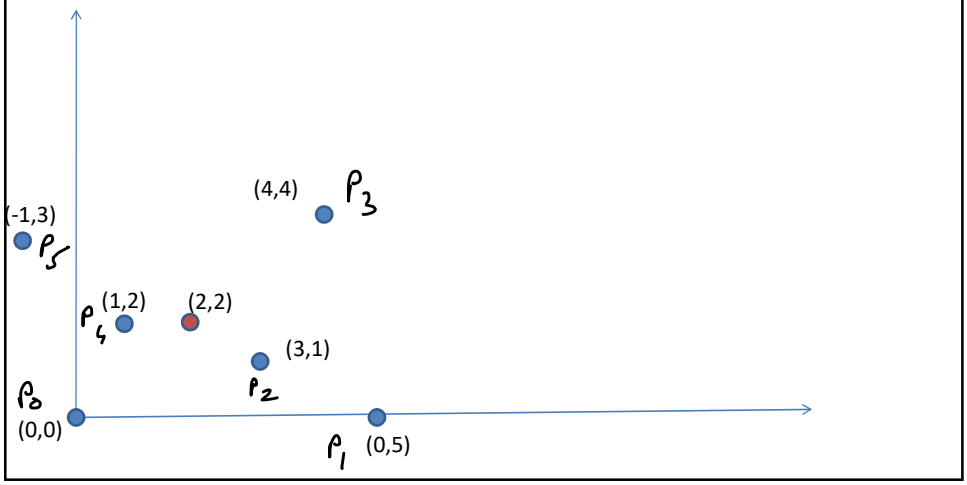
Jarvis's march Algorithm

{{0,0},{3,1},{4,4},{1,2},{0,5},{2,2},{-1,3}}



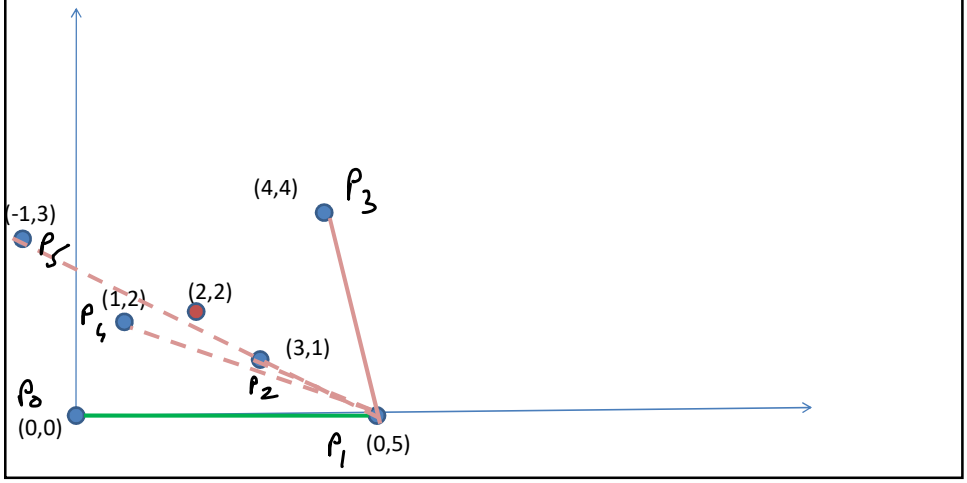
Jarvis's march Algorithm

{{0,0},{3,1},{4,4},{1,2},{0,5},{2,2},{-1,3}}



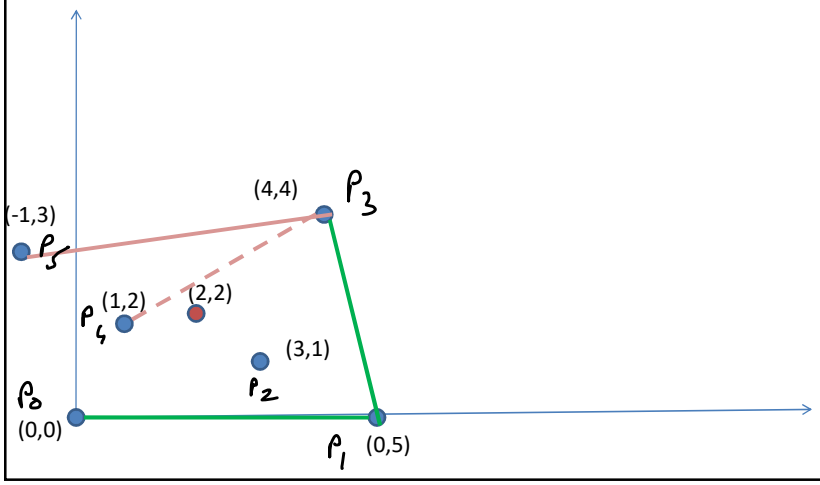
Jarvis's march Algorithm

{{0,0},{3,1},{4,4},{1,2},{0,5},{2,2},{-1,3}}



