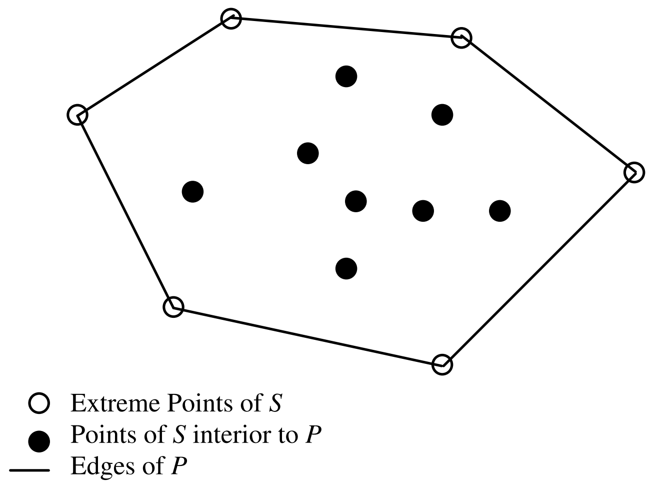
Computational Geometry
(Divide-and-Conquer)

<u>Definition</u>: The **convex hull** of a set *S* of points, denoted *hull*(*S*), is the smallest polygon *P* for which each point of *S* is either on the boundary or in the interior of *P*.

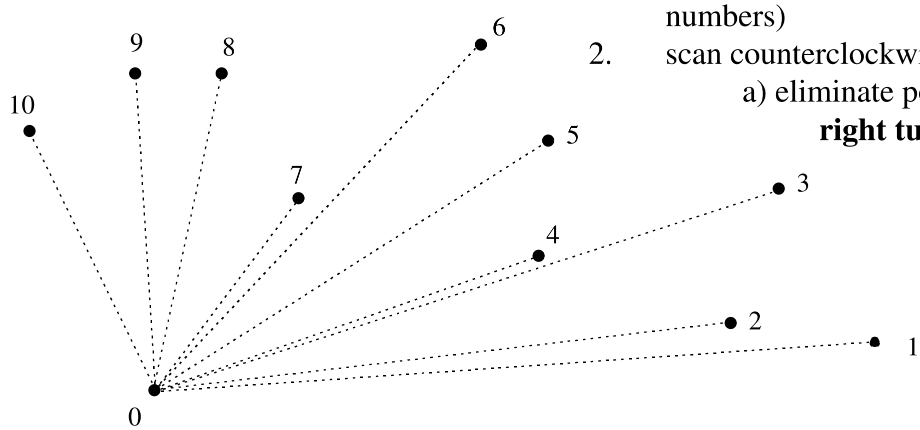


<u>Intuition</u>: If each point is a nail sticking out from a board and you take a rubberband and lower it over the nails, so as to completely encompass the set of nails, and then let the rubberband naturally contract, the rubberband will give you the edges of the convex hull of the set of points, and those nails that correspond to a change in slope of the rubberband represent the extreme points of the convex hull.

<u>Theorem:</u> Sorting is linear-time transformable to the convex hull problem; therefore, identifying the extreme points that represent the convex hull of a set *S* of *n* planar points, requires $\Omega(n \log n)$ time.

<u>Proof</u> (Preparata & Shamos, 1985): Given a set of *n* positive real numbers, $x_1, x_2, \ldots, x_n$, the convex hull algorithm can be used to sort them with only linear overhead, as follows. Corresponding to each number x_i is the point *i*-th point (x_i, x_i^2). Notice that these points all lie on the parabola $y = x^2$. The convex hull of this set consists of a list of the points sorted by *x*-coordinate. One pass through the list will enable us to read off the values of x_i in order. QED

Graham's Scan
(*not* divide-and-conquer)



1. Order the points (given as numbers)
2. scan counterclockwise
   a) eliminate point if makes a **right turn**

Show an **example** of Graham's scan. Show that one goes around the set of points in a counterclockwise fashion, starting at the lowermost point p_0. Each time a point is encountered that represents a turn to the right (in the sense of the scan), at least one point is eliminated.

