CompGeometry handin 2

Mads Toftrup
Jens Kristian Refsgaard Nielsen

October 29, 2021

Contents

1 Test data **2**
1.1 Square . . . 2
1.2 Disc . . . 2
1.3 Circle . . . 3
1.4 Fixed or Circle+disc . . . 3
1.5 Polynomial curve . . . 3

2 Part A: Graham scan **3**
2.1 Graham scan evaluation . . . 3
2.2 Sorting . . . 3
2.3 Graham scan experimentation . . . 4

3 Part B: Gift wrapping **4**
3.1 Experimentation . . . 4

4 Part C: Chan's Algorithm implementation **5**
4.1 Experimentation . . . 5

5 Algorithm comparison **6**
5.1 Time measurements . . . 8

6 Algorithmic correctness **8**

7 Part E Chan's Algorithm analysis **9**

8 Part F: Marriage before conquest analysis **10**

9 Time measurements **11**

A Code **12**

1 Test data

We use 5 different types of test data to test our convex hull implementations.

Data distribution	Square	Disc	Circle	fixed h = 5	Polynomial curve
Convex hull size	$O(\log(n))$	$O(n^{1/3})$	n	h	n

1.1 Square

Sampling points on a square is very simple, take two uniformly random $[0,1]$ values and you have a uniformly random point from $[0,1]^2$.
The source below shows that the size of the convex hull of such a square is $\log(n)$ where n is the number of sampled points.
Link
Our experiments support this claim

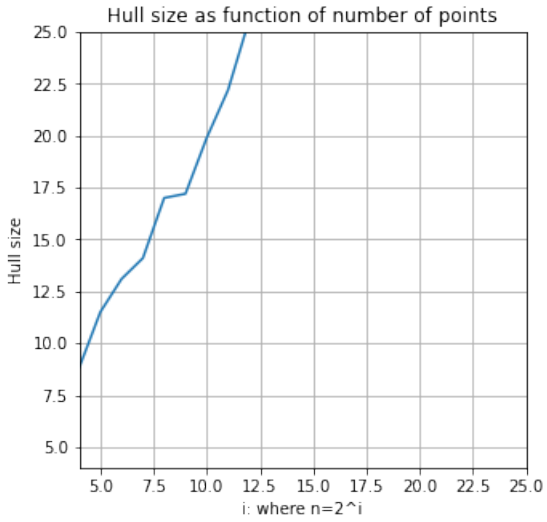


Figure 1: Hull size of a square

By plotting with a logarithmic x-axis we can conclude that in practice the hull size of square has a log(n) dependency on the number of points, which a constant factor of approximately 2.

1.2 Disc

A radius 1 disc is a set

$$D = \{x \in \mathbb{R}^2 \mid \|x\| \leq 1\}$$

we sample this by sampling from a $[-1,1] \times [-1,1]$ square and removing points with norm above 1. This probability that a uniformly sampled square point is within the radius 1 disc is $\pi/4 \approx 79\%$ so the overhead of this method is not too bad.
Running experiments we find that the convex hull of a disc is of size about $O(n^{1/3})$, which we see from the log log plot below that has a slope of approximately 1/3

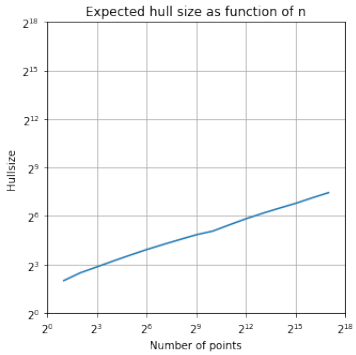


Figure 2: Hull size of a disc

1.3 Circle

Sampling points on the unit circle boundary is actually easier, sample an angle θ from $[0, 2\pi]$ and compute the point $(\cos(\theta), \sin(\theta))$ this is a unit vector by definition, up to floating point errors. As such the convex hull trivially must include every point. This data structure is not used for experiments as it fulfills the same role as the polynomial curve, namely that of having all points on the hull

1.4 Fixed or Circle+disc

To get a data structure where we can control the hull size h as an input variable, we take a radius 2 circle of size h and embed a radius 1 disc of size $n-h$ inside it. This might not work if the circle points are sampled unluckily, but if we enforce $h \geq 4$ we can take the 4 points $(\pm\sqrt{2}, \pm\sqrt{2})$ which are the corners of a length 2 square which fully encapsulate the radius 1 disc.