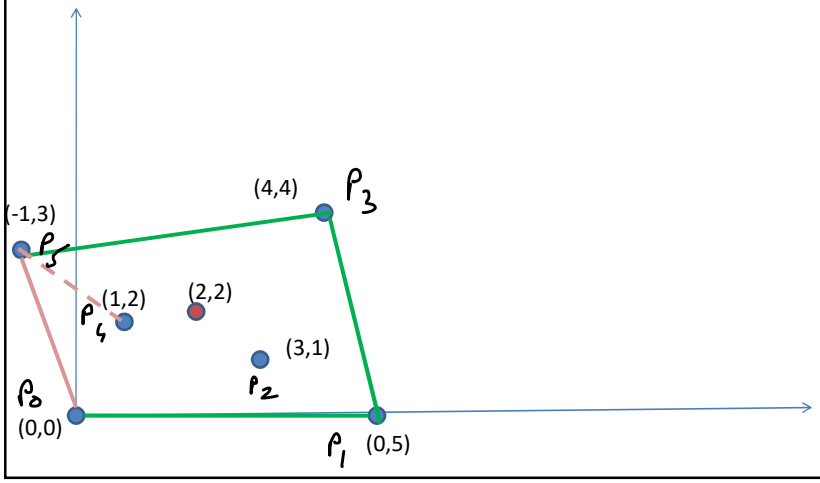
Jarvis's march Algorithm

{{0,0},{3,1},{4,4},{1,2},{0,5},{2,2},{-1,3}}



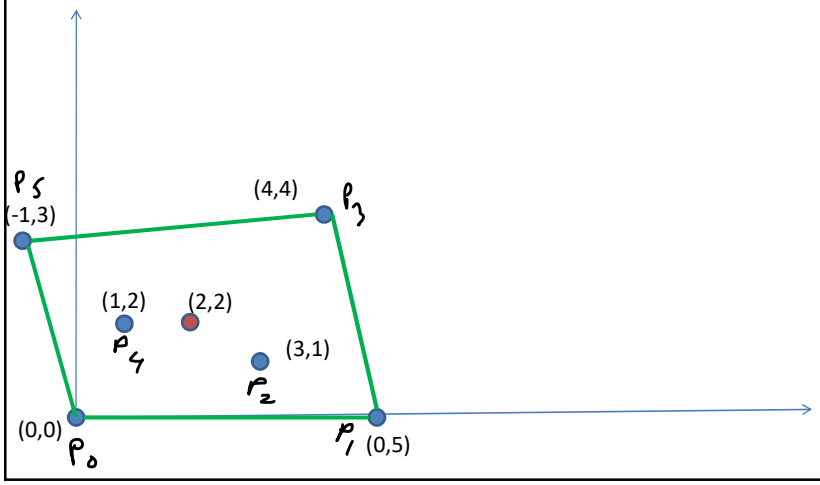
Jarvis's march Algorithm

{{0,0},{3,1},{4,4},{1,2},{0,5},{2,2},{-1,3}}

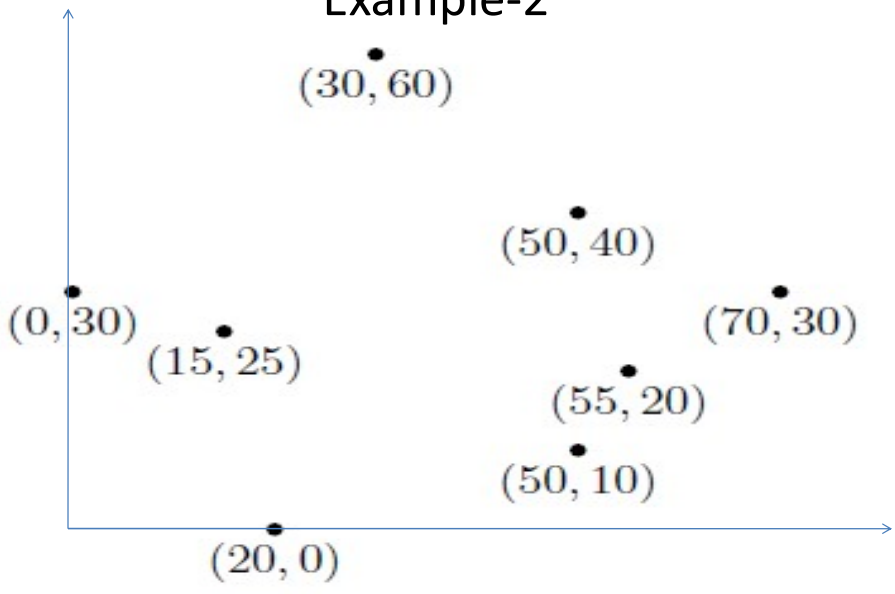


Jarvis's march Algorithm

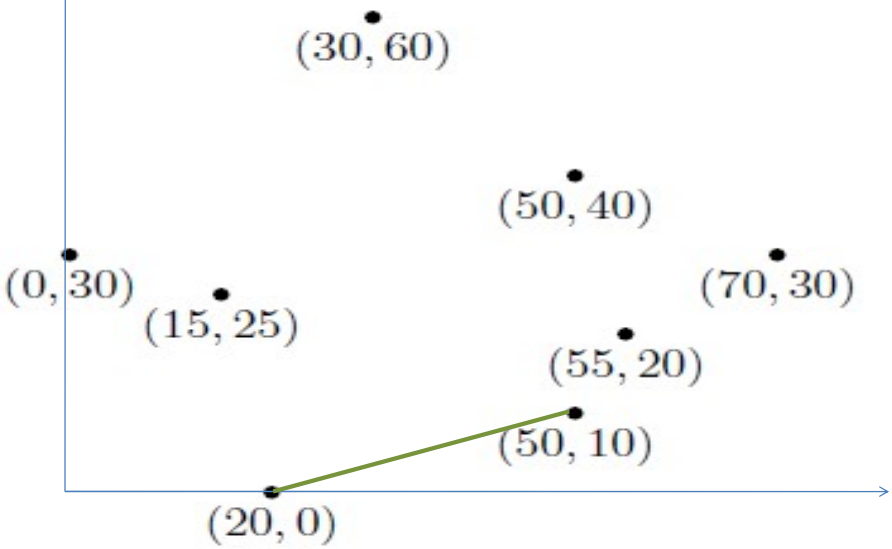
{{0,0},{3,1},{4,4},{1,2},{0,5},{2,2},{-1,3}}



