MacKenzie K. Cooper
CS 584, Algorithm Design & Analysis
Portland State University
March 17, 2019

Implementing Convex Hull:
A Comparison between Graham and Jarvis Algorithms

Introduction

The purpose of this paper is to quantitatively analyze, compare, and contrast two implementations of algorithms to solve the Convex Hull problem. The Graham Scan and Jarvis March algorithms will be the key subjects of analysis, as they are interpreted, implemented, and tested given various inputs.

What is Convex Hull?

The Convex Hull Problem is a problem in computational geometry. The convex hull is the smallest convex polygon that contains all the vertices of a given set on a multi-dimensional Euclidean plane. For the purposes of this paper, the Convex Hull Problem will be limited in scope to the 2-dimensional Euclidean plane. The problem has numerous practical applications, including, but not limited to, pattern recognition, image processing, and Voronoi tessellations.
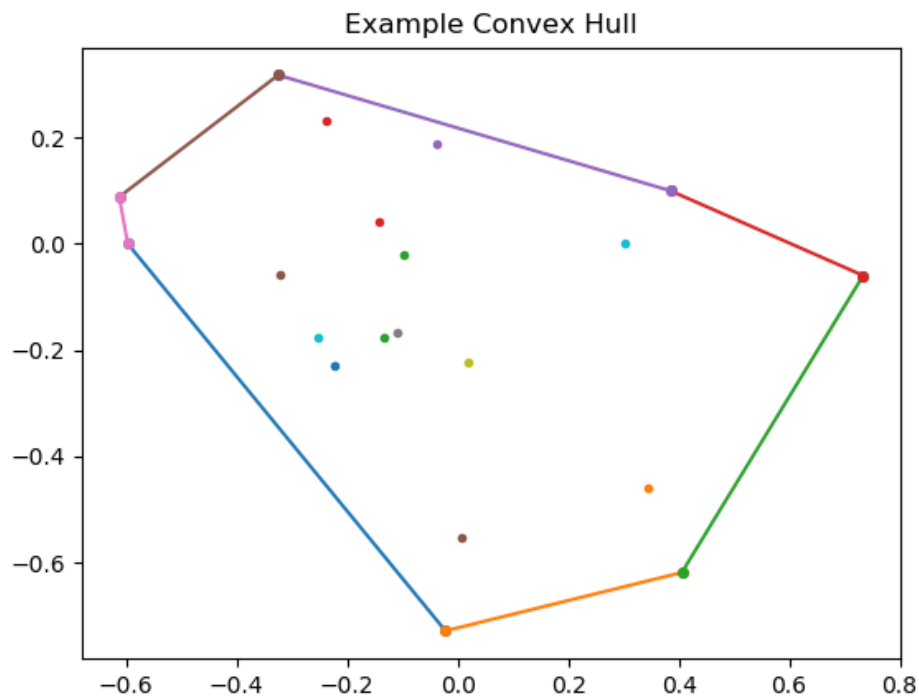


*Figure 1: Example of a Convex Hull Solution*

## Graham Scan Algorithm

This algorithm was first published by Ronald Graham in 1972, giving its name. First, find a vertex guaranteed to be on the hull by finding the most extreme vertex in any given direction (up, down, left, right), choosing the most extreme in a different axis if there are any ties. In relation to our starting vertex, we calculate the polar angle of all other vertices in the set and then sort them in counter-clockwise order. In the case of any vertices with the same polar angle, we eliminate all but the furthest from our starting vertex. We take an empty stack and push our starting vertex and the rightmost vertex from our now sorted set onto it. Then we loop across the remaining vertices, pushing them each onto the stack, and popping from the stack while the current vertex is collinear or to the right of the line formed by the vertices second-to-top of the stack and the top of the stack, until once again, we form a left turn. This loop continues until all points are considered, at which point all the vertices on the stack are the vertices on the hull in counter-clockwise order, starting with our start vertex.

## Jarvis March Algorithm

This algorithm was first published by Ray Jarvis in 1973, giving its name. First, as above in Graham Scan, find a vertex guaranteed to be on the hull. Enter a loop, choosing another vertex to be our candidate vertex arbitrarily from the remainder of the set. Enter a subloop comparing all other vertices to the line formed by our last vertex on the hull and our candidate vertex. Should any vertex not form a left turn, we replace our candidate vertex with this comparison vertex and continue. At the end of this subloop, our candidate vertex will be the most counter-clockwise and we add it to our solution set along with our starting vertex. The outer loop will continue until our candidate vertex is found to be our starting vertex, therefore completing our polygon.

