On a simple, practical, optimal, output-sensitive randomized planar convex hull algorithm

Binay K. Bhattacharya*   Sandeep Sen†

Abstract

In this paper we present a truly practical and provably optimal $O(n \log h)$ time output-sensitive algorithm for the planar convex hull problem. The basic algorithm is similar to the algorithm presented in Chan, Snoeyink and Yap[2] where the median-finding step is replaced by an approximate median. We analyze two such schemes and show that for both methods, the algorithm runs in expected $O(n \log h)$ time. The expected number of comparisons can be made smaller than $5n \log h$ for the upper-hull. We further show that the probability of deviation from expected running time approaches 0 rapidly with increasing values of n and h for any input.

Our experiments suggest that this algorithm is a practical alternative to the worst-case $O(n \log n)$ algorithms like Graham's and especially faster for small output-sizes. Our approach bears some resemblance to a recent algorithm of Wenger[13] but our analysis is substantially different.

1 Introduction

The planar convex hull problem is perhaps the most studied problem in computational geometry and a large body of literature deals with computing convex hulls. Graham [5] was the first to present an $O(n \log n)$ worst-case time algorithm. This algorithm is optimal as Yao [14] showed that $\Omega(n \log n)$ is the lower bound of the convex hull problem for the worst-case input. Some simple algorithms have $O(n)$ expected time for known distributions of points such as uniform in a box, normal, etc. The first output-sensitive algorithm was proposed by Chand and Kapur [3]. The two-dimensional version of their algorithm is known as the rope fence method and was independently reported by Jarvis [6]. The rope fence method takes $O(nh)$ time to compute h extreme edges of the convex hull. Kirkpatrick and Seidel [8] proved an $\Omega(n \log h)$ lower bound when both input and output sizes are considered, so Yao's lower-bound is a special case when $\log h \in \Omega(\log n)$. They also proposed an $O(n \log h)$ optimal algorithm based on the prune-and-search technique developed by Dyer [4] and Megiddo [9]. However, it has high constants and is considered prohibitively complicated for implementation. Very recently, in [1], two $O(n \log h)$ algorithms have been proposed. One uses the linear-time median finding algorithm and the other uses a clever grouping technique. Although the latter algorithm does not have any expensive median-finding step it relies on a sophisticated logarithmic time tangent-finding routine.

*School of Computing Science,Simon Fraser University, Burnaby, Canada.

†Department of Computer Science and Engineering, Indian Institute of Technology, New Delhi, India.

Attempts have also been made to design simple and efficient randomized algorithms. Seidel [11] proposed a simple $O(n \log n)$ expected running time randomized incremental algorithm. A number of attempts were made to design a planar convex-hulls along the lines of quicksort. These are known as *quickhulls* (see [10]) and although the experimental evidence has been encouraging, the theoretical analysis could only demonstrate $O(n^2)$ running time. Recently, a significant improvement was achieved by Wenger [13] who proposed an $O(n \log h)$ output-sensitive quickhull-like randomized algorithm. The algorithm is claimed to be very efficient even though the hidden constant in the analysis is very high and the analysis itself involves undue complications. Wenger did not provide any tail estimates of the running time and as such appears hard to analyze by using his approach. Shafer and Steiger [12] have also claimed an $O(n \log h)$ expected time quickhull based randomized algorithm - however their proof seems to hold only for $h \in \Omega(n)$.

In this paper we present a truly simple randomized algorithm which computes the convex hull in $O(n \log h)$ expected time. It can be viewed as a further simplified version of one of the algorithm proposed by Chan et al.[2] We have used two different strategies to replace the median finding step in [1]. In one strategy, the median element of a given set is replaced by a random element of the set. In the other strategy the median element is replaced by the exact median of some randomly selected elements. [1] Note that this is not the same as replacing the median-finding step by a randomized median-finding algorithm.

We provide tail estimates of our randomized algorithm for both the strategies. It is shown that the algorithm terminates in $O(n \log h)$ time with probability $1 - O(h^{-1})$ using the first strategy and with probability $1 - O(2^{-\Omega(n^{1/4})})$ using the other. Unlike Wenger's [13] analysis, the constant associated with the expected running time is quite small. We feel that this algorithm is a genuine adaptation of the quicksort algorithm to the planar hull problem - we will let the reader be a judge of that. The proposed algorithm also bears some resemblance to Wenger's randomized quickhull [13] and some of our observations can be used to simplify his analysis.

We present our algorithm in next section along with its analysis. We have implemented the algorithm and some experimental results are discussed in section 3.

2 The Algorithm

Let S be a set of n points whose convex hull has to be constructed. We compute the convex hull of S by constructing the upper and the lower hull of S. Let p_l and p_r be the extreme points of S in the x direction. Let S_a (S_b) be the subset of S which lie above (below) of the line through p_l and p_r. Clearly, $S_a \cup \{p_l, p_r\}$ and $S_b \cup \{p_l, p_r\}$ determine the upper and the lower convex hulls. We first describe the basic algorithm *Basic-Randomized-Upper-Hull.* to determine the upper hull using $S_a \cup \{p_l, p_r\}$. Subsequently, we refine it further that will be arguably better in practice.

In the following description, we denote the line segment joining p and q by $\overline{pq}$. The slope of the line joining p and q is denoted by $slope(pq)$. The predicate *left-turn*(x, y, z) is true if the sequence x, y, z has a counter-clockwise orientation, or equivalently the area of the triangle

[1] In [2] it was suggested as a possible alternative without any analysis. However, it was also discovered independently by the authors.

has a positive sign.

