Note: These lecture notes are based on lecture notes by Jeff Erickson and the textbook "Computational Geometry" by Berg et al.

1 Convex Hull

A set of points is said to be *convex* if it contains all points in the line segments connecting each pair of its points.

The *Convex hull* of a set of points P may be defined (equivalently) as:

1. The minimal convex set containing P
2. The intersection of all convex sets containing P
3. The union of all simplexes over points in P
4. The union of all convex combinations of points in P

A *simplex* is a polytope with the smallest number of vertices for a space of given dimensionality. In 2-D a simplex is a triangle; in 3-D, a tetrahedron.

A *convex combination* of a set of points $P' = \{p_1, \ldots p_\ell\}$ is any point $\sum_{i=1}^{\ell} \lambda_i p_i$, for any set of numbers λ_i, $i \in [1, n]$ such that $\forall_{i \in [1,n]} \lambda_i \geq 0$ and $\sum_{i=1}^{\ell} \lambda_i = 1$

1.1 Algorithmic Problem

2D Convex Hull Problem:
Given: A set of points P in the plane.
Output: Smallest, convex polygon containing all points in P

Definitions:

- *Polygon:* A region of the plane bounded by a cycle of line segments, joined end-to-end. The line segments are called *edges* and the points where there are joined are called *vertices*
- *Smallest:* Removing any point from the convex hull will violate convexity or the fact that it contains P.

This problem is equivalent to

- Finding the *largest* convex polygon whose vertices are points in P.
- Finding the set of all convex combinations of points in P (i.e. all points on any line segment between any pair of points).
- Finding the intersection of all convex regions containing P

Questions:

- Can there be a convex set of points (not necessarily a polygon) that contains P and has smaller area than the convex hull?
- What is the maximum number of vertices on the convex hull when $|P| = n$, and all points in P are in general position (see below)?

1.2 Applications

- *Recipes:* Each recipe for a crepe is a point in a 4-D space that gives the amount of each ingredient in the crepe (e.g. egg, milk, water and flour). Fundamentally, a crepe **is defined as** the convex-hull of all these points. Similarly, pancakes and flan are defined as separate convex hulls in the same space.
- *Robotics:* Find convex-hull of obstacles to simplify motion-planning problems
- *Chemical-Engineering:* Have several input mixtures of oil containing component A and B at a certain ratio. For example, have mixtures that are $(.3, .7)$, $(.5, .5)$, $(.9, .1)$. Can you achieve goal ratio (x, y) by mixing the input mixtures? Solution: Is (x, y) in the convex hull of the input points?
- *Biology:* Determining the territory that is "owned" by an animal or a pack.
- *Economics:* Utility functions are often convex - this reflects "diminishing marginal utility". Convex functions mean: "Less than Linear", i.e. [The function value is] Less than linear [interpolation between any sets of points on the function]. When functions are convex, it is often possible to efficiently find a global minimum. For a function whose negative is convex, we can often find a global maximum.
- *Computer Science:* Via duality and projections, we can use a solution to convex hull to also find Voronoi Diagrams, Delaunay Triangulations and Upper Envelopes.

1.3 General Position

Throughout this class we will assume points are in *general position*: no 3 points are on a line. This is analogous to assuming now two numbers are the same in sorting.

Technique to remove this assumption is *Symbolic Perturbation*. Intuitively, the idea is to perturb the coordinates of every point by a uniformly random amount chosen for a range small enough to have no impact on the output. Then, with all but negligible probability, the points will be in general position.

2 2D Convex Hull Algorithms

2.1 Counter-clockwise Ordering

A key primitive for our 2D convex hull algorithms will be a tool for measuring "how counter-clockwise" a pair of points is with respect to some fixed point. This primitive is analogous to the comparison operation of two numbers in sorting algorithms.