1.5 Polynomial curve

Finally we take the right leg of the curve $f(x) = (x, x^2)$ by taking small steps $x_{k+1} = x_k + r$ for r sampled uniformly from $[0, 1]$, now this can cause some roundoff errors in the sidedness tests if r is close to 0, so sampling from $[0.000001, 1]$ is preferred. Shuffling the list after generating the points ensures that the sequencing does not interfere with our results.

2 Part A: Graham scan

2.1 Graham scan evaluation

Graham scan consists of sorting the points which is done in $O(n \log(n))$ and then adding each point to the hull and removing it if it is actually redundant, each point is removed at most once, so this runs in $O(n)$ for a combined $O(n \cdot \log(n))$. The bottleneck of Graham scan is then the sorting algorithm used.

2.2 Sorting

We found a minor difference in asymptotic runtime between the sorted method in python and mergesort, neither was able to achieve a perfect $nlog(n)$ runtime. But notice that the x-axis is in millions of figure 3. Even though mergesort was the furthest from the theoretical $nlog(n)$ we used this to implement graham scan because it enabled us to use a software counter for the number of comparisons. By using software counters we overcome much noise of other factors impacting the performance.

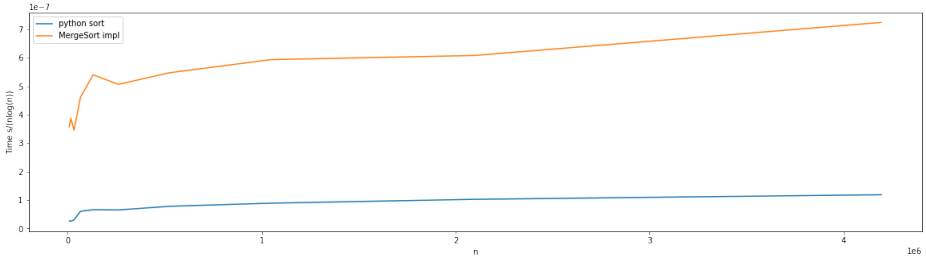


Figure 3: Time comparison between python library sort and Geek4Geeks mergesort implementation

2.3 Graham scan experimentation

With the software counters implemented in mergesort and left turn we run on the square, curve, circle and fixed data structures. We expect the following behaviour

Data distribution	Square	Disc	fixed h = 5	Polynomial curve
Convex hull size	$O(\log(n))$	$O(n^{1/3})$	h	n
Graham scan expected	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$

And observe

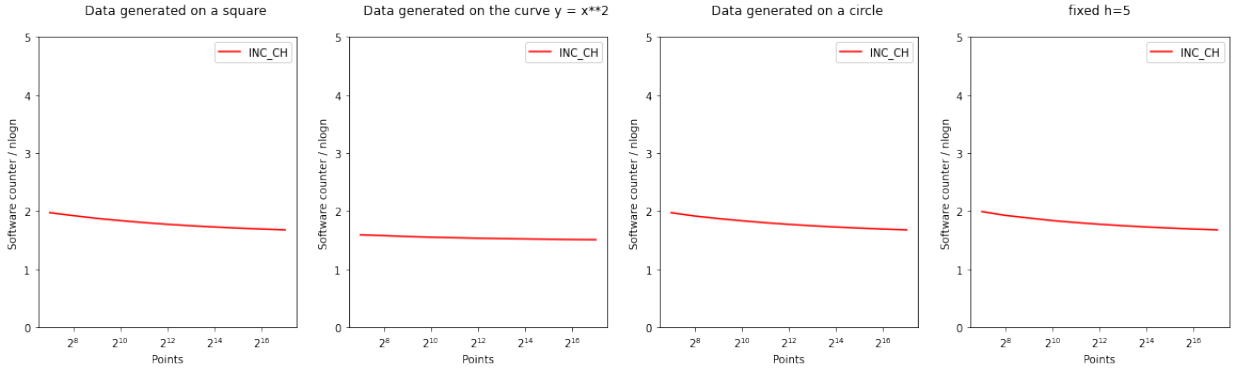


Figure 4: Software counter experiments on Graham scan

In accordance with the table we divide the observed software counter with the expected function. Observe that all lines above are constant, so our experiments support our hypothesis.