Algorithm	**Basic-Randomized-Upper-Hull**(S_a, p_l, p_r)
Input:	Given $S_a = \{p_1, p_2, \ldots, p_n\}$ and the leftmost extreme point p_l and the rightmost extreme point p_r. All points of S_a lie above the line $\overline{p_l p_r}$.
Output:	Extreme points of the upper hull of $S_a \cup \{p_l, p_r\}$ in clockwise order.
Step 1.	If $S_a = \{p\}$, then return the extreme point $\{p\}$.
Step 2.	Select randomly a pair $\{p_{2i-1}, p_{2i}\}$ from the the pairs $\{p_{2j-1}, p_{2j}\}$, $j = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$.
Step 3.	Select the point p_m of S_a which supports a line with slope$(p_{2i-1}p_{2i})$. (If there are two or more points on this line then choose a p_m that is distinct from p_l and p_r). Assign $S_a(l) = S_a(r) = \emptyset$).

Step 4.	For each pair $\{p_{2j-1}, p_{2j}\}$, $j = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$ do the following
	(assuming $x[p_{2j-1}] < x[p_{2j}]$)
	Case 1: $x[p_{2j}] < x[p_m]$
	    if *left-turn* (p_m, p_{2j}, p_{2j-1}) then $S_a(l) = S_a(l) \cup \{p_{2j-1}, p_{2j}\}$
	    else $S_a(l) = S_a(l) \cup \{p_{2j-1}\}$.
	Case 2: $x[p_m] < x[p_{2j-1}]$
	    if *left-turn* (p_m, p_{2j-1}, p_{2j}) then $S_a(r) = S_a(r) \cup \{p_{2j}\}$
	    else $S_a(r) = S_b(r) \cup \{p_{2j-1}, p_{2j}\}$.
	Case 3: $x[p_{2j-1}] < x[p_m] < x[p_{2j}]$
	    $S_a(l) = S_a(l) \cup \{p_{2j-1}\}$;
	    $S_a(r) = S_a(r) \cup \{p_{2j}\}$.

Step 5.	(i) Eliminate points from $S_a(l)$ which lie below the line joining p_l and p_m. (ii) Eliminate points from $S_a(r)$ which lie below the line joining p_m and p_r.
Step 6.	If $S_a(l) \neq \emptyset$ then *Basic-Randomized-Upper-Hull*$(S_a(l), p_l, p_m)$. Output p_m. If $S_a(r) \neq \emptyset$ then *Basic-Randomized-Upper-Hull*$(S_a(r), p_m, p_r)$.

Chan et al.[2] showed that when the slope of the line containing the selected pair at step 2 is the median of the $slope(p_{2j-1}p_{2j})$, $j = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$, the algorithm has $O(n \log h)$ worst case running time, where h is the size of the extreme points on the upper convex hull.

2.1 Implementation

We have used two simple strategies to implement step 2. These strategies are:

- Strategy 1: Select a random pair uniformly from the pairs $\{p_{2j-1}, p_{2j}\}$, $j = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$. The slope of the line containing the selected pair is taken to be the approximation of the median slope of the $slope(p_{2j-1}p_{2j})$, $j = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$.

- Strategy 2: Select the exact median slope of a set R of about $\sqrt{\frac{n}{2}}$ slopes selected randomly from $slope(p_{2j-1}p_{2j})$, $j = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$. We will implement this step by choosing each pair to be in R with probability $1/\sqrt{\frac{n}{2}}$.

Strategy 1 requires one random number to be generated where as $\sqrt{\frac{n}{2}}$ random numbers are needed for Strategy 2. Once the random slopes are selected for strategy 2, the exact median slope of the selected slopes can be obtained in $O(\sqrt{n} \log n)$ time by sorting the slopes first.

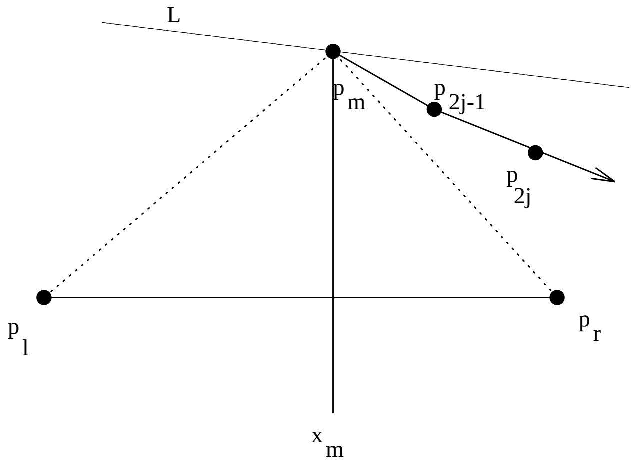

Figure 1: *Left-turn*(p_m, p_{2j-1}, p_{2j}) is true but *slope*$(\overline{p_{2j-1}p_{2j}})$ is less than the median slope given by L.

In step 4, procedure *left-turn*(a, b, c) determines whether c lies to the left of the directed line joining a to b. Here the *left-turn* test is more general [2] than the test based on only slopes use in Chan et al.[2]. When the pair $\{p_{2j-1}, p_{2j}\}$ is being considered, Chan [1] tested whether the slope$(p_{2j-1}p_{2j})$ is greater than the median slope when $x(p_{2j-1}) > x(p_m)$. However, *left-turn* (p_m, p_{2j-1}, p_{2j}), when $x(p_{2j-1}) > x(p_m)$, is more general in the sense that it subsumes the case considered in Chan et al.[2] (see Fig. 1). The *left-turn* test in case 2 (i.e. when $x(p_{2j}) < x(p_m)$) is similarly more general.