Consider 3 points, x, y, z. Let $x = (a, b), y = (c, d)$ and $z = (e, f)$. Then the three points are in *Counterclockwise (ccw) order* if the cross-product of the two vectors $(c, d) - (a, b)$ and $(e, f) - (a, b)$ is greater than 0. This holds iff

$$(f - b)(c - a) - (d - b)(e - a) > 0$$

One can also think of the slope of the line given by (a, b), (c, d) being less than the slope of the line given by (a, b), (e, f). In particular, if (a, b) is to the left of (c, d) and (e, f), then ccw order holds only if

$$\frac{d - b}{c - a} < \frac{f - b}{e - a}$$

Cross-multiplying, we get that the points are in ccw order iff

$$(f-b)(c-a) > (d-b)(e-a)$$

This final equivalence is true even if $x = (a, b)$ is not the leftmost point.

Next, we say that a point q in a set of points P is the *Closest Clockwise Point* from a given point p if the following holds. The slope of the line supported by p and q is less than the slope of the line supported by p and q' for any other $q' \in P$.

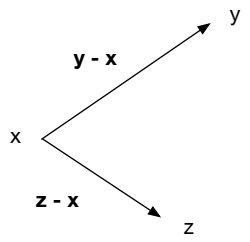


Figure 1. From the viewpoint of x, y is ccw of z. The case where x is on the other side of the line supported by y and z is also handled by using the cross product.

2.2 Jarvis' Algorithm (Wrapping)

Let's start with something simple. Jarvis' algorithm works by first putting the point with smallest x-coordinate into the hull. Then, it repeatedly finds the next closest ccw point from the last point added to the hull, and adds that point to the hull.

procedure JARVIS(P)
  $\ell \leftarrow$ leftmost point in P
  $p \leftarrow \ell$
  repeat
    $q \leftarrow$ closest clockwise point from p, among all points not yet in hull
    $next(p) \leftarrow q$ ▷ next(p) is next point in convex hull
    $p \leftarrow q$
  until $p = \ell$
end procedure

The step where we find q takes $O(n)$ time (to find a minimum). Thus the entire algorithm takes $O(hn)$ time where h is the number of points on the convex hull.

2.3 Divide and Conquer

Can we design a more efficient algorithm? Let's try Divide and Conquer. In this algorithm, we: (1) partition the points using a random "pivot" point; (2) recursively compute the hull of each partition, and (3) then patch up the two hulls to get the hull of all the points.

procedure DIVIDEANDCONQUER(P)
  $p \leftarrow$ point chosen uniformly at random in P
  $L \leftarrow$ all points with x-coordinate less than p; $R \leftarrow P - L$; Then add p in the smaller of L and R

