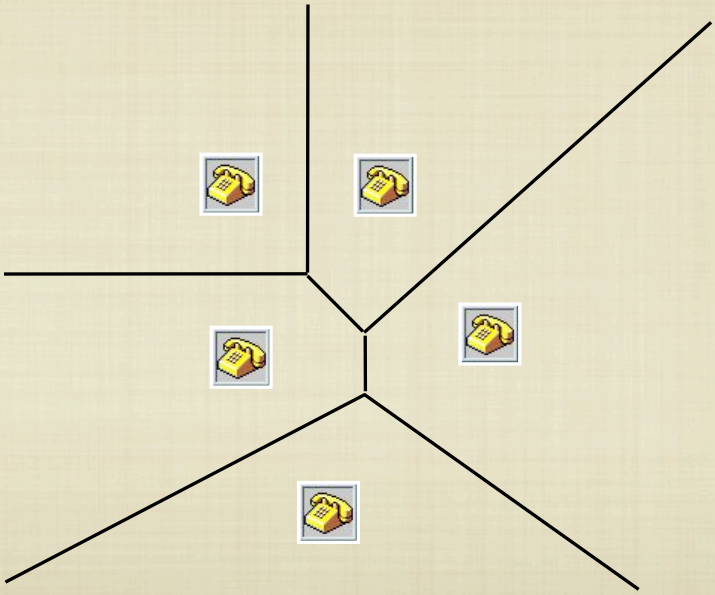
CMP189
Algoritmos Geométricos

Prof. João Comba

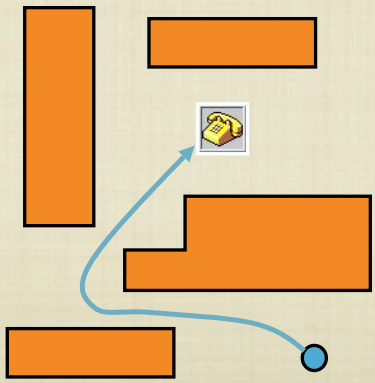
Proximidade

Diagrama de Voronoi (1907)

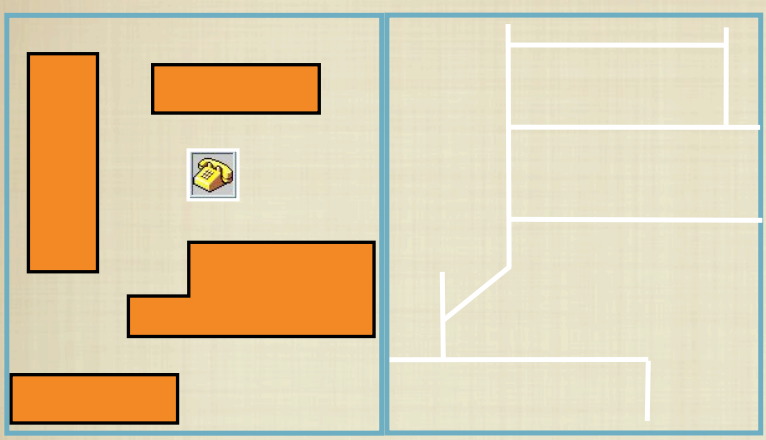


Planejamento de movimento

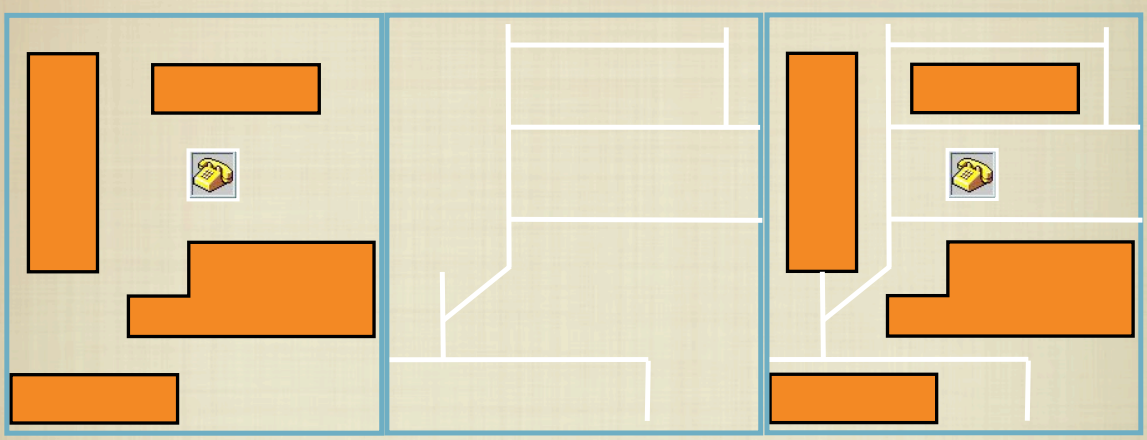
- Encontrar o caminho mais curto sem colidir com paredes e objetos



Sobreposição de Mapas



Sobreposição de Mapas



Algoritmos Geométricos

- **Geometria Computacional:**
  - Início na década de 70
  - Problemas geométricos
  - Meta: Buscar algoritmos e estruturas de dados com bom desempenho quando os problemas escalam de tamanho
  - Aplicações: Computação Gráfica, Processamento de Imagens, Robótica, Sistemas Geográficos de Informação, VLSI, CAD, etc.