3 Part B: Gift wrapping

The gift wrapping algorithm works by optimizing a ray by trying all n and then adding the point whose ray has the highest value to the hull. This is done for each point in the hull, giving a $O(nh)$ algorithm.

3.1 Experimentation

We run the same experiments as for Graham scan, but expect different results based upon the following table

Data distribution	Square	Disc	fixed h = 5	Polynomial curve
Convex hull size	$O(\log(n))$	$O(n^{1/3})$	h	n
Graham scan expected	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$
Gift wrapping expected	$O(n\log(n))$	$O(n^{4/3})$	$O(n)$	$O(n^2)$

We observe

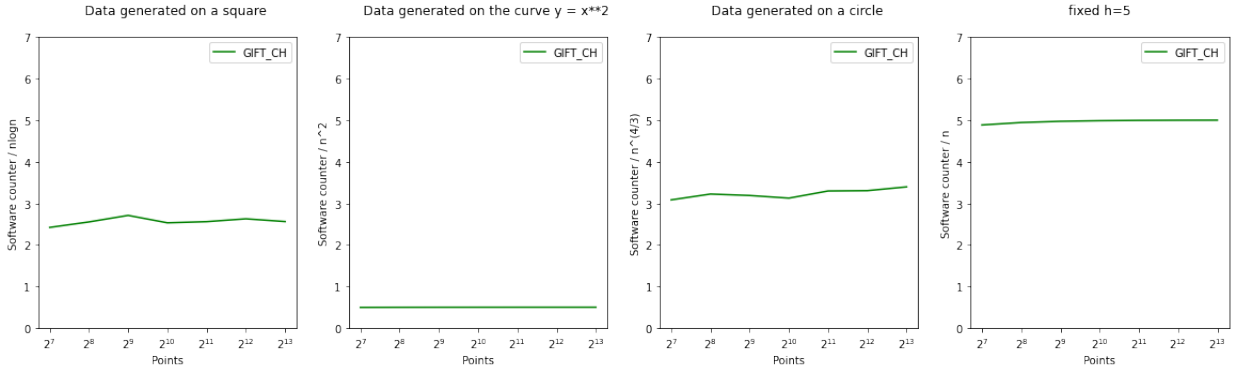


Figure 5: Software counter experiments on Gift wrapping

In accordance with the table we divide the observed software counter with the expected function. Observe that all lines above are constant, so our experiments support our hypothesis. There is some slight noise in the value of the constant.

4 Part C: Chan's Algorithm implementation

For a detailed analysis of the runtime of our implementation of Chan's algorithm see 7.
To achieve the theoretical runtime, we noted that in python it is linear in the length of the list to remove a point from the start of the list, while it is constant to pop from the back. To fix this we mirrored the dataset such that removing the first point can be done with a pop operation.
We used binary search to compute the tangent of a point to a hull, with the observation that the tangent makes a left turn with every element "to the left" of it on the hull and a right turn with every element to the right. But as everything is mirrored this becomes branching right when not left_turn(p,pts[mid+1],pts[mid]), i.e. when a right turn is made between p p[mid] and p[mid+1] in the non-mirrored case.
To compute the lower hull we initialize p to be the point with the maximal x value and run from right to left looking for the minimum x value. We also pop if the hull point is to the right of p, instead of the left as is proper for the upper hull.