Algorithm:

1. Find p_0, the point with **minimum** y-coordinate.
2. **Sort** points about p_0 in a counterclockwise fashion (use polar coordinates).
3. Following CLR:
   a) PUSH(p_0, *stack*)
   b) PUSH(p_1, *stack*)
   c) PUSH(p_2, *stack*)
   d) for $i \leftarrow 3$ to n
      i) **do while** the angle formed by points Next-To-Top(*stack*), Top(*stack*), and p_i makes a *nonleft* turn:
         Pop(*stack*)
      ii) PUSH(*stack*, p_i)
4. **return** S

Proof of correctness: Discuss basics, but refer students to CLR.
Running Time:
1. Minimum takes $\Theta(n)$ time.
2. Sorting takes $\Theta(n \log n)$ time.
3. Each PUSH operations takes $\Theta(1)$ time.
4. The for-loop is executed n-3 times. Each PUSH operation in the for-loop takes $\Theta(1)$ time.
5. Each iteration of the while-loop takes $\Theta(1)$ time.
6. Using *amortized* analysis, it can be shown that the interior of the while loop is executed a total of $\Theta(n)$ times (total for the *entire* algorithm, not each time encountered by the for-loop). Notice that each point is pushed onto the stack exactly one time - step 3.d)ii). Notice that there is at most one POP operation for each point that was previously PUSHed onto the stack. Notice that during the while loop, the algorithm cannot back up over points that remain on the stack - each iteration of the while loop will remove exactly one item. Therefore, the total number of POP operations in the algorithm is $\Theta(n)$.

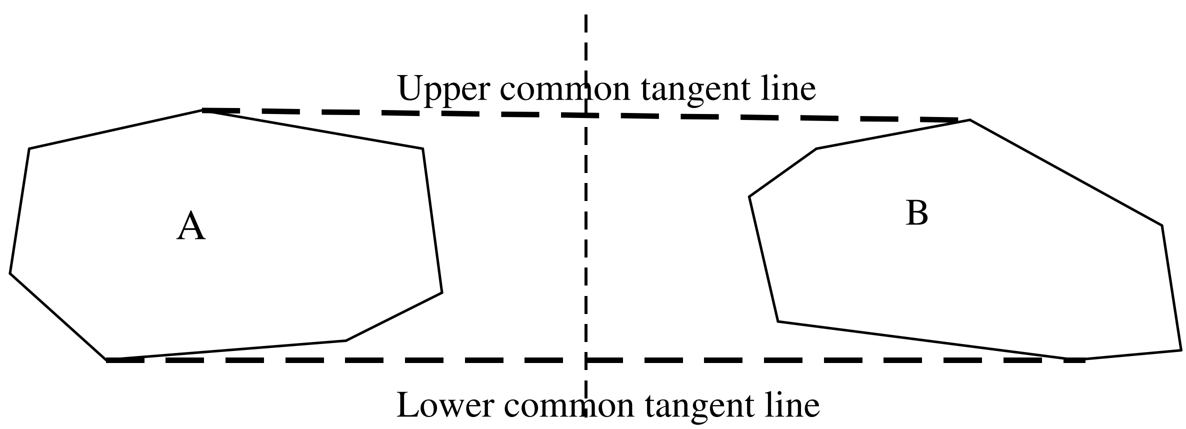
Hence, the total running time is dominated by the sort routine and is $\Theta(n \log n)$.

Mention **Jarvis's march**, which computes the convex hull by *package wrapping*:
- Running time is $O(nh)$, where h is the number of vertices in *hull*(S).
- The algorithm typically runs by searching in each iteration for the next point in the chain by comparing angles.
- Therefore, when the number of extreme points is $o(\log n)$, Jarvis's march is asymptotically faster than Graham's scan.

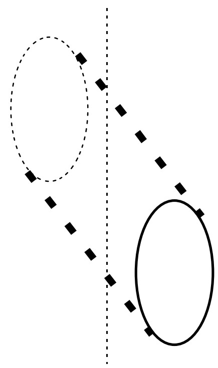
Divide-and-Conquer Solution

<u>Basic Solution:</u> Identify the extreme points of S, where $|S|=n$.

1. If $n \leq 2$, then **return** the points, as they are the extreme points of S. Otherwise:
2. Divide the n points by x-coordinates into 2 sets, A and B, each of size $n/2$, where all points in A are to the left of all points in B. (Note: dividing into 2 sets is arbitrary and machine-dependent.)
3. Recursively compute $hull(A)$ and $hull(B)$.
4. Stitch $hull(A)$ and $hull(B)$ together to determine $hull(S)=hull(hull(A) \cup hull(B))$.
    a) Find the upper and lower common tangent lines (often referred to as *lines of support*) between $hull(A)$ and $hull(B)$.
    b) Discard the points in the quadrilateral formed by the 4 points that represent the tangent lines.
    c) Renumber the extreme points (i.e., *enumerate* the extreme points so that they remain ordered for subsequent iterations).