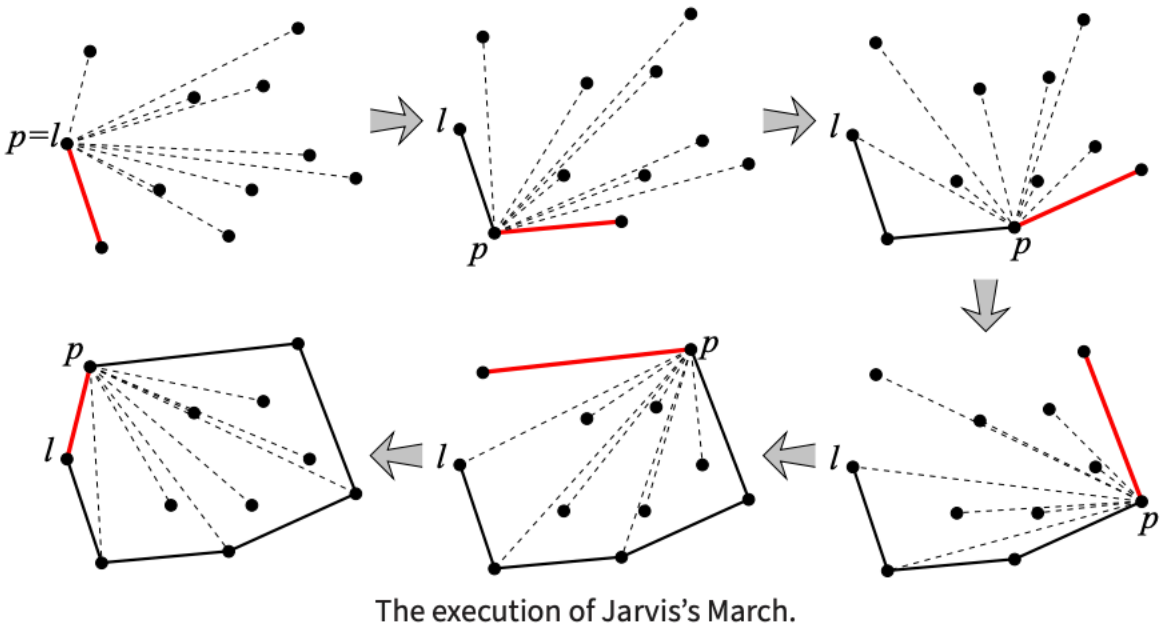


Figure 2. Jarvis' March From Jeff Erickson's lecture notes.

$C_L \leftarrow$ convex hull of L (computed recursively)
$C_R \leftarrow$ convex hull of R (computed recursively)
"Merge" C_L and C_R to get convex hull of P
Create two bridges between the rightmost point in C_L and leftmost point in C_R
while Either endpoint of either bridge is in a concave corner **do**
    Remove the vertex at the middle of this corner from the hull
    Update that bridge to connect the endpoints of that concave corner
end while
end procedure

For an ordered sequence of points, p, q, r in a potential convex hull, we say that p, q and r form a *concave corner* if from p's viewpoint, r is ccw of q. Then q is the corner vertex that is removed from the hull.

The figure shows the merging. The red line segments are the bridges. The yellow wedges represent concave corners found. Note that the middle point of the wedge is the point that is always removed.

The runtime is a random variable. We can use linearity of expectation to create a recurrence relation for the expected run time, where $T(n)$ is the expected runtime of the algorithm when there are n points. See Section A for the proof that the expected runtime is $O(n \log n)$.

2.4 GrahamScan

Can we solve convex hull deterministically in $O(n \log n)$? The following algorithm shows that this is possible. The proof of correctness is a bit subtle.

procedure GrahamScan(P)
    $\ell \leftarrow$ leftmost point in P

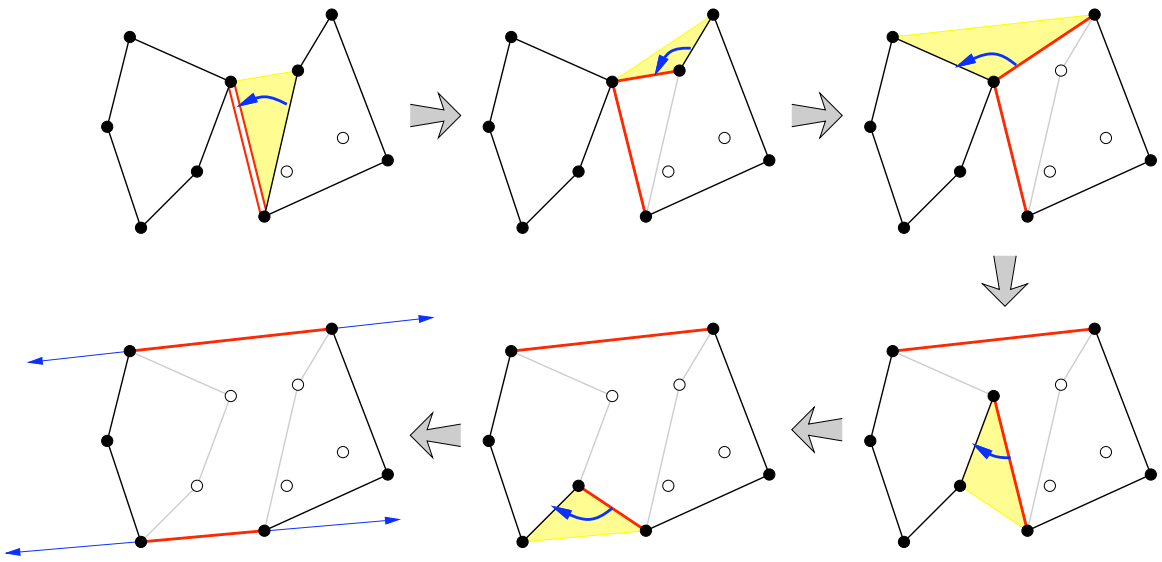
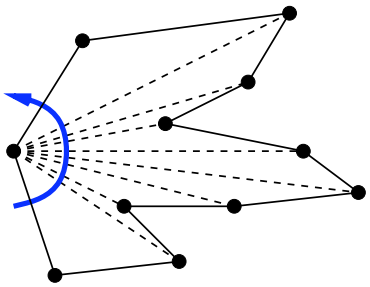