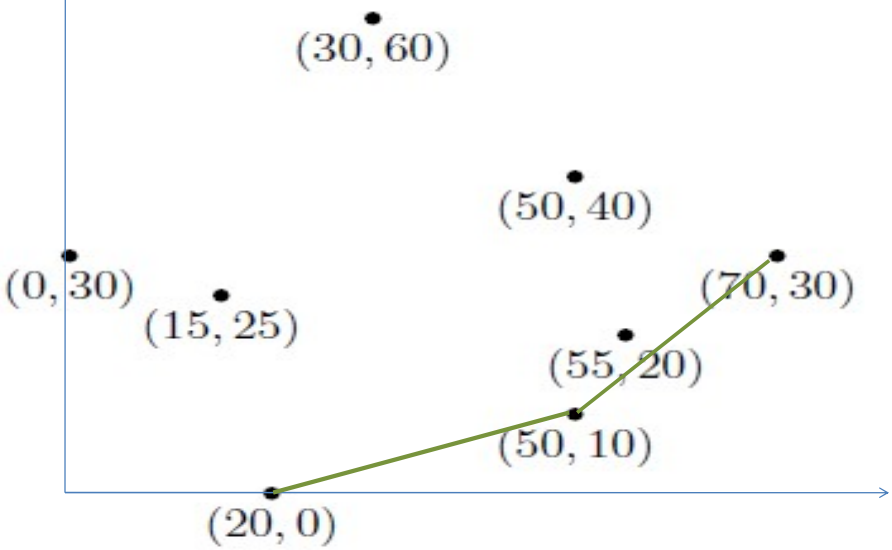
Example-2

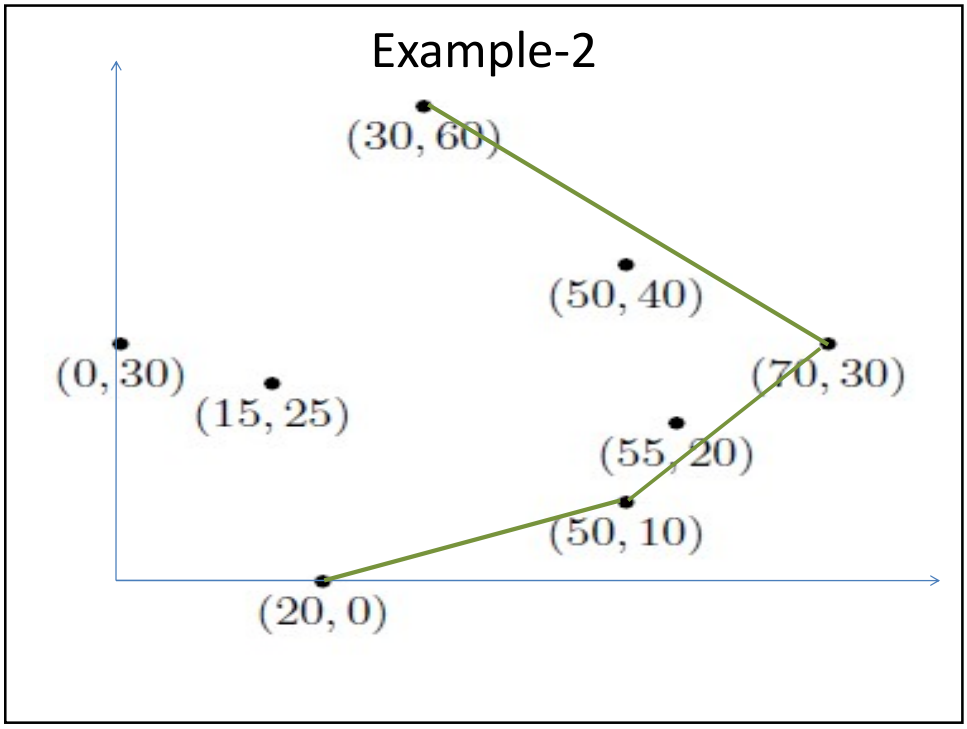


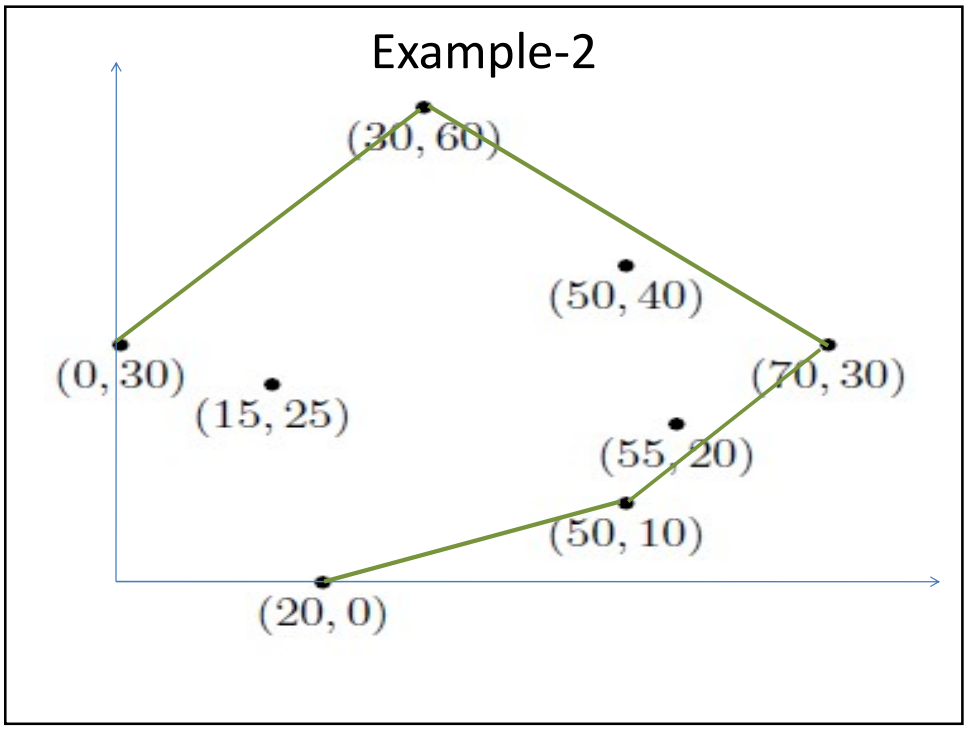
Example-2



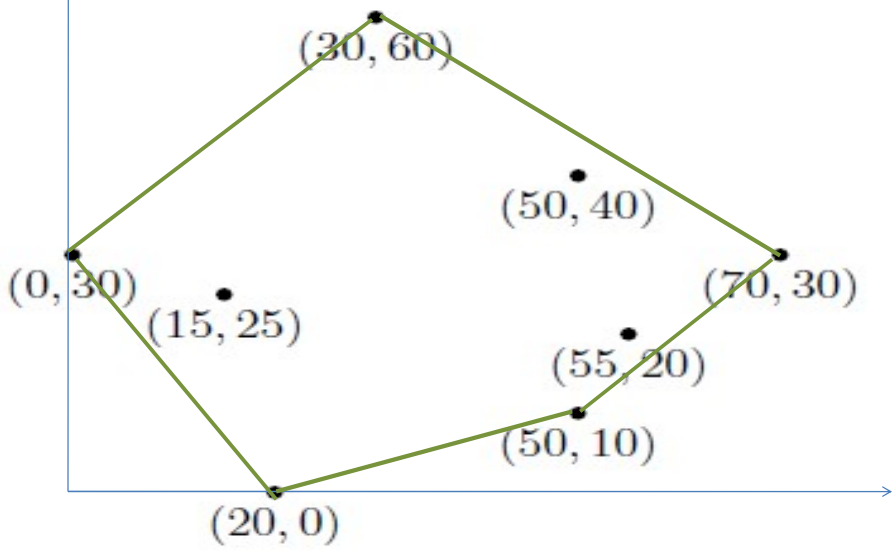
Example-2




Example-2
(30, 60)
(50, 40)
(0, 30)
(70, 30)
(15, 25)
(55, 20)
(50, 10)
(20, 0)


Example-2
(30, 60)
(50, 40)
(0, 30)
(70, 30)
(15, 25)
(55, 20)
(50, 10)
(20, 0)

Example-2



Convex Hull (Divide & Conquer)

Divide and Conquer is a popular technique for algorithm design. We use it here