4.1 Experimentation

You know the drill by now

Data distribution	Square	Disc	fixed h = 5	Polynomial curve
Convex hull size	$O(\log(n))$	$O(n^{1/3})$	h	n
Graham scan expected	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$
Gift wrapping expected	$O(n\log(n))$	$O(n^{4/3})$	$O(n)$	$O(n^2)$
Chans expected	$O(n\log(\log(n)))$	$O(n\log(n))$	$O(n)$	$O(n\log(n))$

We observe

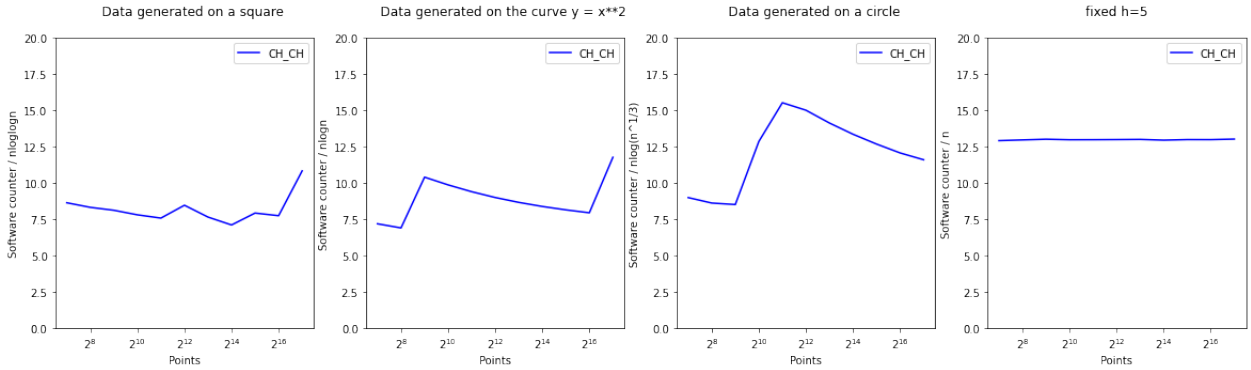


Figure 6: Software counter experiments on Chans algorithm

In accordance with the table we divide the observed software counter with the expected function. Notice the spikes in runtime when the hull size is not a constant, these are due to Chan running an extra iteration of the outermost loop. Chan guesses that the hull is of size 2^{2^i} so when the true size is $2^{2^i} + j$ for relatively small j we expect to observe a spike in the number of left turns made.

5 Algorithm comparison

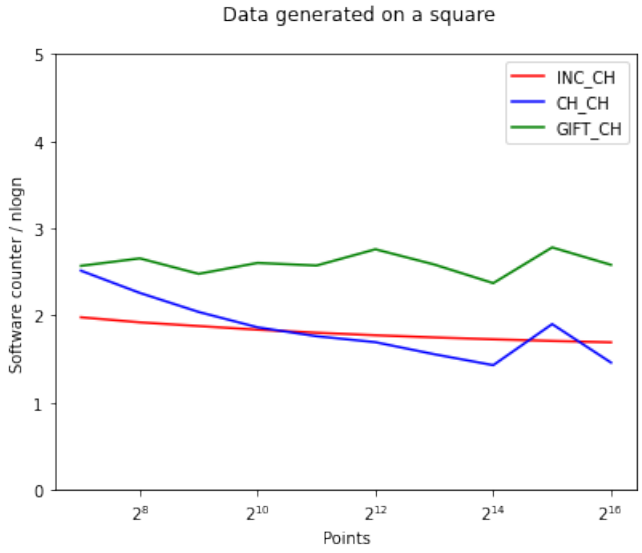


Figure 7: Software counter experiments for all algorithms

Running on square we see that as expected both gift and graham scan runs in $O(n\log(n))$ time while $Chan$ beats them with a theoretical $O(n\log\log(n))$