Discuss tangent lines and possibilities of how they exist geometrically:



Discuss binary search algorithms to find both the upper and lower common tangent lines.

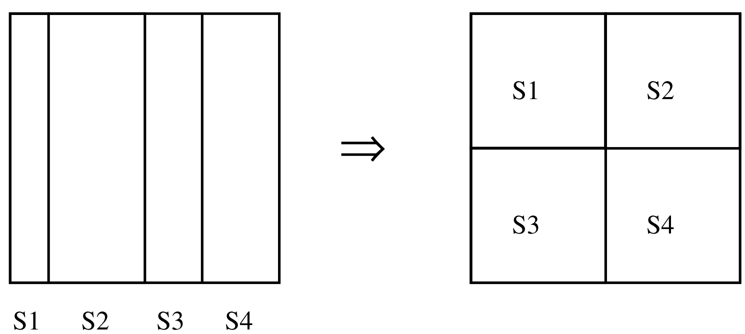
- The algorithm is based on lines collinear with hull edges and whether or not points of the other half are above a line collinear with an edge under consideration.
- Given that the points in each half are ordered, this algorithm is done via a binary search.

Running Times of Convex Hull D&C Algorithm

Sequential Algorithm:

- Sorting requires $\Theta(n \log n)$ time.
  - However, it is critical to note that the sort only needs to be performed *once* at the beginning of the algorithm as a *preprocessing* step and not every time through the recursion.
- The stitch step requires $\Theta(\log n)$ time to determine the lines of support (i.e., the common tangent lines) as well as $\Theta(n)$ time to reorder (compress) the data.
- So, there is $\Theta(n \log n)$ preprocessing time and $T(n) = 2T(n/2) + O(n) = O(n \log n)$ running time.
- Therefore, the running time for the algorithm is optimal at $\Theta(n \log n)$.

Mesh Algorithm: Given n points, arbitrarily distributed one point per processor on a mesh of size n^2, the convex hull of the set S of points can be determined in $\Theta(n^{1/2})$ time. First, sort the points into shuffled row-major order so that the following holds:

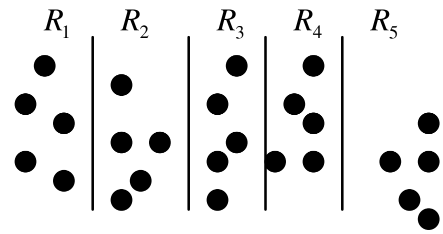


Binary Search:

1. Binary search from S1 to S2 and from S2 to S1, where after each iteration, the remaining data is compressed *together* so that the running time of the binary search is linear in the edgelength of the submesh that the points initially reside in.
2. Broadcast the 4 tangent points.
3. Eliminate points inside of quadrilateral and renumber.

- Running time of binary search is $B(n) = B(n/2) + \Theta(n^{1/2}) = \Theta(n^{1/2})$.
- So, the running time is $T(n) = T(n/4) + B(n) = T(n/4) + \Theta(n^{1/2}) = \Theta(n^{1/2})$, plus $\Theta(n^{1/2})$ for the initial sort, which is $\Theta(n^{1/2})$.

PRAM Algorithm (Atallah & Goodrich, 1986):



1. Sort the n points in S by x-coordinate so as to partition S into $n^{1/2}$ sets, denoted $R_1, R_2, \ldots, R_{n^{1/2}}$.
2. Recursively solve the convex hull problem for every $R_i, i \in \{1, \ldots, n^{1/2}\}$ in parallel. After this step, $hull(R_i)$ is known for every R_i.
3. Find $hull(S)$ from $\bigcup_{\forall i} hull(R_i)$. This is done by algorithm **combine**.

Combine:

Input: $hull(R_1), hull(R_2), \ldots, hull(R_{n^{1/2}})$, each with $O(n^{1/2})$ vertices, and ordered such that $hull(R_1) \le hull(R_2) \le \ldots \le hull(R_{n^{1/2}})$.

Output: $hull(S)$ from $\bigcup_{\forall i} hull(R_i)$

Method: Consider only the *upper hull of S* (the lower hull can be computed similarly).