to find the convex hull. The first step is a Divide step, the second step is a Conquer step, and the third step is a Combine step.

The idea is to:

Divide and conquer

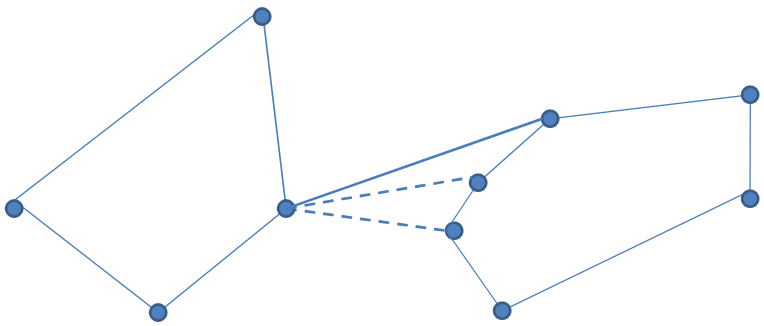
1. Divide the n points into two halves.
2. Find convex hull of each subset.
3. Combine the two hulls into overall convex hull.

Convex Hull (Divide & Conquer)

It helps to work with convex hulls that do not overlap. To ensure this, all the points are *presorted* from left to right. So we have a left and right half, and hence a left and right convex hull.

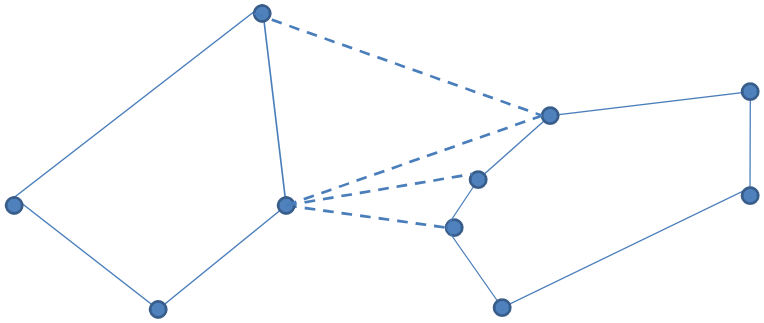
Define a *bridge* as any line segment joining a vertex on the left and a vertex on the right that does not cross the side of either polygon. What we need are the *upper* and *lower* bridges. The following produces the upper bridge.

Convex Hull (Divide & Conquer)



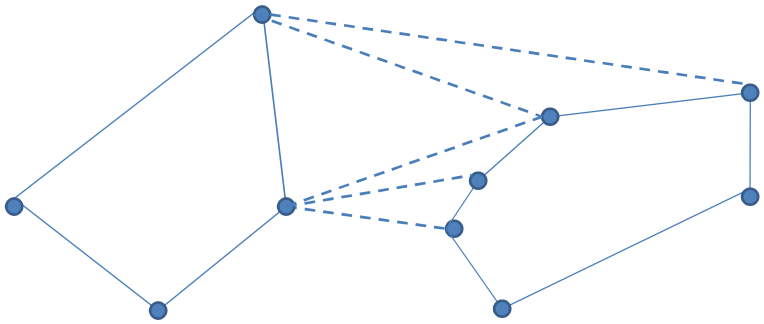
Calculation of Upper Bridge

Convex Hull (Divide & Conquer)



