Homework 3: Convex Hull

Fabian Andres Prada Nino

1) Algorithm's Implementation.

- Jarvis. It was implemented using three methods: buscarInferiorDerecho(), buscarSucesor(), and jarvis(). buscarInferiorDerecho() runs an O(n) search to identify the point with lowest y coordinate (and largest x as a second comparision condition), and this becomes the initial vertex of the convex hull, v_0 . Once a vertex v_k is added to the convex hull, the method buscarSucesor() is applied over v_k to identify the next vertex v_{k+1} of the convex hull in positive orientaton. For each input point p, buscarSucesor() defines vectors centered at v_k and directed to p, and sets as v_{k+1} the point whose vector is leftmost. jarvis() invokes buscarInferiorDerecho() for a single time and then invokes buscarSucesor() until $v_{k+1} = v_0$.
- Graham V1. The implementation of this algorithm was much more complex than Jarvis. It is divided in four major methods: buscarInferiorDerecho(), ordenarVertices(), recorrerVerticesOrdenados(), and grahamVariante1(). The method ordenarVertices() defines vectors centered at the lowest-rightmost point and directed to any other input point. Then, the additional method quicksortVectores() is invoked to sort these vectors, doing this task in O(nlogn) time (taking in to account that quicksort had the best results in randomly generated samples, I prefer this method instead of heapsort and mergesort). recorrerVerticesOrdenados() visits the input points in the order induced by the previous vector sorting. Each time a new point is visited, the current convex path is updated. To do this, I created an array variable, int[]arraySucesion, that stores the points added to the convex hull in the order they are visited, and the variable int ultimaPosArray that indicates where the new point visted must be stored in the array. The method grahamVariante1() invokes the other three major methods in the described order.
- Graham V2. The implementation of this algorithm was harder than Jarvis and a bit easier than Graham V1. The algorithm is divided in three major methods: *quicksortMenosInf(),recorrerMenosInfMasInf()*,and *grahamVariante2()*. *quicksortMenosInf()* sorts the points in positive order according to a point in minus infinity position. To do this, the first criterion of comparision is 'larger x coordinate is lower', and the second criterion is 'lower y coordinate is lower'. The sorting from a point in plus infinity position is precisely the reverse of the previous result, so it's not neccesary to run a new sort. *recorrerMenosInfMasInf()* first visits the list of point sorted by minus infinity position to find the upper path of the convex hull, and inmediately

visits the list of points sorted by plus infinity position to find the lower path of the convex hull.

2)Results in test sets.



Figure 1: These images shows the results of the algorithms applied to the given test sets. The number of points in the convex hull of (a) is 13, and in the convex hull of (b) is 8.



Figure 2: These images shows the results of the algorithms applied to samples of 1000 points in a rectangle(a), a triangle(b), interior of a circle(c), and border of a circle(d). The amount of points in the convex hull for these figures were 11, 13, 28, and 1000 respectively.

3) Comparision of Performance.

I took data sets of 100,1000,10000,20000,40000,80000, and 1600000 points uniformly distributed in rectangles, triangles, circles and border of circles. For each geometric shape, size of the data set, and algorithm, the experiment was run 15 times, and the mean was taken over the last 10 results:



- The results corroborates that the performance of **Jarvis** depends a lot on the amount of points in the convex hull (O(nh)), as seen in class). We can observe that **Jarvis** is better than **Graham V1** and **Graham V2** for triangles and rectangles, and this is because of the small output for such cases (see Figure 2). In the case of points in the interior of circles, the performance of **Jarvis** get worse (compared with **Graham**) as the amount of points in the sample (and consequently in the output) increases. From the previous result, one could think that the amount of expected points in the convex hull in the case of triangle or a rectangle is $o(\log n)$ while in the case of a circle is $\Omega(\log n)$. Finally, in the case of the border of a circle, **Jarvis** performs $O(n^2)$ as expected, since all the points belong to the hull.
- Graham V1 and Graham V2 were almost indifferent to the particularities of the figures, i.e., input distributions and output size, and performed very similarly in the four cases. This was expected since these algorithms are almost indifferent to the geometry of the figure: all the points are sorted to define the order they will be visited, and then they are all visited once (in the case of Graham V1) or twice (in the case of Graham V2). From the obtained resuts it is not possible to say in which figure they performed best. However I expected that the best performance took place for border of circles: once the sorting is done, the round trip is the easiest possible since no exclusions are performed. Probably, this was not clearly evidenced in the results by two reasons: the amount of experiments were insufficients, or the sort step (which is

 $O(n \log n)$ and is indifferent to geometry) is so weighted that it hidden the round trips (which is O(n) and is sensible to geometry).

• Graham V1 and Graham V2 performed almost identically in all the experiments. I expected that they had a similar behaviour, but not so close!!. I expected that the differences in the criterions used for sort the points and the differences in round trips would lead to notable differences in performance. Again, I guess that the weigth of the sort step (which in both cases is quicksort, $O(n \log n)$) hiddens steps like the round trips (O(n)).(see section (5) for extended discussion).