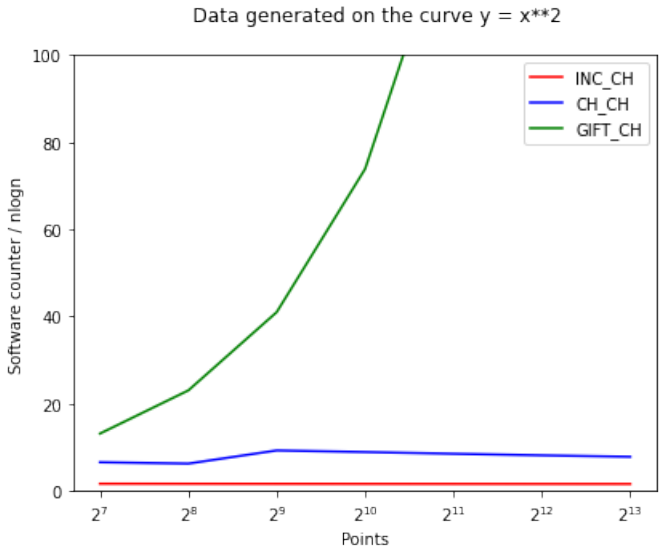
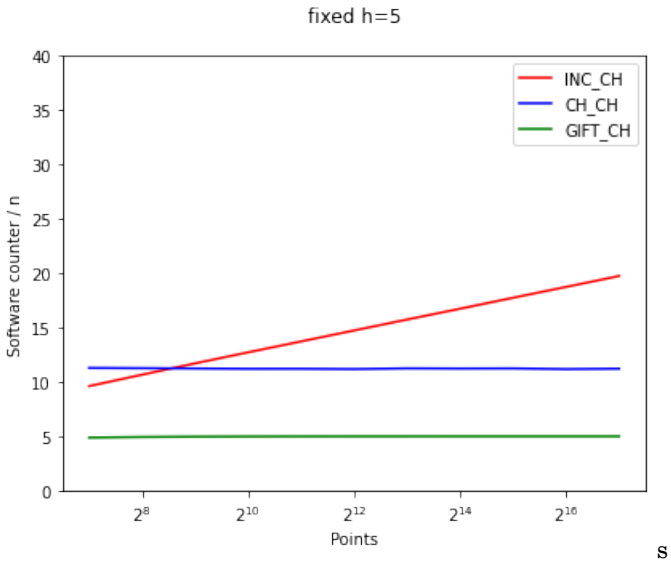


Figure 8: Software counter experiments for all algorithms

On the parabola input both $Chan$ and Graham is supposed to run in $O(n \log(n))$ while giftwrapping runs slower asymtotically at n^2



s

Figure 9: Software counter experiments for all algorithms

Running with a fixed hull size, we see that both Chan and Gift outperforms graham scan asymptotically as expected. Graham scan has a linear line because the x-axis is logarithmic and all values have been divided by n

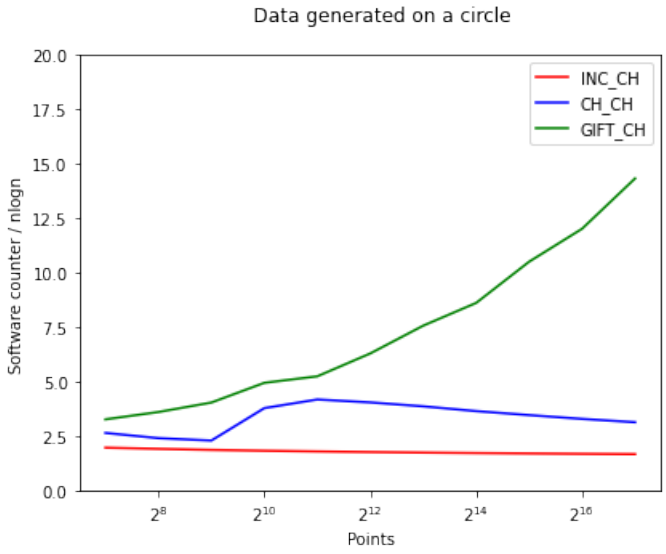


Figure 10: Software counter experiments for all algorithms

When generating points on a circle the expected hull size is $n^{1/3}$ and thus both graham and chan runs in $O(n\log(n))$ while gift wrapping runs in $O(n^{4/3})$ and is holds that $O(n\log(n)) > O(n^{4/3})$ as is seen on the plot.