## Complexity

The lower bound of Convex Hull was established to be $\Omega(n \log n)$ by Andrew Chi-Chih Yao in 1981. To summarize, the complexity of this problem can be reduced to the complexity of a sort, which has been established as $\Omega(n \log n)$, thus giving us the lower bound for this problem.

The sort portion of the Graham algorithm has a complexity of $\Theta(n \log n)$. Upon first inspection, the Graham algorithm appears to be $O(n^2)$ because of the nest loops. However, because any given vertex is only inspected at most twice, the complexity of the search portion of the algorithm is linear. Thus, the sort provides the complexity for Graham Scan.

The outer loop for Jarvis March is across all hull vertices. This gives us a complexity of $\Theta(h)$, where h is the number of hull vertices. The inner loop is across all vertices, giving us a complexity of $\Theta(n)$. Combined, we get a complexity of $\Theta(nh)$. Because this complexity is dependent on the number of vertices that lie on the hull, meaning it is output sensitive. This would make it more efficient on inputs where the number of hull vertices is less than some log of the input size. However, this also means it has a worst case of $\Theta(n^2)$ when all vertices are on the hull.

**GrahamScan(V)**
Description: Finds the convex hull from a set of vertices in a Euclidean plane
Input: Array $V$ of vertices of size $n$
Output: Ordered stack $H$ of vertices that are on the hull in counterclockwise order

(1)                              $v_0 =$ lowest vertex in $V$

(2)                              sort $V$ by polar angle in ccw order in relation to $v_0$

(3)                              $H =$ empty stack

(4)                              $H$.push($V[0]$)

(5)                              $H$.push($V[1]$)

(6)                              $H$.push($V[2]$)

(7)                              for $i = 3$ to $n - 1$

(8)                              while Turn($H$.nextToTop, $H$.top, $V[i]$) $< 1$

(9)                              $H$.pop()

(10)                              $H$.push($V[i]$)

(11)                              return $H$

*Figure 2: Pseudo Code for Graham Scan Algorithm*

**JarvisMarch(V)**
Description: Finds the convex hull from a set of vertices in a Euclidean plane
Input: Array $V$ of vertices of size $n$
Output: Ordered array $H$ of vertices that are on the hull in counterclockwise order

(1)                                  $h =$ lowest vertex in $V$

(2)                                  $i = 0$

(3)                                  do while $h \neq H[0]$

(4)                                  $H[i] = h$

(5)                                  $v = V[0]$

(6)                                  for $j = 1$ to $n - 1$

(7)                                  if $v == h$ or Turn($H[i]$,$v$,$V[j]$) $< 0$

(8)                                  $v = V[j]$

(9)                                  $++i$

(10)                                $h = v$

(11)                              return $H$

*Figure 3: Pseudo Code for Jarvis March Algorithm*

**Turn(a,b,c)**
Description: Finds the angle of 3 vertices in a Euclidean plane
Input: Vertices $a$, $b$, $c$
Output: 1 if the angle $\angle abc$ turns left, 0 if the angle $\angle abc$ forms a straight line, and -1 if the angle $\angle abc$ turns right

*Figure 4: Description for Turn Helper Function*

## Testing the Implementations

For testing the correctness of my implementations, I implemented a verification function which takes in the solved graph vertices and edges. This function requires that the edges be sorted in counter-clockwise order and that each edge also have its vertices in counter-clockwise order, which fortunately is the output of both algorithms in implementation. It begins by checking that the edges form a closed polygon, with each edge checking that it shares a vertex with its neighbors. The next check again uses each edge to see that all vertices are on the left of the edge formed by the vertices in counter clockwise order.

## Generating Inputs for Testing

When beginning this project, the minimum testing constraints prescribed testing the best, worst, and various in-between cases for each algorithm across three exponential input sizes. It is worth noting that my best and worst cases are in reference to the Jarvis March algorithm. This is because Jarvis March is output sensitive. Graham Scan will run the same on inputs of the same size with different numbers of vertices on the hull, and thus its best and worst cases are independent of the number of hull vertices.

To generate my best case, I implemented a function to generate a triangle formed from 3 vertices, with all vertices of the input size lying within the triangle. Thus, the best solvable case is that our hull is of minimum size. To generate my worst case, I implemented a function to place all the vertices along the circumference of a circle. In order to create sets of vertices where I could control the portion of vertices on the hull, I implemented a function that would place a set percentage of the vertices on the circumference of a circle and the rest within that circle.