Algoritmos Geométricos

- Problemas Típicos:
  - Entrada: Objetos Geométricos (pontos, linhas, planos, etc)
  - Saída: Resultados de consulta sobre os objetos, ou um novo objeto geométrico
- Ingredientes:
  - Geometria, Topologia, Algoritmos, Estruturas de dados, Análise de algoritmos, Problemas numéricos

Algoritmos Geométricos

- Limitações Comuns:
  - Natureza discreta: aproximações discretas de fenômenos contínuos
  - Objetos planares ou lineares (aproximações de objetos curvos)
    - Permite concentrar em aspectos combinatórios dos problemas em detrimento de aspectos numéricos
  - Problemas de baixa dimensão
    - Primordialmente 2D (fácil de visualizar) e 3D (até certos limites)

Histórico da área

- **Década de 80:**
  - Desenvolvimento de técnicas eficientes de projeto e análise de algoritmos
- **Década de 90:**
  - Maioria dos problemas abertos foi resolvida
  - Problema: distância muito grande entre teoria e prática (muitas das soluções eram quase impossíveis de serem implementadas ou sensíveis a casos especiais)
- **Atualmente:**
  - Simplificar algoritmos existentes, lidar com problemas numéricos, produzir bibliotecas de procedimentos geométricos

Projeto de algoritmos geométricos

1. Projeto de algoritmo básico ignorando situações especiais (casos degenerados). Ex. Pontos colineares, segmentos de linha verticais, etc). Assumir posição geral.

Projeto de algoritmos geométricos

1. Projeto de algoritmo básico ignorando situações especiais (casos degenerados). Ex. Pontos colineares, segmentos de linha verticais, etc). Assumir posição geral.
2. Lidar com casos degenerados
   - Adicionar vários casos especiais
   - Modificar o algoritmo original para integrar casos especiais

Projeto de algoritmos

1. Projeto de algoritmo básico ignorando situações especiais (casos degenerados).
   Ex. Pontos colineares, segmentos de linha verticais, etc).
   Assumir posição geral.
2. Lidar com casos degenerados
   - Adicionar vários casos especiais
   - Modificar o algoritmo original para integrar casos especiais
3. Implementação
   - Usar biblioteca de algoritmos geométricos
   - Desenvolver suas próprias classes

Aplicações

- **Computação Gráfica:**
  - Interseções entre primitivas
  - Primitiva apontada pelo mouse
  - Subconjunto de primitivas dentro de uma região
  - Remoção de superfícies ocultas em 3D
  - Cálculos de sombras em ray-tracing e radiosidade
  - Detectar colisões entre objetos

Aplicações

- **Robótica:**
  - Planejamento de movimento de robô
  - Planejamento de articulações de robo
    - Alcance do braço
    - Orientações das diferentes juntas

Aplicações

- **Sistemas Geográficos de Informação:**
  - Representações de dados geográficos:
    - Topografica, Hidrografica, Separações físicas e políticas, estradas (mapas diversos)
    - Dados são muito numerosos
  - Extrair informações eficientemente:
    - Qual a média de chuva em uma dada região
    - Como guiar um carro e definir rotas de navegação
    - Sobreposição de mapas

Aplicações

- **CAD/CAM:**
  - Placas de Circuitos, Projetos arquitetônicos
  - Entidades geométricas (problemas geométricos aparecem)
    - Interseções e uniões de objetos
    - Decomposição de objetos em formas mais simples
    - Visualização dos objetos projetados

Conteúdo

1. Fundamentos
2. Interseções de Segmentos de Linha
3. Particionamento de Polígonos
4. Envoltórias Convexas
5. Diagramas de Voronoy
6. Triangulações de Delaunay
7. Algoritmos de Busca Pontual e Intervalar
8. Quadtrees, BSP-trees, R-trees
9. Grafos de Visibilidade
10. Planejamento de Movimento

Conteúdo

- **Técnicas de Projeto de Algoritmos**
  - Lugares geométricos
  - Transformações geométricas
  - Dualidade
  - Space Sweep
  - Estruturas Hierárquicas
- **Estruturas de Dados:**
  - Cascadeamento Fracional
  - Estruturas de Dados Persistentes
  - Estruturas de Dados Cinéticas
  - Análise caso médio

Avaliação

- **Trabalhos teóricos e práticos**
- **Trabalho Final:**
  - Definido de comum acordo com o aluno / orientador
  - Alguns temas sugeridos
  - Sugestões por parte dos alunos são estimuladas
    - Preferencialmente dentro da área de pesquisa de mestrado
  - Resultados esperados:
    - Artigo técnico mostrando os resultados obtidos (em inglês)
    - Apresentação final
- **Prova Final**

Bibliografia

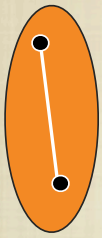
- bibliografia básica:

- Computational Geometry: Algorithms and Applications. M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. Springer Verlag, 1997.