5.1 Time measurements

Time observations

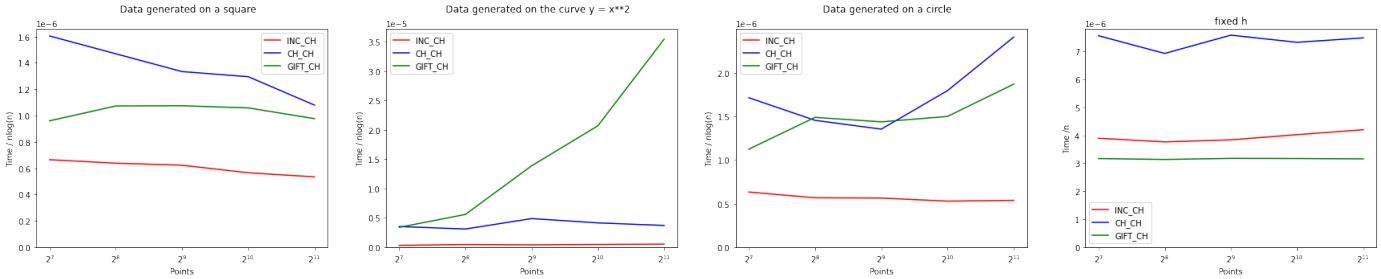


Figure 11: Time experiments

We see similar asymptotic runtime tendencies as with software counters, but with more noise. The plots are more spiky than the software counters. Additionally chan is worse than gift wrapping on all except the parabola due to overhead. but this might cancel out with bigger inputs.

6 Algorithmic correctness

To test whether our implementations of the convex hull algorithms is correct, we use Scipy.spatial. ConvexHull and test that the 4 algorithms compute the same set of points as the hull. When a discrepancy is detected, the hulls are plotted for visual inspection.

We observe that there is a slight error with Chan's lower hull algorithm, that happens with frequency $\frac{5}{4 \cdot 300} = 0.416\%$. The error is likely an edge case of the binary search that is unreachable in upper hull computation.

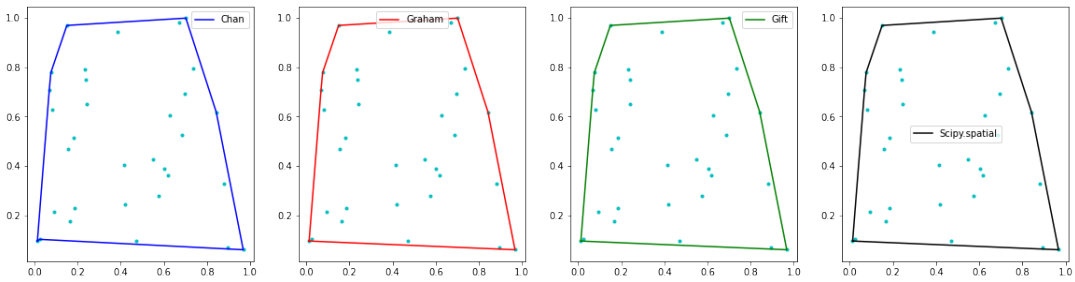


Figure 12: Chan lower hull error

The error is that the lower hull picked the second to last point instead of the last point.

7 Part E Chan's Algorithm analysis

Chan's Algorithm consists of two procedures, CH_CH and UHwithSize.
Ch_CH runs UHwithSize in the worst case log(log(n)) times with ascending values of i, this i defines a guess on an upper bound of the hull size defined as $h_i = 2^{2^i}$, a vital step is that the algorithm terminates the first time h_i is greater than the true upper hull size essentially solving the following for i

$$2^{2^{i-1}} < h_{\text{true}} \leq 2^{2^i}$$

while also computing the upper hull. We then find an initial runtime of

$$\sum_{i=1}^{\log\log(h)} \text{UHwithSize}(P, h) \tag{1}$$

Now let us analyze UHwithSize, first it partitions in $O(\frac{n}{h} \cdot h) = O(n)$ time, then we use Graham's Scan to compute convex hulls on the $m = \frac{n}{h}$ partitions of size h in

$$\frac{n}{h} \cdot \text{GrahamScan}(h) = \frac{n}{h} \cdot O(h \log(h)) = O(n \log(h)) \tag{2}$$