Figure 3. Merging Hull - From Jeff Erickson's lecture notes.



A simple polygon formed in the sorting phase of Graham's scan.

Figure 4. Sorting counter-clockwise wrt to leftmost point ℓ

Sort all points in P in CCW order with respect to ℓ
Let p, q and r be consecutive vertices in the sorted P. (Imagine there are 3 pennies on these vertices.)
repeat
if p, q and r are in ccw order **then**
Output vertex back penny is on as part of convex hull
Move back penny forward to the successor of r (in P)
else
Remove q from P, move middle penny *back* to predecessor(p)
end if
until r equals ℓ
end procedure

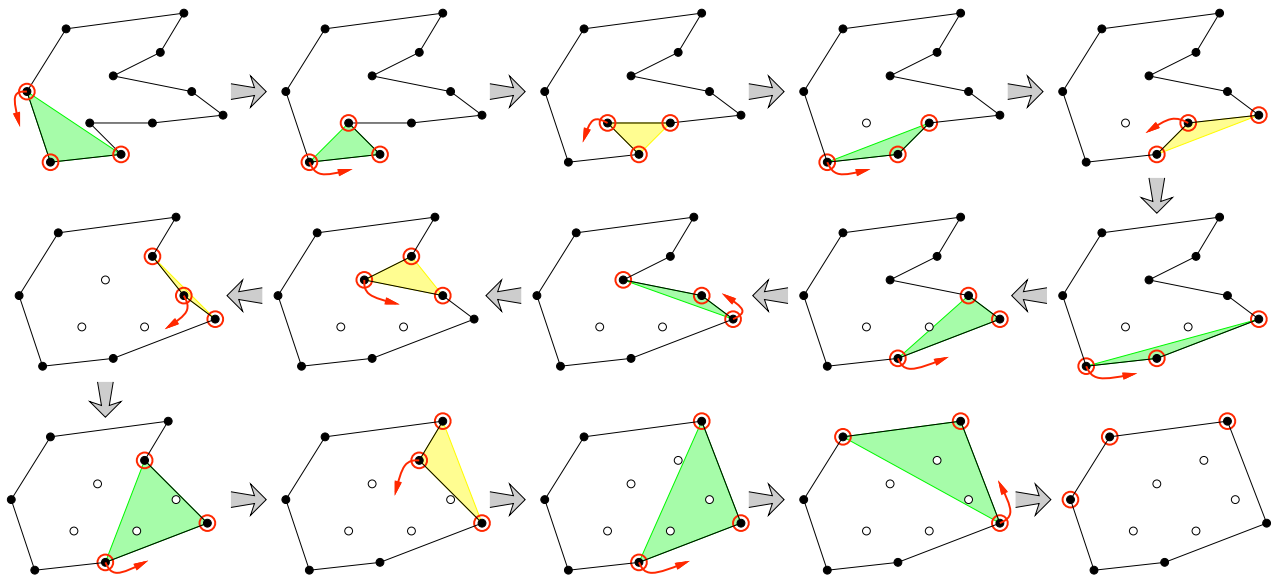


Figure 5. Steps of Graham Scan

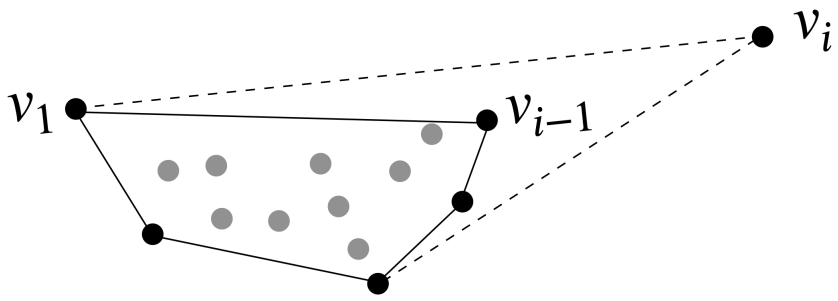


Figure 6. Inductive Step for Graham Scan

2.4.1 Runtime

Whenever a penny moves forward, it is to a vertex never seen before, so the first part of the if statement happens $O(n)$ times.

Whenever a penny moves backward, a vertex is removed from P, so this step happens $O(n)$ times.

Thus, total runtime of algorithm is dominated by the sorting which takes $O(n \log n)$ time.

2.4.2 Correctness

The following is based on this proof from the CMU CS15-456 Computational Geometry Class. (However, I have made significant edits, any mistakes introduced are my own).

Theorem 1. *GrahamScan outputs the convex hull of any set of at least* 3 *input points.*

To show this, we first prove a key lemma. Let $v_1, v_2, \ldots v_n$ be the points of the input sorted in CCW order. For any $1 \leq i \leq n$, let $V_i = v_1, v_2, \ldots v_j$; and let $CH(V_i)$ be the convex hull of these i points. Define $OS(V_i)$ (for output stack) to be the vertices output by the algorithm as being on