1. Assign $n^{1/2}$ PEs to each R_i. For each R_i, find (and store) the tangent lines to each of the other $R_j, i \ne j$. Notice that a total of $n^{1/2} \times (n^{1/2} - 1) = O(n)$ such upper and lower tangent lines are to be determined by the entire system.
   a) Use a sequential binary search to determine a single tangent line by a single processor.
   b) Let $T_{i,j}$ be used to denote the (upper) common tangent line between R_i and $R_j, i \ne j$.
   c) Notice that all O(n) tangent lines are determined in $O(\log n)$ time.
2. Let V_i be the tangent line with smallest slope in $\{T_{i,1}, T_{i,2}, \ldots, T_{i,i-1}\}$. That is, V_i represents the tangent line of minimum slope that "comes from the left" in terms of R_i. Let v_i be the point of contact of V_i with $hull(R_i)$.
3. Let W_i be the tangent line with largest slope in $\{T_{i,i+1}, T_{i,i+2}, \ldots, T_{i,n^{1/2}}\}$. That is, W_i represents the tangent line of maximum slope that "comes from the right" in terms of R_i. Let w_i be the point of contact of V_i with $hull(R_i)$.

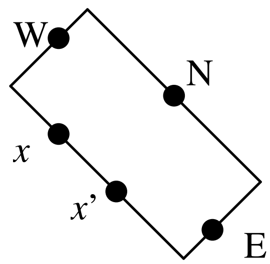
4. Notice that both V_i and W_i can be found in $O(\log n)$ time by the $n^{1/2}$ PEs assigned to R_i. This only requires a min or max operation, respectively, to be performed by the $n^{1/2}$ PEs.
5. Since neither V_i nor W_i can be vertical, they intersect and form an angle (with the interior point upward). If this angle is $< 180^o$, then none of the points of $hull(R_i)$ belong to $hull(S)$, while otherwise, all points from v_i to w_i, inclusive, belong to $hull(S)$. This requires $\Theta(1)$ time.
6. Compress all of the extreme points of $hull(S)$ into a compact region in memory in $O(\log n)$ time by performing parallel prefix computations.

Therefore, the running time of the algorithm is $\Theta(\log n)$.

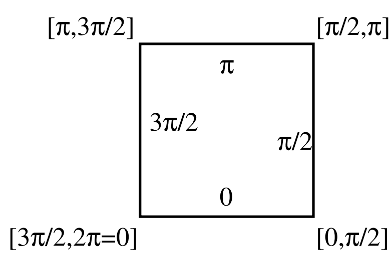
<See figures on pages 42-43 of M&S>

Discuss cost of the algorithm (optimal) and discuss the model required. Mention M&S result that solves for EREW PRAM, but don't do the algorithm.

Smallest Enclosing Box
(on a MCC of size *n*)



<u>Strategy:</u>
1. For *every* hull edge, determine W, N, and E.
2. Take the minimum over the areas of all such boxes.
3. Use a 1-step grouping operation based on "angles of support" so that these points can be computed simultaneously for all hull edges *xx'*.



<u>Algorithm:</u> {can be done within every *labeled* set of points}
1. Every *extreme point* in *hull(S)* creates 2 records, each corresponding to one of the extreme points angles of support.
2. Every hull *edge* creates 1 record corresponding to its angle of support.
3. Sort all records together by angles of support.
4. Perform a broadcast between ordered intervals, as determined by matching pairs of point records, to determine the points in question. Note that the point records are modified by adding $\pi/2$, π, or $3\pi/2$, depending on which point E, N, or W is being sought.

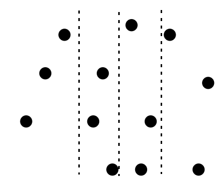
Running time is $\Theta(n^{1/2})$.

Closest Point Problem

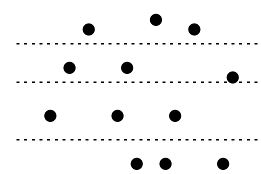
Defn.: Given a set of n planar points distributed in an arbitrary fashion one point per processor on a MCC of size n, find the nearest point to each point of the set.

Algorithm:

1. Vertical Regions
   a) Sort the points by x-coordinate so as to create 4 vertical slabs.
   b) Solve the problem recursively within each vertical slab.



2. Horizontal Regions
   a) Sort the points by y-coordinate so as to create 4 horizontal slabs.
   b) Solve the problem recursively within each horizontal slab.



3. Sort the points to group by boxes, as determined by the intersection of horizontal and vertical slabs.
   a) For the points in each box, there are only 2 points closest to each corner that could be closer a point in the box than any point found so far.
   b) For the $\leq 8 \bullet 16$ such points, pass them through the mesh so that all points can view them and determine if any are the desired points.

Running time: $T(n) = 2T(n/4) + \Theta(n^{1/2}) = \Theta(n^{1/2})$