To enhance the performance of our algorithm in practice, we will now include certain optimizing heuristics that have been used in quickhull-based algorithms. In step 3, if the pair $\{p_{2i-1}, p_{2i}\}$ realizing a slope satisfies the property that the line containing $\overline{p_{2i-1}p_{2i}}$ must not intersect the line segment $\overline{p_l p_r}$, then it guarantees that p_{2i-1} or p_{2i} does not lie inside the triangle $\triangle p_l p_{2i} p_r$ or $\triangle p_l p_{2i-1} p_r$ respectively. Wenger [13] used this feature during the selection of the random slope. As a result of this the extreme point p_m computed in step 4 will be distinct from p_l and p_r. Notice here that when the line containing $\overline{p_{2i-1}p_{2i}}$ intersects $\overline{p_l p_r}$, either p_{2i-1} or p_{2i} can not be an extreme point and can, therefore, be eliminated from the set.

The final algorithm is formally described below.

[2] This was suggested by Mike Houle

Algorithm	**Randomized-Upper-Hull**(S_a, p_l, p_r)
Input:	Given $S_a = \{p_1, p_2, \ldots, p_n\}$ and the leftmost extreme point p_l and the rightmost extreme point p_r. All points of S_a lie above the line $\overline{p_l p_r}$.
Output:	Extreme points of the upper hull of $S_a \cup \{p_l, p_r\}$ in clockwise order.
Step 1.	If $S_a = \{p\}$, then return the extreme point $\{p\}$.
Step 2.	Select randomly a pair $\{p_{2i-1}, p_{2i}\}$ from the the pairs $\{p_{2j-1}, p_{2j}\}$, $j = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$.
Step 3:	(i) Delete p_{2i-1} from S_a if p_{2i-1} lies inside the triangle $\triangle p_l p_{2i} p_r$; set $n \leftarrow n-1$; Goto 2. (ii) Delete p_{2i} from S_a if p_{2i} lies inside the triangle $\triangle p_l p_{2i-1} p_r$; set $n \leftarrow n-1$; Goto 2.
Step 4.	{ Line joining p_{2i-1} and p_{2i} does not intersect $\overline{p_l p_r}$. } Select the point p_m of S_a which supports a line with slope$(p_{2i-1}p_{2i})$. Assign $S_a(l) = S_a(r) = \emptyset$.

Step 5.	For each pair $\{p_{2j-1}, p_{2j}\}$, $j = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$ do the following (assuming $x[p_{2j-1}] < x[p_{2j}]$)
	Case 1: $x[p_{2j}] < x[p_m]$
	    if *left-turn* (p_m, p_{2j}, p_{2j-1}) then $S_a(l) = S_a(l) \cup \{p_{2j-1}, p_{2j}\}$
	    else $S_a(l) = S_a(l) \cup \{p_{2j-1}\}$.
	Case 2: $x[p_m] < x[p_{2j-1}]$
	    if *left-turn* (p_m, p_{2j-1}, p_{2j}) then $S_a(r) = S_a(r) \cup \{p_{2j}\}$
	    else $S_a(r) = S_b(r) \cup \{p_{2j-1}, p_{2j}\}$.
	Case 3: $x[p_{2j-1}] < x[p_m] < x[p_{2j}]$
	    $S_a(l) = S_a(l) \cup \{p_{2j-1}\}$;
	    $S_a(r) = S_a(r) \cup \{p_{2j}\}$.

Step 6.	(i) Eliminate points from $S_a(l)$ which lie below the line joining p_l and p_m. (ii) Eliminate points from $S_a(r)$ which lie below the line joining p_m and p_r.
Step 7.	If $S_a(l) \neq \emptyset$ then *Randomized-Upper-Hull*$(S_a(l), p_l, p_m)$. Output p_m. If $S_a(r) \neq \emptyset$ then *Randomized-Upper-Hull*$(S_a(r), p_m, p_r)$.

2.2 Analysis

We will first analyze only the basic algorithm - the second version will behave at least as well as the basic algorithm. Subsequently, our proof techniques for the tail estimates will exploit an important consequence of the additional refinement of the second algorithm. Also, we suspect that the second algorithm will be better in practice for some classes of inputs because some non-hull points may be eliminated *cheaply* in Step 3.

Note that the only additional space (besides the stack used for recursive calls) used by our algorithm, is for implementing **Step 2** to find an approximate median. This is $O(1)$ for strategy 1 and $O(\sqrt{n})$ for strategy 2. The more interesting aspect of the analysis is bounding the time-complexity.

For the purpose of analysis we will view the algorithm as a tree $\mathcal{T}$ where the root node corresponds to input set S_a of size n and each internal node v of the tree represents a recursive call. If a sub-problem is empty, we attach a dummy leaf-node corresponding to it. (If both subproblems are empty then the node itself is a leaf node.) So each node has two children corresponding to the left and right subproblems. We will denote this tree by $\mathcal{T}$. For each node $v \in T$, *size(v)* denotes the size of the subproblem associated with the node v and *height (v)* denotes the height of v in $\mathcal{T}$.

In the tree $\mathcal{T}$ corresponding to the modified algorithm, we note that each node a distinct extreme point, namely p_m is discovered. So $\mathcal{T}$ has $O(h)$ nodes for the modified algorithm and we will exploit this observation when we analyze the tail estimates for running time.