the hull, **plus** the 3 vertices covered by pennies **when the condition of if statement is met (i.e. we have a green triangle in Figure 5)** after processing points V_i.

Two key points. First, note that $OS(V_i)$ is well defined for all $i \in [1, n]$ since, in the case where the front penny is on v_i and the triangle is not green, the middle penny keeps moving until we get a green triangle with the front penny on v_i. Second, remember that the last 3 vertices in $OS(V_i)$ need not be vertices that are eventually output as being in the convex hull. Instead, they are the vertices that the 3 pennies are on.

Lemma 1. *For any set of $n \geq 3$ vertices as defined above as input to Graham-Scan, we have*

$$OS(V_n) = CH(V_n)$$

Proof: We prove by induction on $i \in [3, n]$ that $OS(V_i) = CH(V_i)$

BC: (i=3) $OS(V_3)$ is the set of all 3 vertices in V_3 in CCW order, and this is $CH(V_3)$.

IH: $OS(V_{i-1}) = CH(V_{i-1})$.

IS: Consider $OS(V_i)$. By the IH, $OS(V_{i-1})$ is in CCW order, and by the properties of the algorithm, when v_i is placed on the output stack, $OS(V_i)$ is in CCW order (See Figure 6).

We also know that $OS(V_i)$ starts at vertex v_1 and ends at vertex v_i. Our goal is to show

1. All points in V_i are inside $OS(V_i)$
2. All points in V_i that are not on $CH(V_i)$ are not in $OS(V_i)$.

We first show (1). By the IH, all points in V_{i-1} are inside $OS(V_{i-1})$. When processing the point v_i, we may have removed some points in $OS(V_{i-1})$. But, by the properties of the algorithm, these are exactly the points that are in the wedge induced by the line from v_1 to v_i and the line from v_i to some previous point on $OS(V_{i-1})$ (See Figure 6). Thus all points in V_i are within $OS(V_i)$.