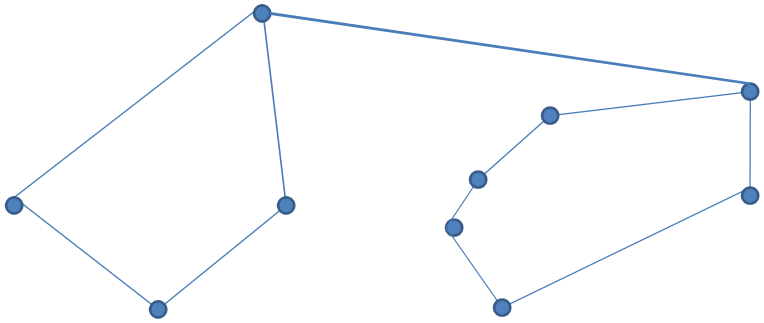
Calculation of Upper Bridge

Convex Hull (Divide & Conquer)



Calculation of Upper Bridge

Convex Hull (Divide & Conquer)

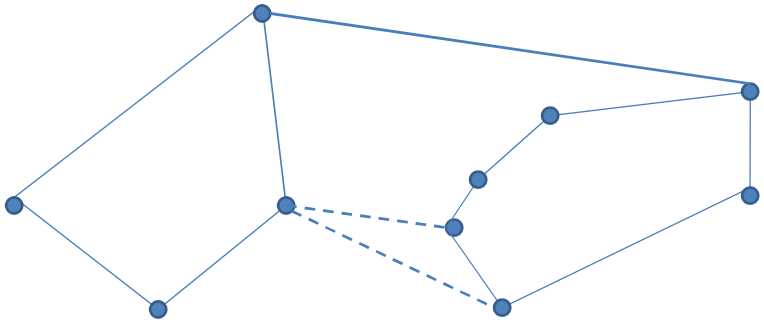


Calculation of Upper Bridge

Convex Hull (Divide & Conquer)

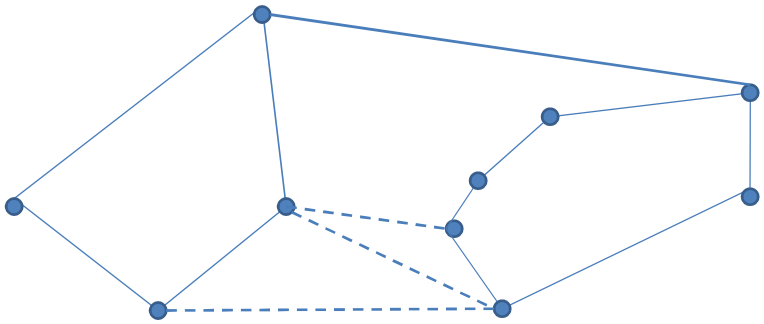
1. Start with any bridge. For example, a bridge is guaranteed if you join the rightmost vertex on the left to the leftmost vertex on the right.
2. Keeping the left end of the bridge fixed, see if the right end can be raised. That is, look at the next vertex on the right polygon going clockwise, and see whether that would be a (better) bridge. Otherwise, see if the left end can be raised while the right end remains fixed.
3. If made no progress in Step 2 (cannot raise either side), then stop else repeat Step 2.

Convex Hull (Divide & Conquer)



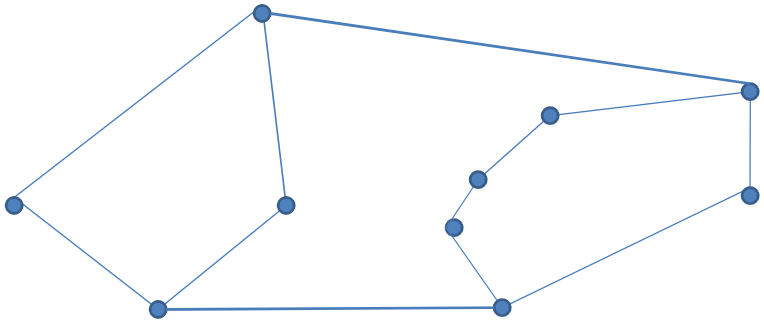
Calculation of Lower Bridge

Convex Hull (Divide & Conquer)



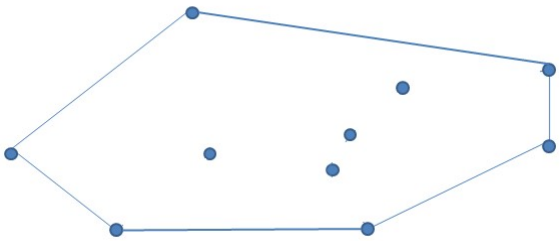
Calculation of Lower Bridge

Convex Hull (Divide & Conquer)



Calculation of Lower Bridge

Convex Hull (Divide & Conquer)



Convex Hull (Divide & Conquer)

Now, we need to determine the running time of the algorithm. The key is to perform Step 2 in constant time. For this it is sufficient that each vertex has a pointer to the next vertex going clockwise and going counterclockwise. Hence the choice of data structure: we store each hull using a *doubly linked circular linked list*.