In the h loop we compute the tangent to the m upper hulls produced earlier, we can compute the tangent with binary search, after computing the m tangents, we compute the tangent with the m smaller convex hulls, potentially of size h, to find the best one in accordance with the algorithm. In total the loop takes

$$\begin{aligned} h \cdot (m \cdot O(\log(h)) + \log(m)) &= O\left(h \cdot \left(\frac{n}{h} \cdot \log(h)\right)\right) \\ &= O(n \log(h)) \end{aligned}$$

In total the worst case runtime of UNwithSize(P, h) becomes $O(n \log(n))$.
Let h be the true upper hull size, bringing the above considerations together we find the runtime of

Chans algorithm to be

$$\begin{aligned}\sum_{i=1}^{\log\log(h)} n\log(2^{2^i}) &= \sum_{i=1}^{\log\log(h)} n2^i \\ &= n\sum_{i=1}^{\log\log(h)} 2^i \\ &= n2(2^{\log\log(h)}-1) \\ &= n2(\log(h)-1) \\ &= O(n\log(h))\end{aligned}$$

Note that we also implemented the lower hull in the completely analogous way, say h_u is the size of the upper hull and h_ℓ is the size of the lower hull, then the runtime becomes

$$O(n\log(h_u)+n\log(h_\ell)) = O(n\log(h_u \cdot h_\ell)) \tag{3}$$

This is not an issue however, as say $h_u = h_\ell = n/2$ then

$$O(n\log(h_u \cdot h_\ell)) = O(n\log(n^2/4) = O(n\log(n)) \tag{4}$$

8 Part F: Marriage before conquest analysis

Claim: The MbC_CH algorithm runs in $O(n\log(h))$ time where h is the number of points of the convex hull.
Proof: The algorithm follows the following 5 steps:

1. Find the point with median x coordinate $p_m = (x_m, y)$ and partition the input into two sets P_ℓ and P_r where P_ℓ contains all the points with x-coordinate smaller than x_m and P_r contains the rest of the points.

2. Find the point p_ℓ with the smallest x-coordinate (if there are more than one, take the one with the largest y-coordinate) and the point p_r with the largest x-coordinate (if there are more than one, take the one with the smallest y-coordinate).

   Prune all the points that lie under the line segment $p_\ell p_r$.

3. Find the "bridge" over the vertical line $X = xm$ (i.e., the upper hull edge that intersects line $X = x_m$).

   Let (x_i, y_i) and (x_j, y_j) be the left and right end points of the bridge.

4. Prune the points that lie under the line segment $(x_i, y_i),(x_j, y_j)$ (these will be the points whose x-coordinate lie between x_i and x_j.

5. Recursively compute the upper hull of P_ℓ and P_r.

To analyse this algorithm recall from the lectures that step **1** can be performed using median of medians in $O(n)$ time.
Doing step **2** can simply be done with a linear scan to find the points, followed by another linear scan with a sidedness test to determine which points are below and have to be pruned, in $O(n)$ time.
Finding the bridge of step **3**, we saw at the lectures can be expressed as a linear program that can be solved in linear time. $O(n)$
For step **4** another linear scan to prune points under the bridge, $O(n)$
Finally the interesting step is the recursion of step **5**. Up to now we have established the recursion

$$f(n,h) = f(n/2, h_\ell) + f(n/2, h_r) + O(n) \tag{5}$$

We know that there are $n/2$ points in each partition since we chose the median, not the average. However the recursion depth can potentially be h as the recursion tree is unbalanced, say the left branch can be covered by a single bridge while the right branch is a half circle where every point will be on the hull. In this case we get recursion depth $h_r = h - 1$. However this turns out to be inconsequential for the analysis.
We prove that $f(n,h) = O(n\log(h))$ by induction on h.
if $h = 1$ all points are on a vertical line, which is a very uninteresting case, so let us assume $h \geq 2$ and proceed with the induction.
Induction start: Assume that $h = 2$, then all points are pruned in step 4, if we use soft inequality. Some points could have the same x values as the bridge points, but by step 2, they have lower y values and are pruned. We have then performed $O(n)$ work and the induction stands $f(n,2) = O(n) \leq cn \cdot \log_2(2)$ as wanted. We use this constant c for the asymptotic notation throughout the proof.
Induction step: From here the argument is purely mathematical, note that we must have $h = h_\ell + h_r$. Now the induction hypothesis states $f(n,h') \leq cn\log(h')$ for $1 \leq h' < h$. Consider then