Estimating the number of nodes in the Basic algorithm is not as easy since one of the end-points can be repeatedly chosen as p_m. However, if p_m is p_l, then at least one point is eliminated from the pairs whose slopes are larger than the supporting line L through p_l. If L has the largest slope, then there are no other points on the line supporting p_m (Step 3 of Basic algorithm). Then for the pair (p_{2j-1}, p_{2j}), whose slope equals that of L, *left-turn* (p_m, p_{2j}, p_{2j-1}) is true. See Figure 1, so p_{2j-1} will be eliminated. Hence it follows that the number of nodes in the tree corresponding to the Basic algorithm is $O(n+h)$.

Lemma 1 The number of recursive calls in the Basic Randomized-Upper-Hull algorithm can be bounded by $O(n)$ and in the modified algorithm by $O(h)$.

Let N represent the set of $slopes(p_{2j-1}p_{2j}), j = 1, 2, \ldots \lfloor \frac{n}{2} \rfloor$. Let k be the rank of the $slope(p_{2i-1}p_{2i})$, selected uniformly at random from N. Let n_l and n_r be the sizes of the subproblems determined by the extreme point supporting the line with $slope(p_{2i-1}, p_{2i})$. Chan et al.[2] showed that $max(n_l, n_r) \leq 3n/4$ for $k = n/4$. Using similar arguments we can show that

Lemma 2: $max(n_l, n_r) \leq n - min(\lfloor \frac{n}{2} \rfloor - k, k)$.

Proof Without loss of generality, let us bound the size of the right sub-problem. There are $\lfloor \frac{n}{2} \rfloor - k$ pairs with slopes greater than or equal to $slope(p_{2i-1}p_{2i})$. At most one point out of every such pair can be an output point to the right of p_m. Note that even if some pairs (say s) are eliminated in Step 3 of the modified algorithm, the bound remains $\lfloor \frac{n}{2} \rfloor - k - s + s = \lfloor \frac{n}{2} \rfloor - k$ $\Box$.

When Strategy 1 is employed, it is easy to see that k lies in $[\frac{n}{8}, \frac{3n}{8}]$ with probability $\frac{1}{2}$. We will now show that that k lies in $[\frac{n}{8}, \frac{3n}{8}]$ with probability exceeding $1 - 2^{-\Omega(\sqrt{n})}$ when strategy 2 is employed. This is equivalent to showing that k lies in $[1, \frac{n}{8}]$ with probability less than $2^{-\Omega(\sqrt{n})}$. We make use of probabilistic inequalities also known as Chernoff bounds (see appendix). Let N be the set of $\lfloor \frac{n}{2} \rfloor$ slopes. Let R be the random subset of N where each element is chosen with probability $1/\sqrt{\frac{n}{2}}$. Suppose the median slope, say z, of R has rank less than $\frac{n}{8}$ in N. This means that more that $\frac{1}{2}\sqrt{\frac{n}{2}}$ elements in R have ranks less than $\frac{n}{8}$ in N. The expected number of slopes of N of rank less than $\frac{n}{8}$ selected in R is $\frac{n}{8}/\sqrt{\frac{n}{2}}$ which is $\sqrt{\frac{n}{2}}/4$. Therefore, from Eq. 4 it follows that the probability that at least $\sqrt{\frac{n}{2}}/2$ elements in R with ranks at most $\frac{n}{8}$ in N is less than $(\frac{1}{2})^{\sqrt{\frac{n}{2}}/2}.e^{(\sqrt{\frac{n}{2}}/4)}$ which is $2^{-\Omega(\sqrt{n})}$. Therefore, it follows from Lemma 2 that

Lemma 3: For a problem of size n, the probability $\Pr[max(n_l, n_r) \leq \frac{7n}{8}]$ is at least $\frac{1}{2}$ when strategy 1 is used. When strategy 2 is used, $\Pr[max(n_l, n_r) \leq \frac{7n}{8}] \geq 1 - 2^{-\Omega(\sqrt{n})}$.

It may be noted, that these probabilities improve if one or more pairs are eliminated in Step 3 of the modified algorithm.

2.2.1 Expected running time

Let $T(n,h)$ be the expected running time of the algorithm *Randomized-Upper-Hull* to compute h extreme upper hull vertices of a set of n points, given the extreme points p_l and p_r. So the h points are in addition to p_l and p_r, which can be identified using $\frac{3}{2} \cdot n$ comparisons (this is also the best possible - see [7]) initially. Let $p(n_l, n_r)$ be the probability that the algorithm recurses on two smaller size problems of sizes n_l and n_r containing h_l and h_r extreme vertices respectively. Therefore we can write

$$T(n,h) \leq \sum_{\forall n_l, n_r \geq 0} p(n_l, n_r)(T(n_l, h_l) + T(n_r, h_r)) + bn \tag{1}$$

where $n_l, n_r \leq n-1$ and $n_l + n_r \leq n$, and $h_l, h_r \leq h-1$ and $h_l + h_r \leq h$ and $b > 0$ is a constant. Here we are assuming that the extreme point p_m is not p_l or p_r. Although, in the Basic algorithm, we have not explicitly used any safeguards against such a possibility, we can analyze the algorithm without any loss of efficiency.

Lemma 4: $T(n,h) \in O(n \log h)$.