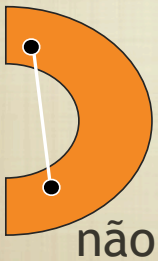
- bibliografia complementar:

- Computational Geometry in C. Joseph O'Rourke. Cambridge University Press, 1993.
- Applications of Spatial Data Structures. Hanan Samet. Academic Press, 1990.
- Ruler, Compass, and Computer: The Design and Analysis of Geometric Algorithms. Leo Guibas, Jorge Stolfi
- Computational Geometry: Course Notes. David Mount (Univ. Of Maryland)
- Introduction to Algorithms. Cormen, Leiserson, Rivest. McGraw Hill.

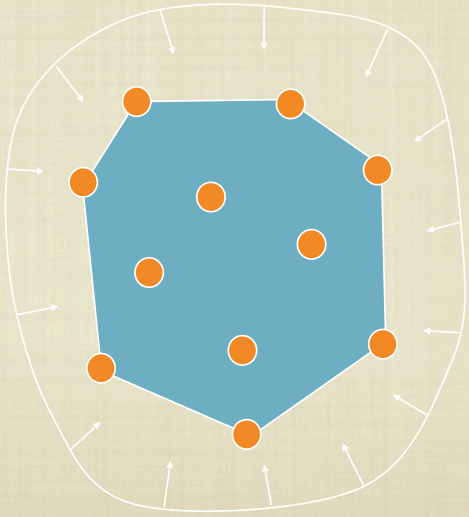
Exemplo: Envoltória Convexa



convexo



não convexo

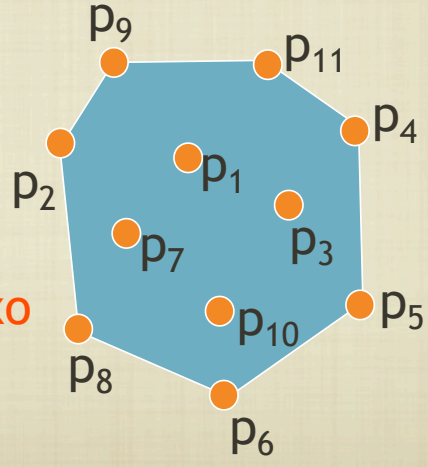


Exemplo: Envoltória Convexa

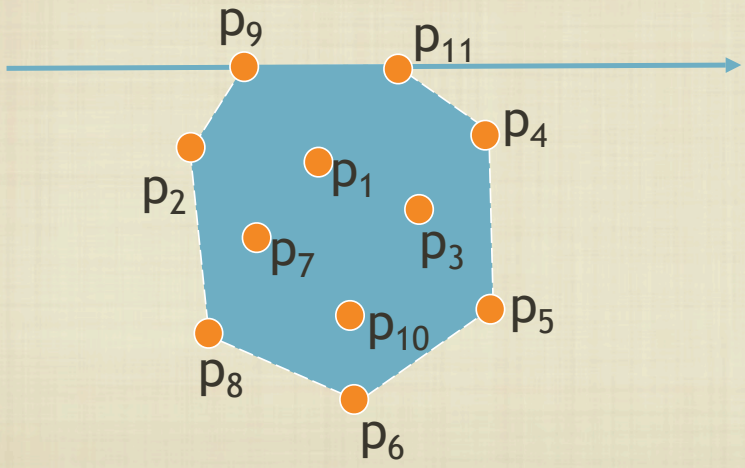
Entrada: $p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}$

