Graham's Scan Algo

let p0 be the point in P with minimum y coordinate, or the leftmost in case of tie } minimum element O(n)

let p1, p2 ... pn be remaining points in P, sorted by polar angle in counterclockwise order around p0 (if more than one pt has same angle, remove all but the one farthest from p0 } sorting in O(n log n)

```
Create stack S
push(p0, S)
push(p1, S)        } O(1)
push(p2, S)

for i = 3 to n
    while ( the angle made by point on stack.top, stack next-to top
            and pi makes a non left turn)
        pop(S)
    Push(pi, S)

Return Stack
```

```
vector <pair<int, int>> graham( vector <pair <int, int>> g)
.{
  auto it = min_element( g.begin(), g.end(), [](const auto &a,
  const auto & b) {
      return (a.second < b.second) || (a.second = b.second
      && a.first < b.first).
  };
```

returns the ~~min~~ element of the array by looking at the y coordinate
index of min (pair.second) of vector. If they are equal, then
look at x coordinate

```
  int p0 = distance( graph.begin(), it);
```

↳ get the index from iterator

```
  vector <pair<int, int>> sorted = sort by polar angle (p0, graph);
```

net
O(n log n)

```
      pair <int, int> ini ~~p0~~ = ~~p0~~ graph [p0]
      vector <pair <int, int>> sorted ~~ans~~;
O(n)  for (int i = 0; i < points.size; i++)
       if (i != p0) sorted.push_back(points[i]);

  ~~sort (sorted.begin~~
O(n log n) ← sort (ans.begin(), ans.end(), [&] (pair <int, int> a,
    pair <int, int> b) {
  double angle A = atan2(a.second - p0.second, a.first - p0.first)
  double angle B = atan2(b.second - p0.second, b.first - p0.first)
  if ( abs(angle A - angle B) < 1e-9) → angles are very close for double or same angle
  return (a.f - p0.first)^2 ~~* a.first)~~
      * (a.second - p0.second)^2
      < (b.first - p0.first)^2 * (b.second - p0.second)^2

return angle A < angle B → return condition
```

To determine which is more outward point

```
stack <pair < int, int >> st

st.push ( sorted [ 0 to 2 ]  → O(1)

pair <int, int > top, next

for (int i = 3 ; i < sorted.~~sorted~~ size () ; i++)
  { while (st.size () > 1
        top = st.top () ;
        st.pop () ;
        next = st.top () ;
        if ( ! non left turn (next, top, sorted [i] )
           { st.push (top)
             break ;  → we have confirmed that top of
           }             stack makes a left (convex) turn
                         ∴ break and look for next sorted
        st.push (sorted [i])   pt
  }

vector <pair <int, int > ans ;
   while ( ! st.empty () )
     ans.push_back (st.top ());
     st.pop () ;
   return reverse ( ans.begin (), ans.end () ) ;  → depends on
                                                    number of points
                                                    in solution
                                                    worst case O(n)
```

Thus net time & complexity is O(n log n)

no Recurrence.