We next show (2). By the IH, $OS(V_{i-1})$ does not contain any points not on the hull $CH(V_{i-1})$. Since the algorithm never revisits a point that is discarded, these points will also not be on $OS(V_i)$. But, there may be points on $OS(V_{i-1})$ that are not on $CH(V_i)$. However, these are exactly the points that are strictly inside the wedge induced by v_1, v_i and the closest clockwise point to v_i on $CH(V_{i-1})$ (again See Figure 6). The algorithm pops off all of these points from the output stack while processing point v_i. Thus v_i is correctly added to $CH(V_{i-1})$. This creates a new convex polygon that includes v_i as a vertex. □

With this lemma in hand, we can prove Theorem 1. First, note that $OS(V_n)$ is the set of points output by Graham-Scan as being in the convex hull, since when the front penny is on v_n and there is a green triangle on all the pennies, these last three pennies will all be output. Second, note that by Lemma 1, $OS(V_n) = CH(V_n)$.

2.5 Chan's algorithm

Chan's algorithm is output sensitive in that the runtime is $O(n \log h)$.

To understand it better, assume first that we know h in advance. Then we do the following

procedure CHAN(P)
Partition P arbitrarily into n/h sets of size h
Compute convex hulls of each partition (using say Graham scan)

Compute convex hull of all these hulls via "wrapping" (as in Jarvis' march)
end procedure

Total runtime is $O(n/h(h \log h)) = O(n \log h)$ for the recursive calls. When we "wrap", we repeatedly need to find the tangent line between a vertex p and any sub-hull. This can be done in $O(\log h)$ time via a type of binary search. Since there are n/h subhulls, takes $O((n/h) \log h)$ time to find the successor of p. Since we find the successor $h-1$ times, wrapping takes $O(n \log h)$ time.

So everything works great if we know h in advance. What if we don't?

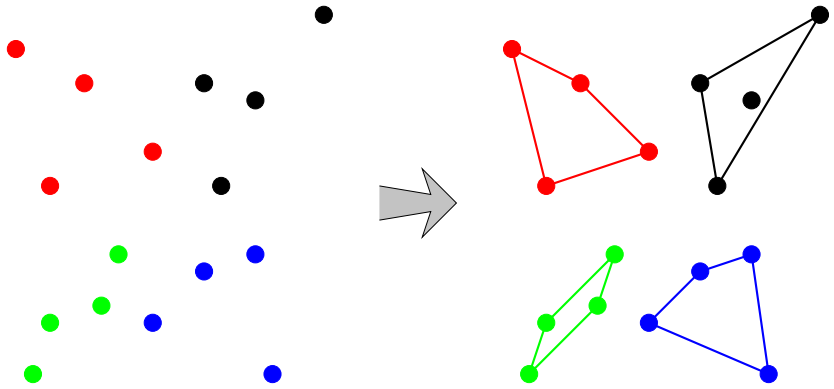


Figure 7. Chan's algorithm: Compute hulls of partitions

2.5.1 Guessing h

In Chan's algorithm, we actually guess increasingly large values of h. Let $h' = 3$ be our first guess. If our guess is too small, we square h' and try again. It's not hard to bound the total cost of all calls to Jarvis' March so that is left as an exercise. We now focus on the cost due to the calls to Graham Scan.

In the final iteration, $h' \leq h^2$, so the total cost of the last iteration is $O(n \log h^2) = O(n \log h)$. Say that there are k total iterations, the cost of all iterations is

$$\sum_{i=1}^{k} O(n \log 3^{2^i}) = O\left(n \sum_{i=1}^{k} 2^i\right).$$

Since a geometric summation is always a constant times its largest term, total runtime is big-O of the time of the last iteration which is $O(n \log h)$.

3 Open Problems

Convex-hull in the CONGEST model. Each point is a wireless node. Can compute in parallel, but in each round, each node can only broadcast $O(polylog(n))$ bits. What is the minimum number of rounds needed to compute the convex-hull? In particular, at the end, we want each node to know (1) if it is in the convex hull; and (2) if so, what are its two neighbors in the hull.