## Testing with Inputs

For my first set of tests, I decided upon input sizes of $10^3$, $10^4$, and $10^5$ (one-thousand, ten-thousand, one-hundred-thousand). For each of these input sizes, I would record the runtime to solution for a minimum hull, a 0% hull (essentially a random hull size), a 25% hull, a 50% hull, a 75% hull, a maximum hull. For each of these cases, I would generate and solve for 5 iterations, handing the same generated set of vertices to both the Graham and Jarvis algorithms, averaging the runtime for each. This first data set yielded somewhat surprising results. As expected, the Jarvis implementation beat the Graham implementation on the minimum hull soundly across all input sizes. At its slowest, Jarvis was five times as fast. On the random input (0%), this edge for Jarvis continues, but with diminishing returns. Once we reach the 25% hull case, we see a dramatic decrease in relative efficiency for Jarvis, which worsens as the size of input increases. For each increase in percentage, we see a commensurate increase, which increases even more radically with growth of input size.

As I approached the larger hull percentages for input size $10^5$, the runtime was taking too long, and the comparison was blow out of proportion. There is a radical increase in run time once 25% hull was reach across all input sizes. In order to better understand the transition to this extreme increase, I reran my tests, this time changing the percentages to increments of 5% instead of 25%. The data from my Graham algorithm for input sized $10^3$ seemed to fluctuate across the hull percentages, but you can see a general rate of growth for Jarvis. Between 0% and 5%, the Jarvis begins to take more time. On input sized $10^4$, this transition is more

pronounced, repeating the same dramatic increase as the previous test suite. On input sized $10^5$, the rate of growth is even more dramatic, but generally following the same pattern.

Again, I wanted to find the point of transition for each input size, so I reran my tests again changing the hull percentages, this time to 1% increments. For input sized $10^3$, again my data seemed more random than representative. At input sized $10^4$, the transition occurs right around 1% before again growing rapidly. At inputs sized $10^5$, the change occurs way before this.

## Interpreting Test Results

It is clear to see the complexity of the algorithms affecting runtime. As the percentage of points on the hull increases, the runtime for Jarvis increases as well, mirroring its O(nh) time complexity. The Graham algorithm behaved somewhat more peculiarly. As expected, at the lowest input size, there was significant variance in runtime. At input size $10^4$ and across all test suites, there was a trend of decrease in runtime as the portion of vertices on the hull increased. However, at the largest input size, this trend doesn't hold. As expected, for input size $10^5$, the runtimes are roughly close enough to have differences chalked up to variance in CPU usage.

## Limitations

I was hampered by the limitations in efficiency of the Python GUI display used to output a visual graph. It was unable to output sets of data larger than 10,000 vertices, because any larger data set would take longer than reasonable to interpret and display. In order to get noticeable changes in runtime for my algorithms, I had to use datasets exceeding 10,000 vertices. There is a disconnect between the datasets used to analyze the algorithms and the visual data supplied as figures in this paper. Thus, these figures have been supplied to provide example datasets, greatly downsized from those used in the analysis, illustrating the essential form they would take. Another limitation was my use of dynamic memory to allocate the graphs and the points. At around 100 million vertices, the program would terminate due to lack of space on the heap.

In addition, there were various changes made through the study. Primarily, I had initially intended to include Chan's algorithm with O(n log h) complexity. However, the implementation of this algorithm had to be abandoned for the sake of time. The half-finished implementation for this algorithm is included as part of the repository, though it may be completed at a future date.