Proof: We will use the inductive hypothesis that for $h' < h$ and for all n', there is a fixed constant c, such that $T(n', h') \leq cn' \log h'$. For the case that p_m is not p_l or p_r, from Eq. 1 we get

$$T(n,h) \leq \textstyle\sum_{\forall n_l, n_r \geq 0} p(n_l, n_r)(cn_l \log h_l + cn_r \log h_r) + bn.$$

Since $n_l + n_r \leq n$ and $h_l, h_r \leq h-1$,

$$n_l \log h_l + n_r \log h_r \leq n \log(h-1) \tag{2}$$

Let $\mathcal{E}$ denote the event that $\max(n_l, n_r) \leq \frac{7}{8}n$ and p denote the probability of $\mathcal{E}$. We know from Lemma 3 that $p \geq \frac{1}{2}$ when strategy 1 is used and $p \geq 1 - 2^{-\Omega(\sqrt{n})}$ when strategy 2 is used.

From the law of conditional expectation, we have

$$T(n,h) \leq p \cdot [T(n_l, h_l|\mathcal{E}) + T(n_r, h_r)|\mathcal{E}] + (1-p) \cdot [T(n_l, h_l + |\bar{\mathcal{E}}) + T(n_r, h_r|\bar{\mathcal{E}})] + bn$$

where $\bar{\mathcal{E}}$ represents the complement of $\mathcal{E}$.

When $\max(n_l, n_r) \leq \frac{7}{8}n$, and $h_l \geq h_r$,

$$n_l \log h_l + n_r \log h_r \leq \frac{7}{8} n \log h_l + \frac{1}{8} n \log h_r \tag{3}$$

The right hand side of 3 is maximized when $h_l = \frac{7}{8}(h-1)$ and $h_r = \frac{1}{8}(h-1)$. Therefore,

$$n_l \log h_l + n_r \log h_r \leq n \log(h-1) - tn$$

where $t = \log 8 - \frac{7}{8} \log 7 \geq 0.55$. We get the same bounds when $max(n_l, n_r) \leq \frac{7}{8}n$ and $h_r \geq h_l$. Therefore

$$\begin{aligned} T(n,h) &\leq p(cn\log(h-1) - tcn) + (1-p)cn\log(h-1) + bn \\ &= pcn\log(h-1) - ptcn + (1-p)cn\log(h-1) + bn \\ &\leq cn\log h - ptcn + bn \end{aligned}$$

Therefore from induction, $T(n,h) \leq cn \log h$ for $c \geq \frac{b}{tp}$.

In case p_m is an extreme point (say p_l), then we cannot apply Eq. 1 directly, but some points will still be eliminated according to Lemma 2. This can happen a number of times, say $r \geq 1$, at which point, Eq. 1 can be applied. We will show that this is actually a better situation, that is, the expected time bound will be less and hence the previous case dominates the solution of the recurrence.

The rank k of $slope(p_{2i-1}p_{2i})$ is uniformly distributed in $[1, \frac{n}{2}]$, so the number of points eliminated is also uniformly distributed in the range $[1, \frac{n}{2}]$ from Lemma 2. (We are ignoring the floor in $\frac{n}{2}$ to avoid special cases for odd values of n - the same bounds can be derived even without this simplification). Let $n_1, n_2 \ldots n_r$ be the r random variables that represent the sizes of subproblems in the r consecutive times that p_m is an extreme point. It can be easily verified by induction (for completeness, we have provided details in the appendix) that $E[\sum_{i=1}^{r} n_i] \leq 4n$ and $E[n_r] \leq (3/4)^r n$ where $E[\cdot]$ represents the *expectation* of a random variable. Note that $\sum_{i=1}^{r} b \cdot n_i$ is the *expected* work done in the r divide steps. Since $cn \log h \geq 4nb + c(3/4)^r \cdot n \log h$ for $r \geq 1$ (and $\log h \geq 4$), the previous case dominates. □

Remarks

(i) $\frac{b}{tp} \leq 4b$, and $\frac{b}{tp} \leq 2b$ for strategies 1 and 2 respectively, where $b \cdot n$ is the cost associated with the divide-step including median-finding. The randomized approximate median-finding takes $o(n)$ steps and in particular, strategy 1 takes $O(1)$ steps, so b is determined by the number of comparisons and *left-turn* tests in Step 4 and Step 5 of the Basic algorithm. For the $n/2$ pairs, we make at most $n/2$ comparisons and $n/2$ *left-turn* tests in Step 4 and Step 5 involves at most n *left-turn* tests. So $b \leq 2$ for *left-turn* tests and comparisons counted together in the recursive calls. By adding $\frac{3}{2} \cdot n + n$ comparisons for finding p_l and p_r and the upper-hull points initially, we obtain a grand total of $6.5n \log h$ and $10.5n \log h$ comparisons (expected) for strategy 2 and strategy 1 respectively.

(ii) The crux of Wenger's [13] analysis also uses the property that the point sets are getting split in a somewhat even manner. However, his proof uses very complicated arguments and consequently, the constants involved are very high (at least 300).

2.3 Tail Estimates

Here we will prove the following two theorems.

Theorem 1: When strategy 1 is used in Step 2, the algorithm *Randomized-Upper-Hull* terminates in $O(n \log h)$ steps with probability exceeding $1 - \frac{1}{h}$.

Theorem 2: When strategy 2 is used in Step 2, the algorithm *Randomized-Upper-Hull* terminates in $O(n \log h)$ time with probability exceeding $1 - 2^{-\Omega(n^{\frac{1}{4}})}$.

Since each leaf node in the recursion tree in the algorithm *Randomized-Upper-Hull* corresponds to an extreme point, we will make use of the following observation.

