Convex Hull

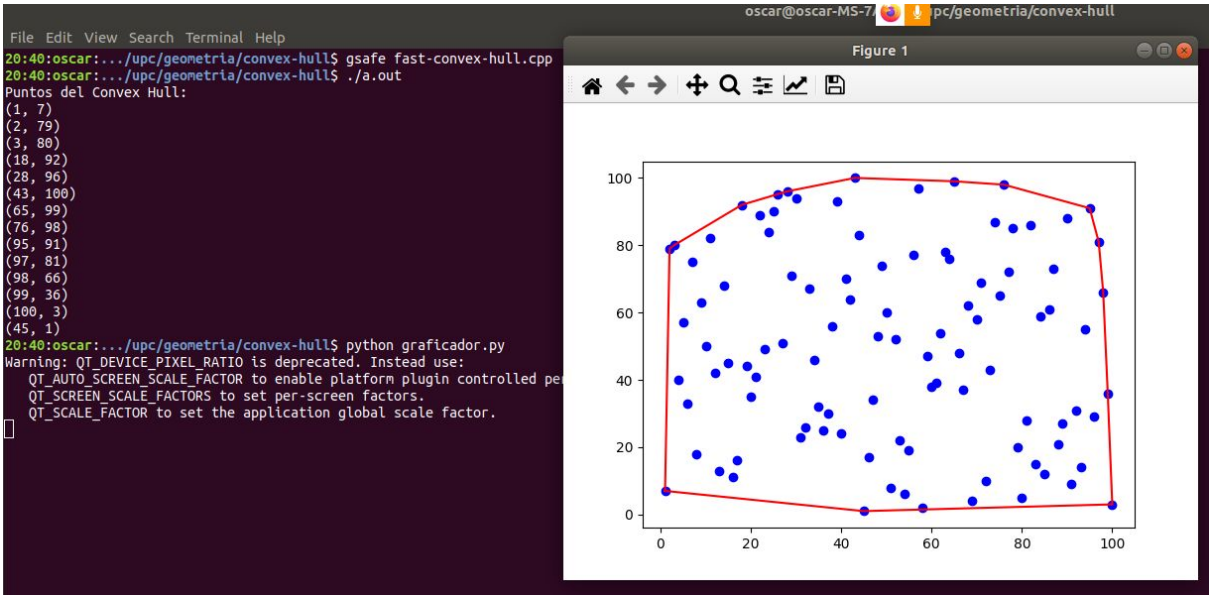
Ejercicio 1: Implementación del algoritmo
Partes principales del código:

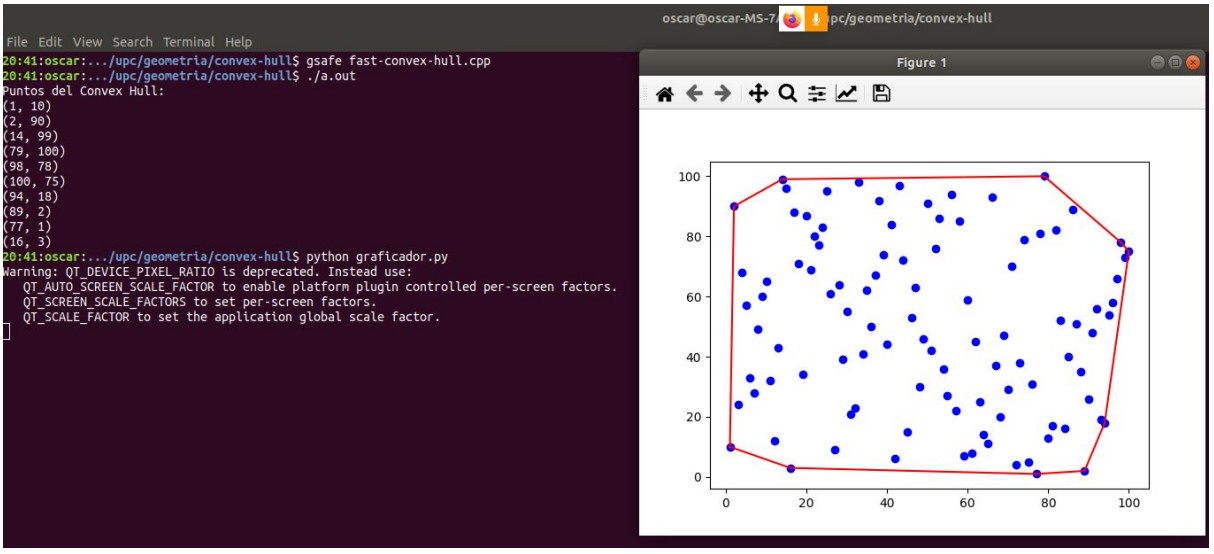
```
//Funcion para obtener el vector pq dados 2 puntos p y q
pair<int,int> getvec(pair<int,int> p, pair<int,int> q){
        return { q.first - p.first, q.second - p.second };
}

//Producto cruz de vectores
int cross(pair<int,int> u, pair<int,int> v){
        return (u.first * v.second) - (u.second * v.first);
}

//Comparador para determinar si tres puntos giran a la izquierda
bool uppercmp(pair<int,int> l, pair<int,int> mid, pair<int,int> r){
        pair<int,int> u = getvec(mid, l), v = getvec(mid, r);
        return cross(u, v) <= 0;
}
//Comparador para determinar si tres puntos giran a la derecha
bool lowercmp(pair<int,int> l, pair<int,int> mid, pair<int,int> r){
        pair<int,int> u = getvec(mid, l), v = getvec(mid, r);
        return cross(u, v) >= 0;
}
```

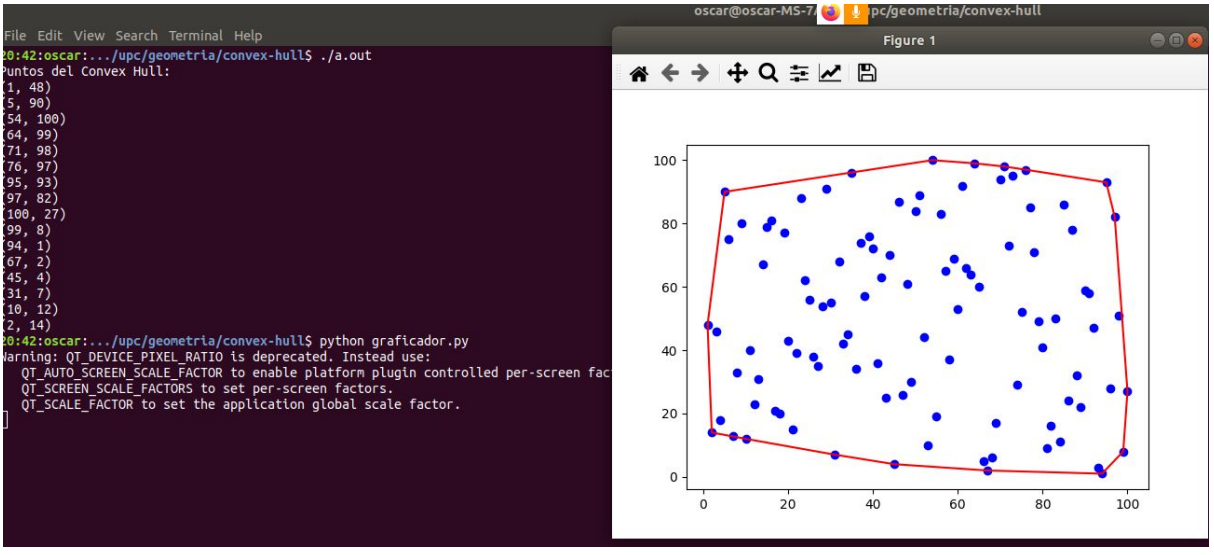
```
//Funcion principal, practicamente identico al algoritmo presentado en clase
//La única variación que hice es evaluar el L_lower en la misma pasada con un comparador distinto
//Me pareció más sencillo de esta forma
vector<pair<int,int>> fast_chull(){
        sort(pts.begin(), pts.end());
        vector<pair<int,int>> upper = { pts[0] }; //L_upper
        vector<pair<int,int>> lower = { pts[0] }; //L_lower
        int lu = 1, lo = 1; //lu y lo son el tamaño actual del L_upper y L_lower respectivamente
        for(int i = 1; i<n; i++){
                //Mientras tengo por lo menos 2 puntos en mi hull superior y tengo un giro a la izq
                while (lu >= 2 && uppercmp(upper[lu-2], upper[lu-1], pts[i])){
                        upper.pop_back(); //elimino el punto del medio
                        lu--;
                }
                upper.emplace_back(pts[i]);
                lu++;
                //Mientras tengo por lo menos 2 puntos en mi hull inferior y tengo un giro a la der
                while (lo >= 2 && lowercmp(lower[lo-2], lower[lo-1], pts[i])){
                        lower.pop_back(); //elimino el punto del medio
                        lo--;
                }
                lower.emplace_back(pts[i]);
                lo++;
        }
        reverse(lower.begin(), lower.end()); //revierto el orden del L_lower
        upper.pop_back(); //Ultimo punto de upper es el primer punto de lower
        upper.insert(upper.end(), lower.begin(), lower.end()); //meto todo el L_lower a L_upper
        upper.pop_back(); //Ultimo punto de lower es el primer punto de upper
        return upper; //retorno el hull completo ahora almacenado en L_upper
}
```