## Conclusion

My implementation for the Graham and Jarvis algorithms and subsequent testing were inadequate to confirming the complexity of the two algorithms. The data gathered was too broad in scope and too small in size, and thus failed to establish any mathematical metric to make comparative analysis. In a potential revisit or replication to this study, I would make changes. I would increase my iteration size to a large enough sample size to better establish trends in the rate of growth for Jarvis and perhaps normalize the variance found in Graham. In addition, I would also like to finish the implementation of Chan's algorithm.
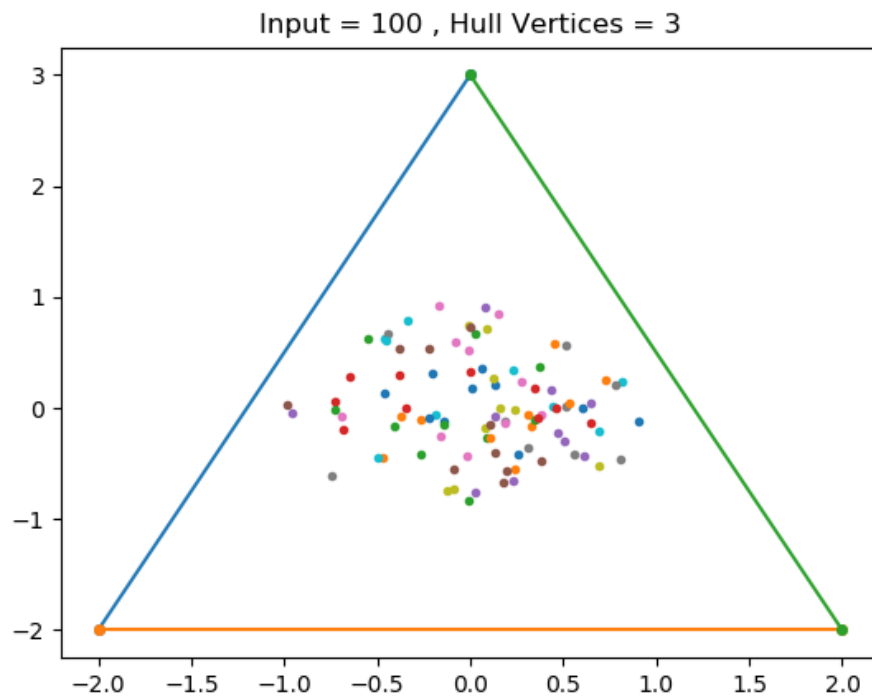
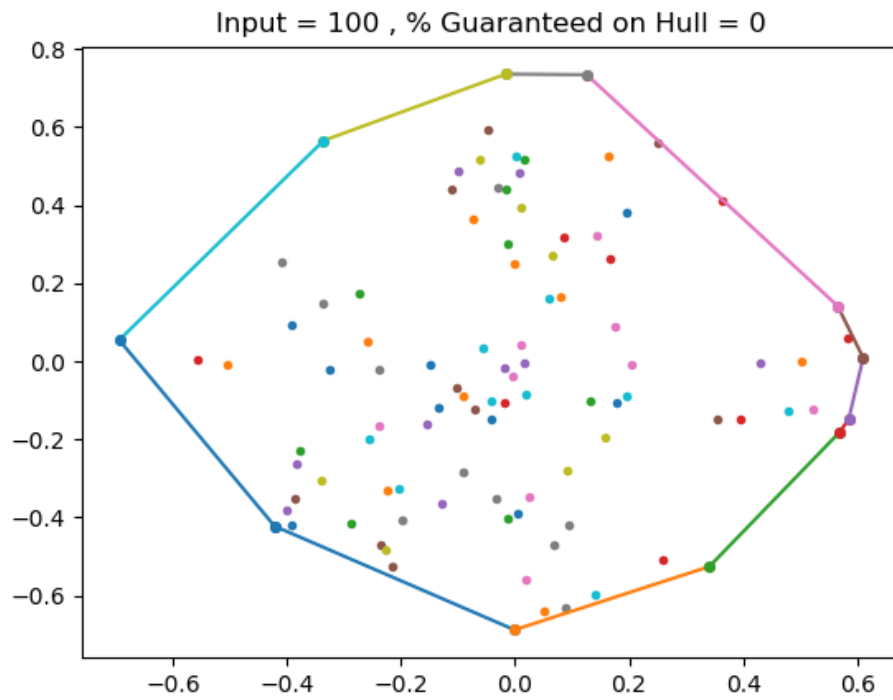*Figure 5: Convex Hull where the number of vertices on the hull is 3*