Lemma 5 The time to process a sub-problem of size n_i that contains h_i extreme points is $O(n_i \cdot h_i)$.

Proof of Theorem 1.

In the proof we will make use of the following lemma.

Lemma 6: If X is a Bernoulli random variable with success probability p, the probability that there are less than $\alpha \ln m$ successes in $3c\alpha \ln m$ independent trials is less than $m^{-\alpha}$ for

some constant $c \geq \frac{3}{2p}$.
(Here ln is the natural logarithm.)

Proof. The expected number of successes in $3c\alpha \ln m$ independent trials is $3pc\alpha \ln m$. Note that for $pc \geq 1$, $\epsilon = \frac{3pc-1}{3pc} \geq 2/3$ in Eq.5. For $pc \geq 3/2$ in Eq. 5, the probability of number of successes being less than $\alpha \ln m$ is $e^{-\alpha \ln m}$ which is $m^{-\alpha}$. $\Box$

Let us consider a node v at level $3\alpha c \ln h$ of the recursion tree for some fixed $\alpha \geq 1$. The splitting step of the algorithm (Step 6) at any node v of the recursion tree is considered successful if the problem size of each of the two children of v is at most $\frac{7}{8}size(v)$. The probability of a successful split, when strategy 1 is used, is $\frac{1}{2}$. From Lemma 6, at any node v of the recursion tree at level $3\alpha c \ln h$, for $c \geq 3$, the probability that $size(v) > (\frac{7}{8})^{\alpha \ln h} n$ is at most $h^{-\alpha}$. The probability that the bound on the size holds for *all* the nodes at depth $3\alpha c \ln h$, is clearly no more than $O(h) \cdot h^{\alpha} \leq h^{\alpha - 1}$. Therefore, for $\alpha \geq \frac{2}{ln\frac{8}{7}}$, the size of each node at level $3\alpha c \ln h$ is at most $\frac{n}{h}$ with probability $1 - \frac{1}{h^{\alpha-1}} \geq 1 - \frac{1}{h}$. Since the time spent at each level of the recursion tree is $O(n)$, $O(n \ln h)$ steps are required to process all nodes of the recursion tree upto level $3\alpha c \ln h$.
To compute work done at the remaining levels, let $v_1, v_2, \ldots v_j$ denote the nodes of $\mathcal{T}$ at level $3\alpha c \ln h + 1$. From Lemma 5, the time required to compute the upper convex hull vertices of the points stored in subtrees rooted at v_i is bounded above by $\sum_i O(h_i . \frac{n}{h})$ which is $O(n)$. $\Box$

Proof of Theorem 2.

For the analysis of Strategy 2, we will pretend that we repeat the sampling at a node v as long as a subproblem size exceeds $\frac{7}{8} \cdot size(v)$ (we will refer to this event as *bad-split*). From Lemma 3, the probability that this has to be repeated k times is less than $2^{-k\Omega(\sqrt{n})}$. In reality, we proceed with the original algorithm which behaves at least as well as this (hypothetical) modified approach. In the actual algorithm, we proceed with the subproblems even if one of subproblem sizes exceed the $\frac{7}{8} \cdot n$ at node v. If the size of this subproblem is n_1, where $\frac{7}{8} \cdot n < n_1 < n$, the probability that the size of one of its subproblems will exceed $\frac{7}{8} \cdot n$ is less than the probability that resampling at node v will exceed the $\frac{7}{8} \cdot n$ bound.

We will divide the analysis into two cases - one for $h \leq \sqrt{n}$ and the other for $h > \sqrt{n}$.

We first consider the case when $h \leq \sqrt{n}$. Let $v_1, v_2, \ldots, v_j$ be the nodes of $\mathcal{T}$ such that $size(v_i), i = 1, 2, \ldots, j$ is less than $\sqrt{n}$ and $size(parent(v_i)), i = 1, 2, \ldots, j$ is at least $\sqrt{n}$.

Let h_j be the number of upper hull vertices of the set represented by the node v_j. Total work done at the subtrees of the recursion tree with roots at $v_1, v_2, \ldots, v_j$ is bounded above by $\sum_{i=1}^{j} h_i size(v_i)$ which is $O(h \cdot \sqrt{n})$. Since $h \leq \sqrt{n}$, the expected running time of the algorithm using strategy 2 is dominated by the time needed to process the nodes of recursion tree which represented subproblems of size at least $\sqrt{n}$. Since there are at most $\sqrt{n}$ such nodes in $\mathcal{T}$, it follows from Lemma 3 that the probability of successful splitting at all of these nodes is $1 - 2^{-\Omega(n^{1/4})}$. If all the splits are successful, we know that the total time is no more than $O(n \log h)$ (from the deterministic version of the algorithm). Therefore, algorithm *Randomized-Upper-Hull* takes $O(n \log h)$ time to compute all h upper hull extreme points with probability $1 - 2^{-\Omega(n^{\frac{1}{4}})}$ when $h \leq \sqrt{n}$.

We now consider the case when $h > \sqrt{n}$, so $\log h \in \Omega(\log n)$. Let us partition the nodes of the recursion tree $\mathcal{T}$ into the following four classes.

1. $N_1 = \{x \in T : size(x) \geq n^{\frac{1}{2}}\}$

2. $N_2 = \{x \in T : \log^2 n \le size(x) < n^{\frac{1}{2}}\}$

3. $N_3 = \{x \in T : \log n \le size(x) < \log^2 n\}$

4. $N_4 = \{x \in T : \log n > size(x)\}$

Clearly, each of these sets have less than n elements.