$$\begin{aligned}
f(n,h) &= f(n/2,h_\ell) + f(n/2,h_r) + O(n) \\
&\leq f(n/2,h_\ell) + f(n/2,h_r) + cn \\
&\overset{\text{I.H.}}{\leq} c\frac{n}{2}\log(h_\ell) + c\frac{n}{2}\log(h_r) + cn \\
&= c\frac{n}{2}\log(h_\ell h_r) + cn \\
&\leq c\frac{n}{2}\log\left(\frac{h^2}{2^2}\right) + cn \\
&\leq cn\log(h/2) + cn\log(2) \\
&\leq cn\log(h)
\end{aligned}$$

Where we use the following argument to see that $h_\ell h_r \leq h^2/4$
Since $0 < h_\ell, h_r$ and $h = h_\ell + h_r$ we can write $h_\ell = \epsilon h$ and $h_r = (1-\epsilon)h$ for some $\epsilon \in (0,1)$. Consider then

$$\begin{aligned}
h_\ell h_r \leq h^2/4 &\iff \epsilon h(1-\epsilon)h \leq h^2/4 \\
&\iff \epsilon(1-\epsilon) \leq 1/4 \\
&\iff \epsilon - \epsilon^2 - 1/4 \leq 0
\end{aligned}$$

Consider the last equation as a polynomial in ϵ then we quickly see that it is a non-positive function with double root $\epsilon = 1/2$ and the inequality holds. □

9 Time measurements

Here we include the analogous plots of runtime analysis, but with seconds instead of software counters. Recall the expected behaviour table.

Data distribution	Square	Disc	fixed h = 5	Polynomial curve
Convex hull size	$O(\log(n))$	$O(n^{1/3})$	h	n
Graham scan expected	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$
Gift wrapping expected	$O(n\log(n))$	$O(n^{4/3})$	$O(n)$	$O(n^2)$
Chans expected	$O(n\log(\log(n)))$	$O(n\log(n))$	$O(n)$	$O(n\log(n))$

Time observations

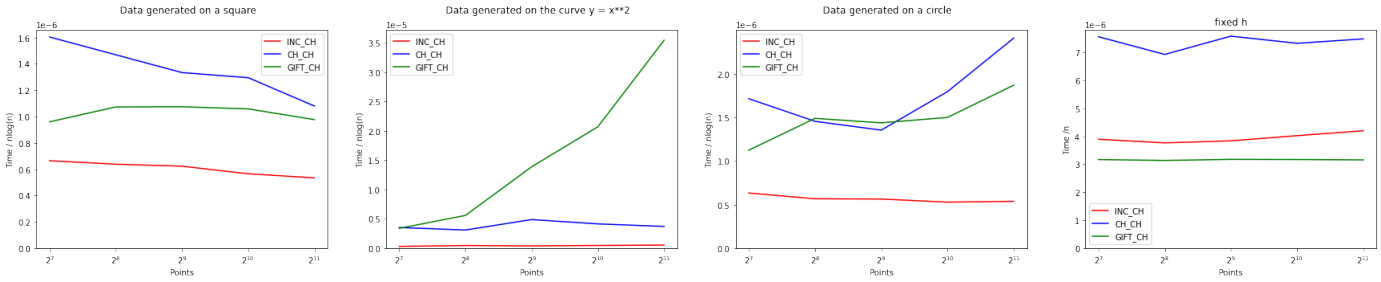


Figure 13: Time experiments

We see similar asymptotic runtime tendencies as with software counters, but with more noise, for example chan being worse that gift wrapping on all except the parabola due to overhead.

A Code

Github repository