El código completo se encuentra adjunto en el zip.

Ejercicio 2: Capturas de pantalla de los resultados obtenidos (casos aleatorios con 100 puntos).







Ejercicio 3: Análisis de complejidad del algoritmo.

Denotemos el tamaño del conjunto de puntos de los que se desea obtener su Convex Hull como ‘N’.

Consideremos los pasos principales del algoritmo:

a) Ordenar el conjunto de puntos por coordenada X ascendente, y romper empates ordenando por coordenada Y ascendente. Se sabe que ordenar un conjunto de elementos puede realizarse en O(NlogN) usando algún algoritmo óptimo de ordenamiento, como Merge Sort. Por lo tanto, la complejidad de este primer paso es **O(NlogN)**.

b) Construir la parte superior del Convex Hull. Este proceso puede simularse con una pila, en la que agregamos puntos a la pila mientras la pila contenga menos de 2 elementos o mientras se formen giros hacia la derecha con el punto que se esta evaluando. Como cada punto se evalúa sólo una vez, y es añadido y/o eliminado del L_upper como máximo 1 vez, y hay N puntos, la complejidad de este paso es **O(N)**.

c) Construir la parte inferior del Convex Hull. Por un análisis completamente análogo al realizado en el punto **(b)**, concluimos que la complejidad de este paso es **O(N)**.

d) Mezclar ambas partes del Convex Hull. Este paso puede realizarse en O(|Lu| + |Lo|), donde |Lu| y |Lo| son los tamaños de la parte superior e inferior del casco convexo respectivamente. Como |Lu| y |Lo| están acotados superiormente por la cantidad de puntos del conjunto, este paso resulta ser **O(N)**.

En conclusión, los pasos (b), (c) y (d) tienen todos complejidad lineal O(N). El único paso con complejidad superior es el paso (a), el cual tiene complejidad O(NlogN). Como este término es de un orden mayor que O(N), la complejidad total resulta **O(NlogN)** por el ordenamiento. Sin embargo, si empezaramos con el conjunto de puntos ya ordenado y se pudiera omitir el paso del ordenamiento, la complejidad sería lineal.