For the nodes in N_1, the probability that we do not have to resample for any node of N_1 is at least $1 - 2^{-\Omega(n^{\frac{1}{4}})}$. This follows from Lemma 3.

For nodes in N_2, the probability that sampling has to be repeated for a fixed node is less than $2^{-\Omega((\log^2 n)^{\frac{1}{2}})}$ which is $O(\frac{1}{n})$ (from Lemma 3). Therefore, the expected number of nodes where sampling has to be repeated is no more than some constant. Therefore, from Eqn.(4) we get that the probability that resampling has to be repeated for $n^{\frac{1}{4}}$ nodes is less than $(\frac{1}{n^{\frac{1}{4}}})^{n^{\frac{1}{4}}}.e^{n^{\frac{1}{4}}}$ which is at most $2^{-\Omega(n^{\frac{1}{4}})}$ for $n > 256$. We are assuming here that sampling is done independently at each node. The $n^{\frac{1}{4}}$ nodes collectively contain less than $O(n^{\frac{3}{4}})$ elements. By repeating sampling $n^{1/4}$ times for these nodes, the probability of a bad split is less than $2^{-n^{1/4}}$. Therefore, with probability $1 - 2^{-\Omega(n^{\frac{1}{4}})}$, the nodes of N_2 can be processed in $O(n)$ time.

For nodes in N_3, the probability of repeating sampling is less than $2^{-\sqrt{\log n}}$ and so the expected number of nodes for which we may have to repeat sampling is less than $n/2^{\sqrt{\log n}}$. Call these nodes N_3' and the probability that N_3' exceeds $n/\log^{10} n$ is less than $2^{\Omega(-n/\log^{10} n)}$. (The choice of $\log^{10} n$ has no special significance, other than that it is large enough for our purpose.) If we resample $\log n$ times at each node of N_3', the probability that the splitting is unsuccessful (after $\log n$ resampling) is at most $O(\frac{1}{n})$. Total work involved is $o(n)$ as the size of each of the $n/\log^{10} n$ nodes is less than $O(\log^2 n)$. Applying the previous argument once again to the unsuccessful nodes of N_3', we can claim that the nodes of N_3 can be processed in $O(n)$ time with probability $1 - 2^{-\Omega(n^{\frac{1}{4}})}$.

We note that for a node v of size less than $\log n$, at most $(size(v))^2 \le size(v) \cdot \log n$ time is required to process the subtree rooted at v. Since the points associated with the nodes of N_4 are disjoint, the total time can be bounded by $O(n \log n)$.

Therefore, when strategy 2 is applied in step 2, the proposed algorithm terminates in $O(n \log h)$ time with probability exceeding $1 - 2^{-\Omega(n^{\frac{1}{4}})}$. □

3 Experimental Result and Conclusions

To compare the two strategies, we ran these algorithms on data-sets upto size 250,000 points. We also ran experiments for different distributions, namely, uniform in a box, uniform in a disc, uniform in an annulus, all points on a circle. The average is computed on the basis of 10 trials. The running time statistics are shown in Table 1. The performances of the two strategies are very comparable in terms of time and also the number of left-turn/right-turn tests - within 20% of each other.

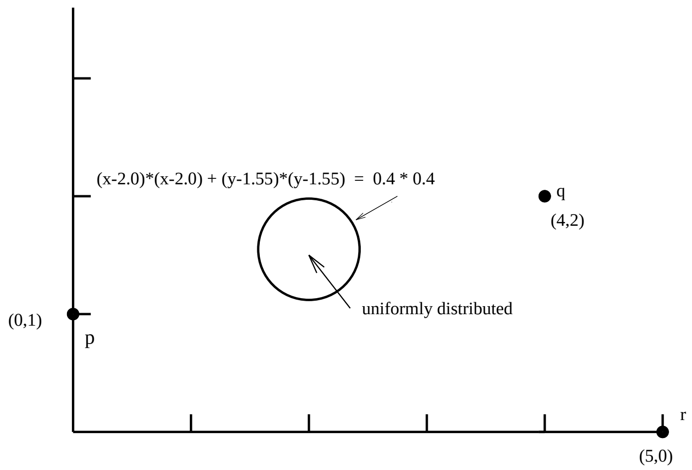

Figure 2: The data set contains p, q and r and the points distributed uniformly in the disc

We also compared these two strategies with the randomized quickhull as proposed in [12]. The performance of quickhull is very comparable with the performances of the two strategies for the data sets distributed uniformly in a box, in a disc, in an annulus, and on a circle. We also also tested these algorithms on data sets (called customized in Table 1) on which quickhull performed very poorly in comparison with the other two. Visual description of this data set is shown in Fig. 2.

For the case when all points are on the hull, our algorithms took less than three times the time required to sort the points and for the smaller outputs our algorithms out-performed sorting by a factor of three. Since the worst-case $O(n \log n)$ algorithms like Graham's scan take more time than sorting, we feel that our algorithm is a practical and provably efficient alternative to the present planar hull algorithms. Although Wenger's algorithm looks somewhat different, we believe that its actual performance is comparable to ours, especially if we do away with generating $O(n)$ random pairs for each recursive call.

Acknowledgement

The authors wish to acknowledge the facilities provided by Department of Computer science at The University of Newcastle, Australia where a part of this research was carried out. The authors also thank the anonymous referee who pointed out an error in an earlier version of the Basic algorithm and also made several useful comments about the presentation of the proof of Lemma 4.