*Figure 6: Convex Hull where 0% of the vertices are guaranteed to be on the hull (effectively random input)*
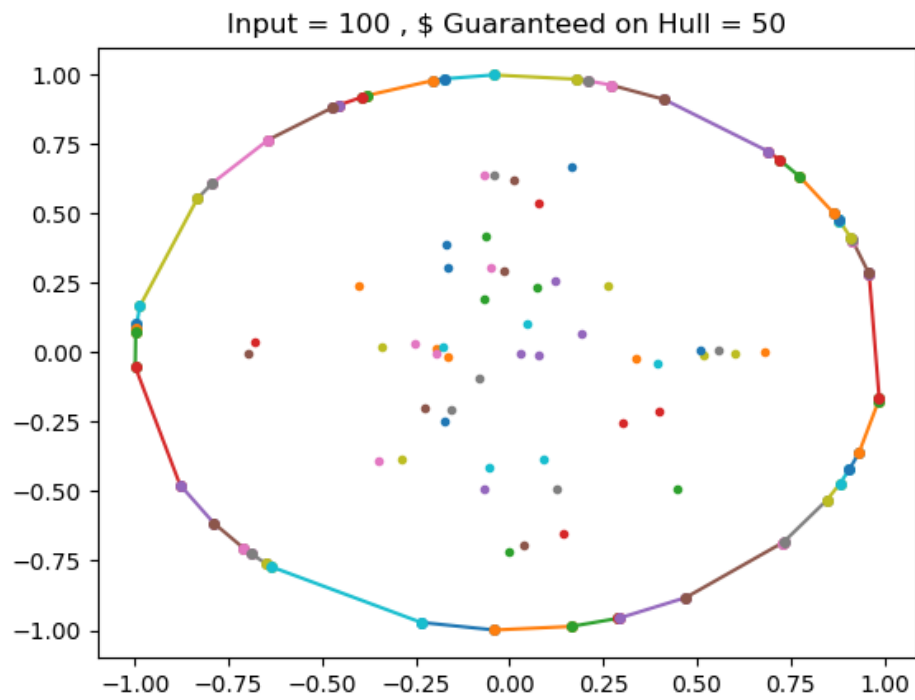
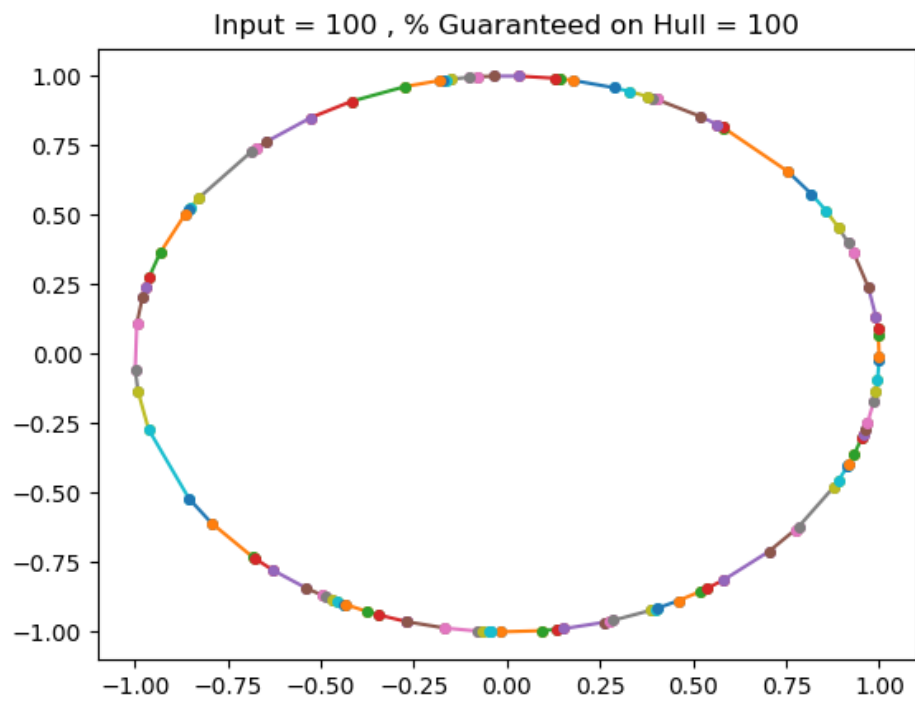*Figure 7: Convex Hull where 50% of the vertices are on the hull*