4)Elimination of Interior Points

To eliminate interior points I implemented an algorithm that eliminates points in the interior of a quadriletaral. The vertex of the quadrilateral are the leftmost, rightmost, utmost and lowest points of the sample. The following table compares the results (in seconds) of applying elimination in the previous figures for samples of 80000 and 160000 points.

	JAR VIS		GRAHAN	/V1	GRAHAMV2	
	Sin Eliminacion	Con Eliminacion	Sin Eliminacion	Con Eliminacion	Sin Eliminacion	Con Eliminacion
Rectangulo						
80000	0,015521	0,01164	0,056634	0,02611	0,052461	0,02505
160000	0,03432	0,02023	0,110684	0,05568	0,0972243	0,047006
Triangulo						
80000	0,0165	0,006846	0,0501	0,01206	0,0567	0,01105
160000	0,02912	0,014005	0,11089	0,02254	0,12664	0,022429
Int Circulo						
80000	0,1049	0,04894	0,051183	0,02115	0,046777	0,01931
160000	0,3884	0,143107	0,094644	0,047733	0,0968312	0,03976
Borde Circulo						
80000	69,5268	69,7476	0,042954	0,05368	0,047664	0,05995
160000	289,2461	289,6923	0,101127	0,105358	0,104407	0,119034

- In the case of **Jarvis**, the improvement was about 33% in the rectangle, 50% in the triangle and 50% in the interior of the circle. Since eliminating interior points is done in O(n) time and do not affect the output size (i.e *h* is not changed) the percentage of improvement for **Jarvis** is closely related with the percentage of points removed. Looking at Figure 3 we observe that 'quadrialteral elimination' removes a higher percentage of points in triangles, then in circles and finally in rectangles.
- In the case of **Graham V1** and **Graham V2**, the improvement was about 50% in the rectangle, 80% in the triangle, and 60% in the interior of the of the circle. Since Graham is $O(n \log n)$ (and independent of the output size h), it is coherent to have better improvements for Graham than Jarvis as effectively we get. In this case, the relation between the percentage of points removed and the percentage of time improvement is even closer (see Figure 3).



Figure 3: These images shows the results of quadrilateral elimination in samples of 1000 points inside a rectangle(a), a triangle(b) and a circle(c). The percentage of removed points was 52%, 92%, and 65% respectively.

• As expected, applying elimination in the case of circle border does not produce any improvement (since there is nothing to eliminate!), and increases a bit the total time of performance of the algorithms.

5)Impact	of	sort	algorithms	in	the	alobal	behaviour
~ .	/=	~ J					3	

	GrahamV1Quicksort	GrahamV2Quicks ort	GrahamV1Insercao	Graham V2 Insercao
Rectangulo				
20000	0,0087725	0,0083071	1,26194	1,21702
40000	0,019215	0,0183965	5,36712	5,01791
80000	0,039986	0,045401	22,3671	21,3358
Triangulo				
20000	0,007723	0,008021	1,2418	1,1933
40000	0,01932	0,02144	5,3527	5,1896
80000	0,04282	0,04604	22,7256	20,9969
Int Circulo				
20000	0,008761	0,008923	1,2644	1,2112
40000	0,01852	0,02076	5,3801	5,01184
80000	0,03999	0,04663	22,538	21,5098
Borde Circulo				
20000	0,007277	0,008294	1,2666	1,19284
40000	0,0159595	0,0163259	5,4359	5,02678
80000	0,035606	0,038809	22,8645	21,3593

• The sort step in **Graham's** algorithm almost define the global performance of the algorithm, I mean, most of implementation time is concerned with the sort step and it effectively eclipses the reamining steps. Therefore, when Quicksort was applied the

global performance of **Graham** (both V1 and V2) was $O(n \log n)$ and when Insertion was applied the global performance was $O(n^2)$.

- In general, terms **Graham V1** and **Graham V2** performs almost identically when Quicksort is applied and they also performs almost identically when Insertion is applied. Looking the table at detail, for the Quicksort case Graham V1 seems to be a bit better than Graham V2, and in the Insertion case the opposite happens. In order to justify this phenomenon take into account the following observations:
 - In the sort step, **Graham V2** only requires of at most two number's comparisions $(p_x \text{ with } q_x, \text{ and in case of equality } p_y \text{ with } q_y)$, while in **Graham V1** it requires finding the lowest-rightmost point (O(n)), assigning a vector to each point of the sample (O(n)), and finally calculating a determinant to compare pairs of vectors(a higher multiplicative constant for the $n \log n$ associated to Quicksort). In conclusion, the sort step in **Graham V1** is more expensive than in **Graham V1**.
 - The round trip in Graham V1 only visits each point once, while the round trip in Graham V2 visits points twice, once for the minus infinity ordering and once for the plus infinity ordering. Therefore, Graham V2 is more expensive than Graham V1 in the round trip step.

When Quicksort is applied for the sort step it does not eclipses totally the round trip step, this glabally produces that **Graham V1** performs a bit better than **Graham V2**. On the other hand, when Insertion is applied it eclipses totally the round trip step, and this globally produces that **Graham V2** performs better than **Graham V1**.

• The previous results also corroborates that **Graham** algorithm (I mean the round trip specifically) is indifferent to the geometric features of the sample. Consequently, this algorithm performs almost identically in all geometric shapes.