References

[1] T.M. Chan. Output-sensitive construction of convex hulls. *Ph.D. Thesis, University of British Columbia*, 1995.

[2] T.M. Chan, J. Snoeyink and C. Yap. Output-Sensitive construction of polytopes in four dimensions and clipped voronoi diagrams in three. *Proc. of the 6-th ACM-SIAM Symp. on Discrete Algorithms*, pp. 282–291, 1995.

[3] D.R. Chand and S.S. Kapur. An algorithm for convex polytopes. *Journal of the ACM*, 17:78-86, 1970.

	Uniform in box (h (average) = 15)	Uniform in disc (47)	Uniform in annulus (98)	Uniform on circle (10,000)	Customized (4)
Strategy 1	16159 113	16480 130	41557 296	182030 1343	19554 129
Strategy 2	16694 130	18866 153	43696 286	194138 1406	23317 172
Quickhull	16218 106	20352 156	37061 280	132983 1243	368947 1792

10,000 points

	Uniform in box (h (average) = 17)	Uniform in disc (81)	Uniform in annulus (164)	Uniform on circle (50,000)	Customized (4)
Strategy 1	80365 666	81331 759	194542 1690	1079605 9226	97413 745
Strategy 2	81842 773	93622 943	211805 1773	1125749 9769	107964 895
Quickhull	100046 836	110609 953	185780 1590	819236 8736	993916 5458

50,000 points

	Uniform in box (h (average) = 19)	Uniform in disc (132)	Uniform in annulus (283)	Uniform on circle (250,000)	Customized (5)
Strategy 1	399569 3446	429617 4230	1008051 8770	6362528 60976	486171 3750
Strategy 2	403184 3763	490443 4780	1105597 9769	6496419 63169	516184 4172
Quickhull	525401 4176	553573 4619	1050328 9029	4710908 54923	7618098 41706

250,000 points

Table 1: Tables comparing the average performances of the two strategies for various distribution. The first entry is the number of left-turn primitives and the second entry is the CPU time on a Sun SPARCstation 10.

[4] M.E. Dyer. Linear time algorithms for two- and three- variable linear programs. *SIAM Journal on Computing*, 13:31-45, 1984.

[5] R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132-133, 1972.

[6] R.A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2:18-21,1973.

[7] D.E. Knuth. The Art of Computer Programming- Volume 3 : *Sorting and Searching.* Addison-Wesley, 1973.

[8] D. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15:287-289, 1986.

[9] N. Megiddo. Linear time algorithm for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 30:852-865, 1983.

[10] F.P. Preparata and M.I. Shamos. *Computational Geometry: an Introduction*, Springer-Verlag, New York, 1985.

[11] R. Seidel. Backward analysis of randomized geometric algorithms. In *New Trends in Discrete and Computational Geometry*, J. Pach, Ed., vol. 10 of *Algorithms and Combinatorics*, Springer-Verlag, pp. 37-38, 1993.

[12] L. Shafer and W. Steiger. Randomizing Optimal Geometric Algorithms. *Canadian Conference on Computational Geometry*, 1993, pp. 133–138.

[13] R. Wenger. Randomized quick hull, *to appear*, Algorithmica.

[14] A. Yao. A lower bound to finding convex hulls.*Journal of the ACM*, 28:780-787, 1981.

A Appendix

The equation (1) is the Chernoff bounds for Binomial distributions and equations (2) and (3) are due to Angluin and Valiant.

Lemma A.1: If X is binomial with parameters (M, p), where M is the number of independent trials and p is the probability of success in a trial, for $t > Mp$, t an integer,

$$\Pr(X \geq t) \leq (Mp/t)^t \exp(t - Mp) \tag{4}$$

$$\Pr(X \leq \lfloor(1-\epsilon)pM\rfloor) \leq \exp(-\epsilon^2 Mp/2) \forall 0 < \epsilon < 1. \tag{5}$$

$$\Pr(X \geq \lfloor(1-\epsilon)pM\rfloor) \leq \exp(-\epsilon^2 Mp/3) \forall 0 < \epsilon < 1. \tag{6}$$

Consider the following probabilistic experiment. From a given set of n ordered elements we choose an element at random. If we choose the rth element, then we repeat the experiment on a ordered set of r elements $(1 \leq r \leq n)$. We continue with this process until we are left with one element. We denote these sets by $N_0, N_1 \ldots$ and their cardinalities by $n_0, n_1 \ldots$ where $n_0 = n$. We are interested in bounding the expected values of $S_n = \sum_i n_i$ and n_k.

Claim 1 $E[S_n] \leq 4n$.

Proof We will prove it by induction on n. Clearly it is true for $n = 1$. Assume it is true upto $n - 1$. In the first trial we choose a random element from a set of n elements. From the law of conditional expectation

$$E[S_n] = \frac{1}{n}(E[S_n] + E[S_{n-1} + \ldots) + n$$

By substituting the induction hypothesis, we obtain the required result.

Claim 2 $E[n_k] \leq (3/4)^k \cdot n$

Proof We use induction on k. Clearly it is true for $k = 0$. Assume it is true upto $k - 1$. The $\Pr[n_k \geq n_{k-1}] \leq 1/2$. So from the law of conditional expectation

$$E[n_k] \leq \frac{1}{2} E[n_{k-1}/2] + \frac{1}{2} E[n_{k-1}]$$

The claim follows by substituting $n_{k-1} = (3/4)^{k-1}$.

Note that both the above claims can be tightened using better analysis but the above bounds are sufficient for our purpose.