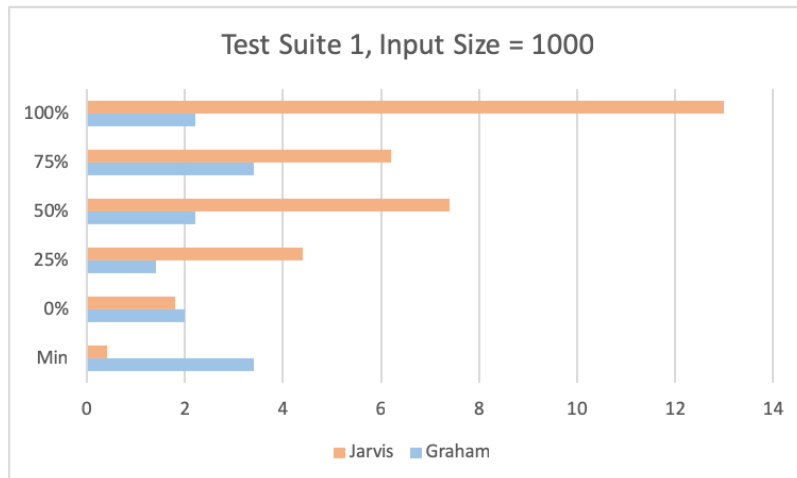
*Figure 8: Convex Hull where all vertices are on the hull*
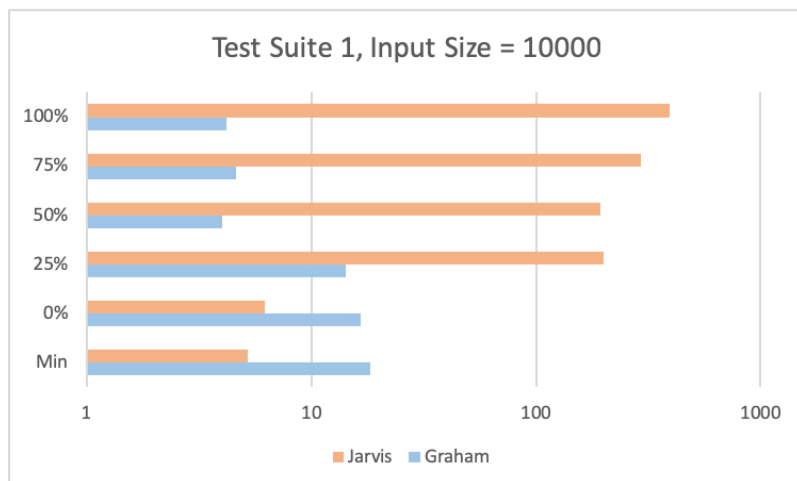
*Figure 9: Data from Test 1, Input Size 1000*


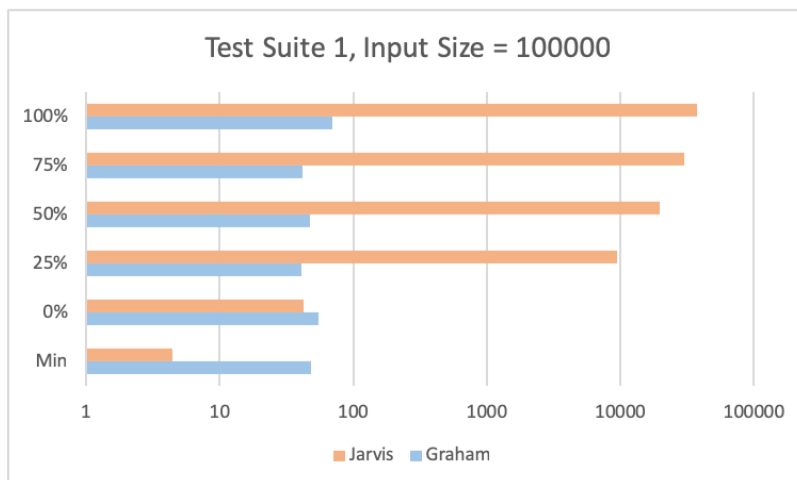*Figure 10: Data from Test 1, Input Size 10000*
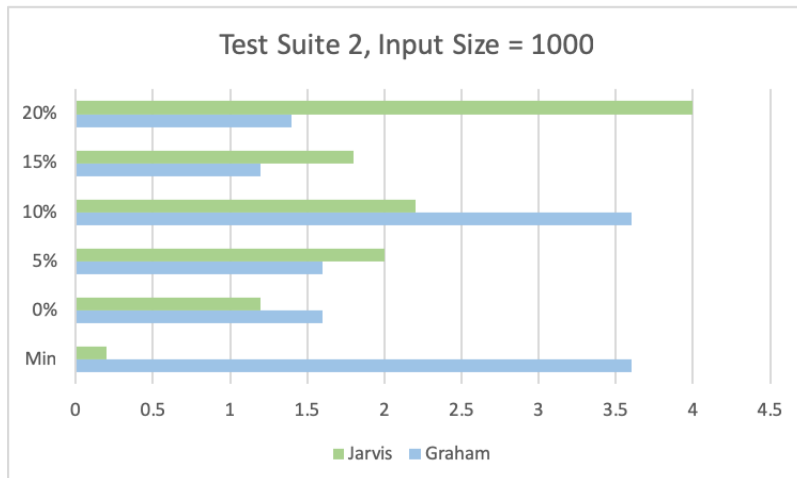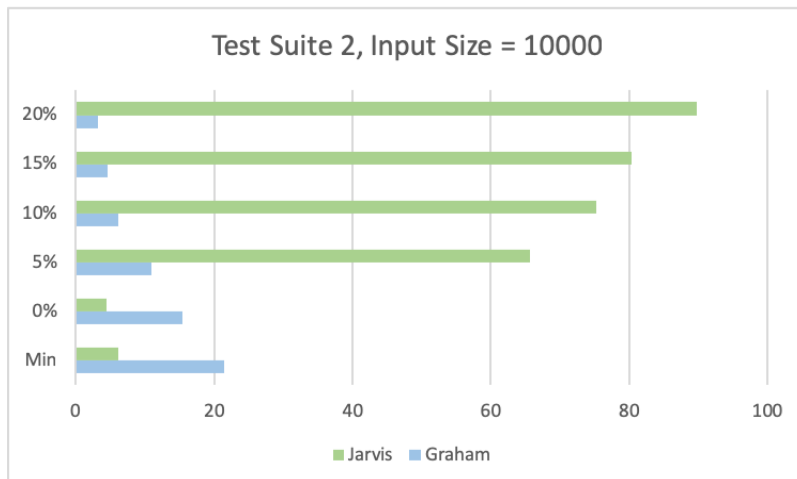