It follows that the total work done in a merge is proportional to the number of vertices. And as we shall see from a later chapter, this means that the overall algorithm takes time $O(n \log n)$.

Convex Hull: Runtime

- Preprocessing: sort the points by x-coordinate — $O(n \log n)$ just once
- Divide the set of points into two sets **A** and **B**: — $O(1)$
  - **A** contains the left $\lfloor n/2 \rfloor$ points,
  - **B** contains the right $\lceil n/2 \rceil$ points — $T(n/2)$
- Recursively compute the convex hull of **A** — $T(n/2)$
- Recursively compute the convex hull of **B** — $O(n)$
- Merge the two convex hulls

Convex Hull: Runtime

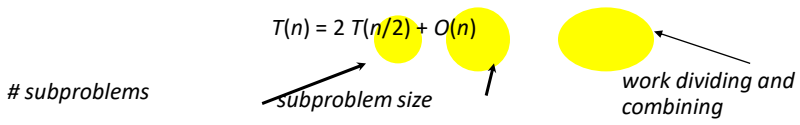
- Runtime Recurrence:

  $T(n) = 2\,T(n/2) + cn$

- Solves to $T(n) = \Theta(n \log n)$

Recurrence
(Just like merge sort recurrence)

1. ***Divide:*** Divide set of points in half.
2. ***Conquer:*** Recursively compute convex hulls of 2 halves.
3. ***Combine:*** Linear-time merge.



Recurrence (cont'd)

$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- How do we solve $T(n)$? I.e., how do we find out if it is O(n) or $O(n^2)$ or …?

Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.

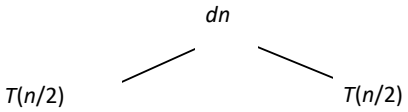
Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.

$T(n)$

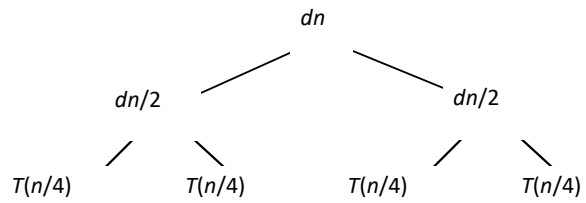
Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



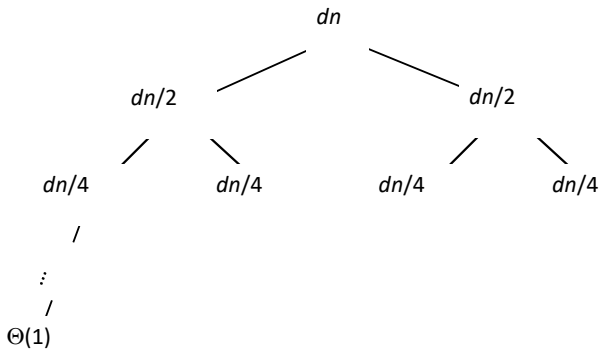
Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



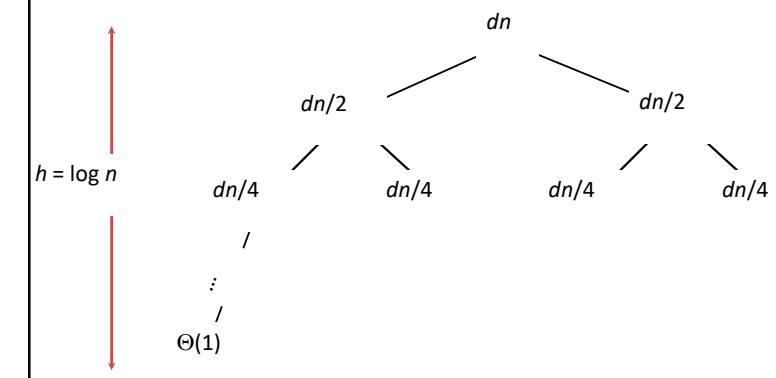
Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