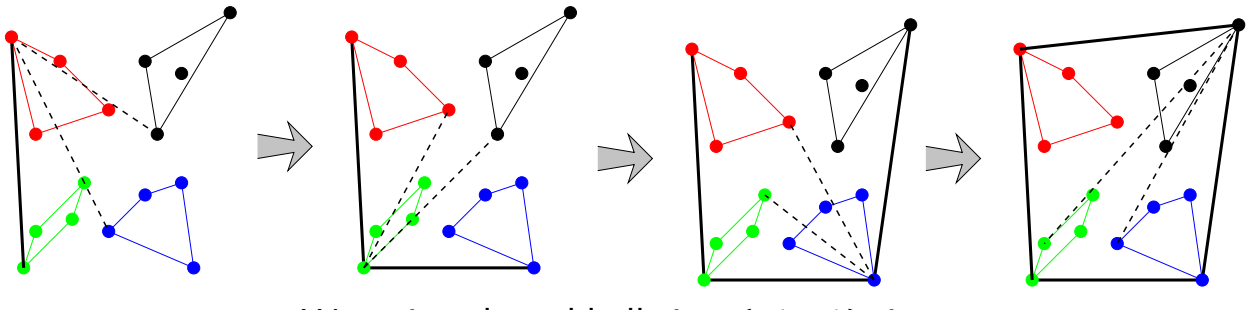


Figure 8. Chan's algorithm: "wrap" the subhulls

A Recurrence for Divide and Conquer Algorithm

We are putting the pivot element in smallest of the two partitions. The amount of time to merge the hull by is n.[1] Let $T(n)$ be the expected runtime of the algorithm on an input of size n. For the recurrence, we can imagine that i ranges over the pivot choice being the i-th smallest value on the x-coordinate. Assume n is even[2] Then, we can first write out the recurrence as the two sums of size $n/2$. To simplify the expression, note that we can pull the time for the merging of the convex hull outside the sum since it is always n

$$\begin{aligned} T(n) &= n + \frac{1}{n}\left[(T(1) + T(n-1) + \ldots + T(n/2) + T(n/2) + T(n/2) + T(n/2) + \ldots T(1) + T(n-1)\right] \\ &= n + \frac{2}{n}\sum_{i=1}^{n/2}\left[T(i) + T(n-i)\right] \\ &= n + \frac{2}{n}\left[T(n/2) + \sum_{i=1}^{n-1} T(i)\right] \end{aligned}$$

Let's now show that this recurrence is $O(n \log n)$.

Lemma 2. *$T(n) \leq Cn \log n$, for all $n \geq n_0$, for two positive constants C and n_0.*

Proof: BC: $n = 2$
This is immediate.

Inductive Step:

For arbitrary n, we have:

[1] To be precise it is less than some constant times n but we can normalize a time step based on the size of that constant to simplify notation.

[2] We can remove this assumption, by for example adding an additional point to P when n is odd.

$$
\begin{aligned}
T(n) &= n + \frac{2}{n}\left[T(n/2) + \sum_{i=1}^{n-1} T(i)\right] \\
&\leq n + \frac{2}{n}\left[C(n/2)\log(n/2) + \sum_{i=1}^{n-1} Ci\log i\right] \\
&\leq n + C\log n + \frac{2C}{n}\left[\sum_{i=1}^{n-1} i\log i\right] \\
&\leq n + C\log n + \frac{2C}{n}\left[\sum_{i=1}^{n/2} i\log(n/2) + \sum_{i=(n/2)+1}^{n-1} i\log n\right] \\
&\leq n + C\log n + \frac{2C}{n}\left[\sum_{i=1}^{n-1} i\log n - \sum_{i=1}^{n/2} i\right] \\
&\leq n + C\log n + C(n-1)\log n - \frac{C(n/2)^2}{n} \\
&= n + Cn\log n - Cn/4 \\
&\leq Cn\log n
\end{aligned}
$$

□

The second line holds by the IH. The last line holds for $C \geq 4$.