*Figure 11: Data from Test 1, Input Size 100000*

*Figure 12: Data from Test 2, Input Size 1000*
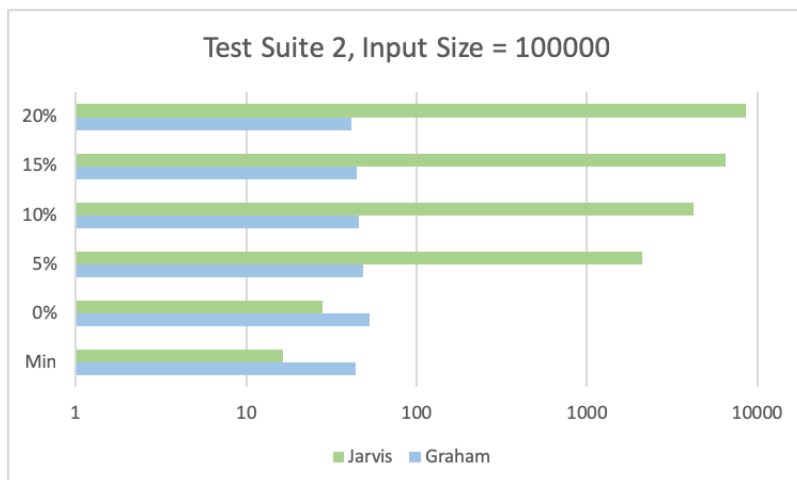


*Figure 13: Data from Test 2, Input Size 10000*



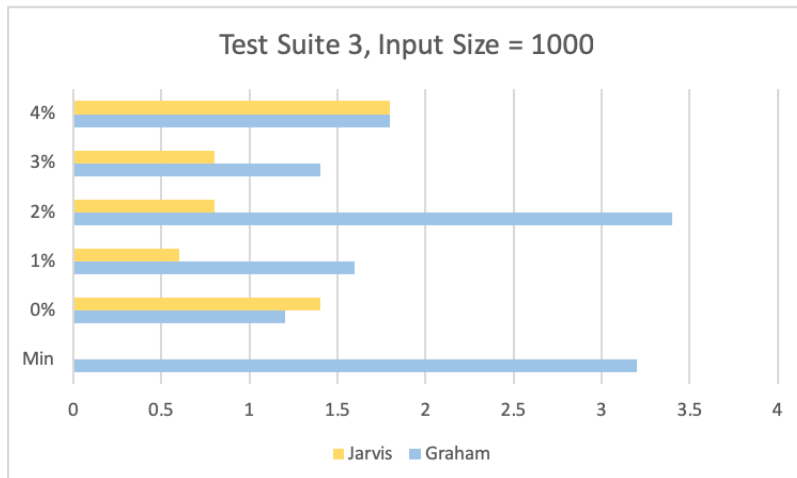*Figure 14: Data from Test 2, Input Size 100000*