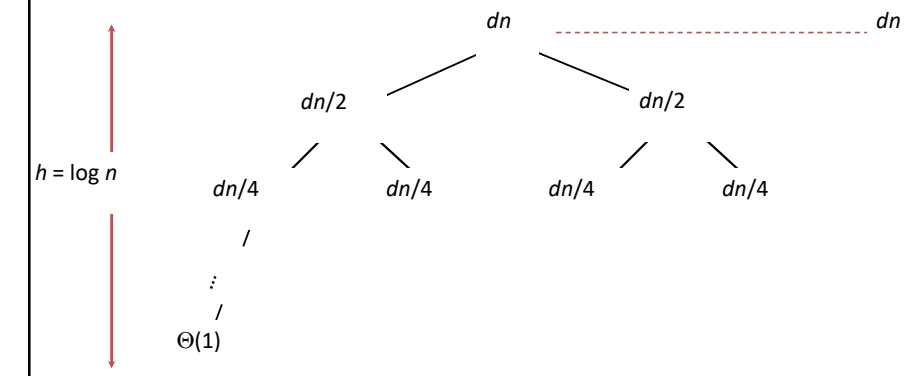
Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



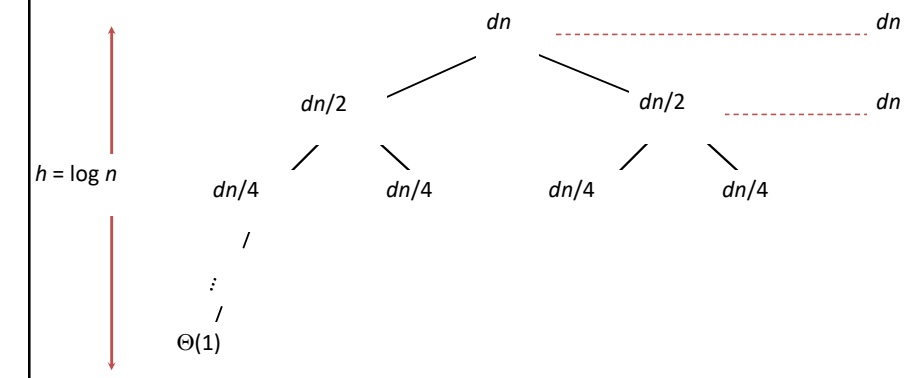
Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



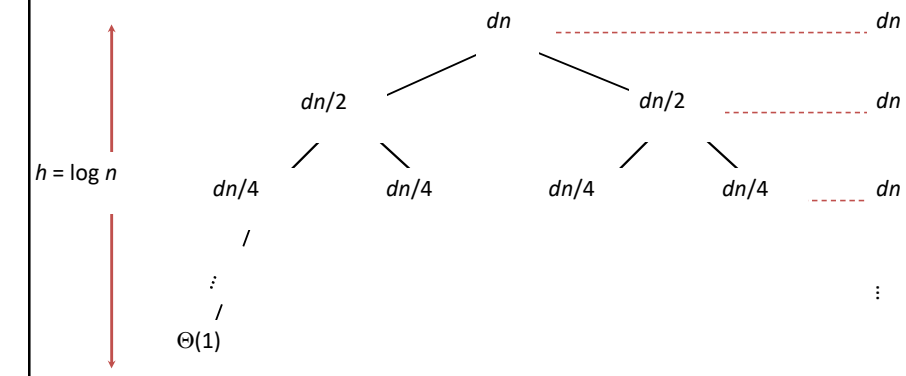
Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



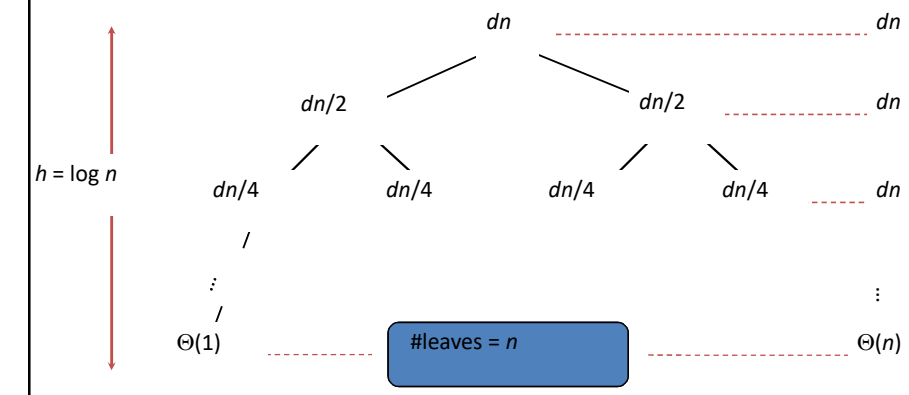
Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.