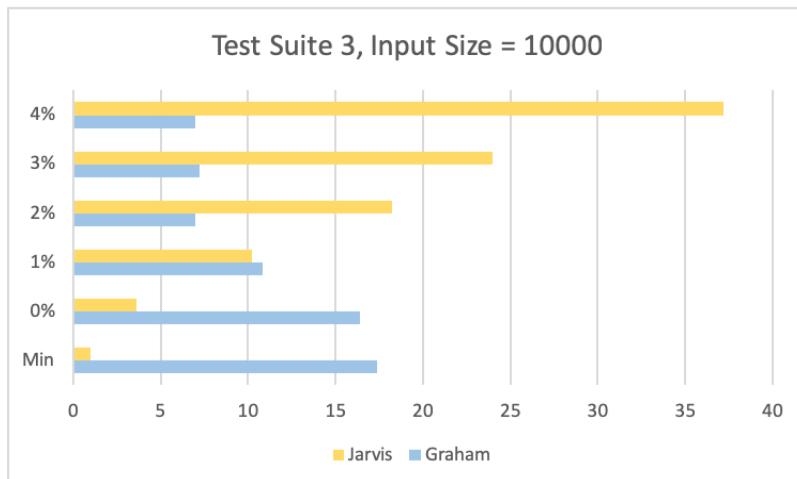*Figure 15: Data from Test 3, Input Size 1000*



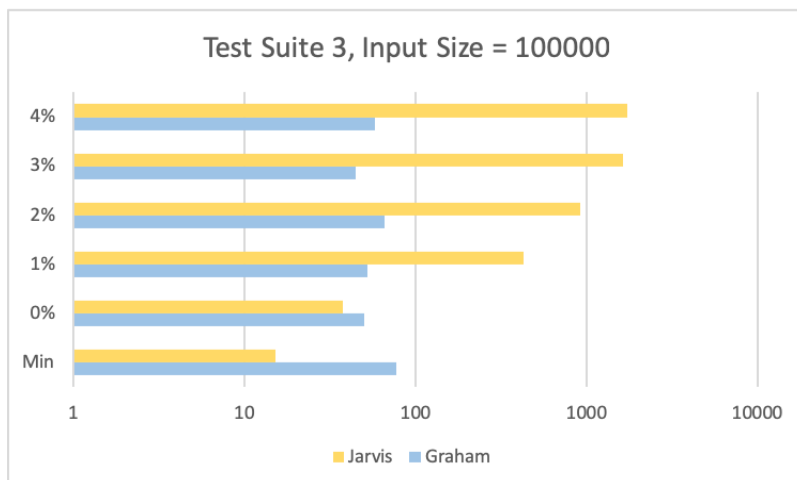*Figure 16: Data from Test 3, Input Size 10000*



*Figure 17: Data from Test 3, Input Size 100000*

Citations

"33.3 Finding the Convex Hull." *Introduction to Algorithms*, by Thomas H. et al Cormen, MIT Press, 2009, pp. 1029–1038.

Chan, T. M. "Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions." *Discrete & Computational Geometry*, vol. 16, no. 4, 1996, pp. 361–368., doi:10.1007/bf02712873.

Day, A. M. "Planar Convex Hull Algorithms in Theory and Practice." *Computer Graphics Forum*, vol. 7, no. 3, 1988, pp. 177–193., doi:10.1111/j.1467-8659.1988.tb00608.x.

"Gift Wrapping Algorithm." *Wikipedia*, Wikimedia Foundation, 29 Nov. 2018, en.wikipedia.org/wiki/Gift_wrapping_algorithm.

Graham, R.l. "An Efficient Algorith for Determining the Convex Hull of a Finite Planar Set." *Information Processing Letters*, vol. 1, no. 4, 1972, pp. 132–133., doi:10.1016/0020-0190(72)90045-2.

Jarvis, R.a. "On the Identification of the Convex Hull of a Finite Set of Points in the Plane." *Information Processing Letters*, vol. 2, no. 1, 1973, pp. 18–21., doi:10.1016/0020-0190(73)90020-3.

Kiers, Bart. "Bkiers/GrahamScan." *GitHub*, 2 Aug. 2018, github.com/bkiers/GrahamScan.

Mcqueen, Mary M, and Godfried T Toussaint. "On the Ultimate Convex Hull Algorithm in Practice." *Pattern Recognition Letters*, vol. 3, no. 1, 1985, pp. 29–34., doi:10.1016/0167-8655(85)90039-x.

Yao, Andrew Chi-Chih. "A Lower Bound to Finding Convex Hulls." *Journal of the ACM*, vol. 28, no. 4, 1981, pp. 780–787., doi:10.1145/322276.322289.