Saída : $p_2, p_9, p_{11}, p_4, p_5, p_6, p_8, p_2$

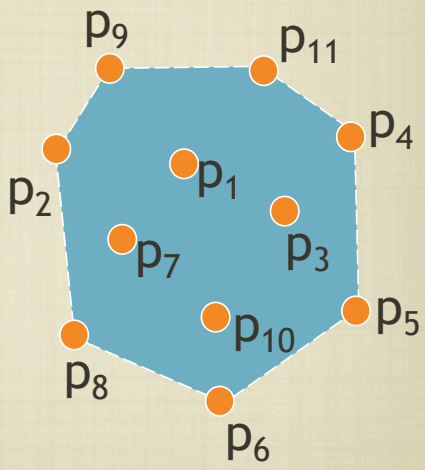
EC é um polígono convexo



Exemplo: Envoltória Convexa



Envoltória Convexa 1



Envoltória Convexa 1

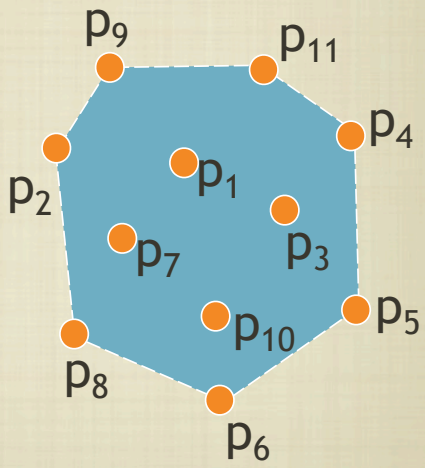
Algoritmo EC1:

Entrada: Conjunto de pontos P no plano

Saída: Uma lista de vértices da EC em ordem

1. E = ø
2. FOR todos pares (p,q)∈PxP (p!=q)
3. DO valido = true
4. FOR todos pontos r∈P(r!=p and r!=q)
5. IF r está a direita da reta pq
6. THEN valido = false
7. IF valido THEN adicione pq para E
8. Para o conjunto E de arestas, construa uma lista de vertices classificados em ordem cronológica

(n^2-n)



Envoltória Convexa 1

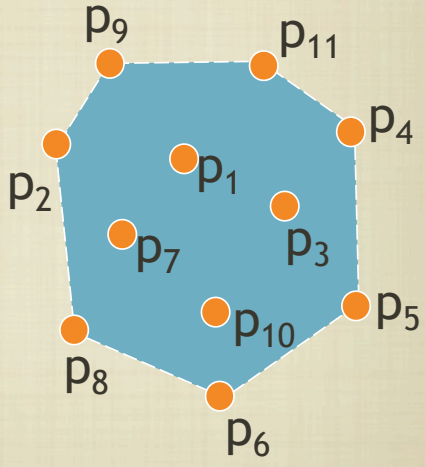
Algoritmo EC1:

Entrada: Conjunto de pontos P no plano

Saída: Uma lista de vértices da EC em ordem

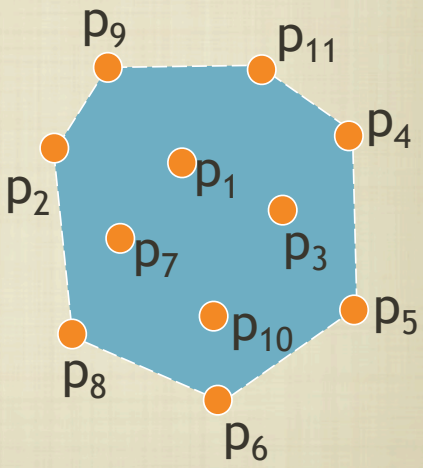
1. E = ø
2. FOR todos pares (p,q)∈PxP (p!=q)
3. DO valido = true
4. FOR todos pontos r∈P(r!=p and r!=q)
5. IF r está a direita da reta pq
6. THEN valido = false
7. IF valido THEN adicione pq para E
8. Para o conjunto E de arestas, construa uma lista de vertices classificados em ordem cronológica

$(n^2-n) * (n-2) = n^3-3n^2+2n$



Envoltória Convexa 1

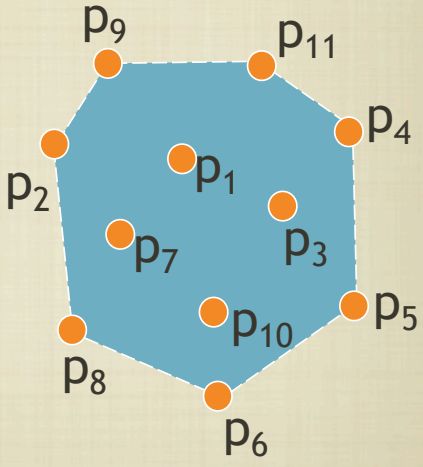
Algoritmo EC1:
Entrada: Conjunto de pontos P no plano
Saída: Uma lista de vértices da EC em ordem
1. E = ø
2. FOR todos pares (p,q)∈PxP (p!=q)
3. DO valido = true
4. FOR todos pontos r∈P(r!=p and r!=q)
5. IF r está a direita da reta pq
6. THEN valido = false
7. IF valido THEN adicione pq para E
8. Para o conjunto E de arestas, construa uma lista de vertices classificados em ordem cronológica



Como implementar o passo 8 ?

Envoltória Convexa 1

Algoritmo EC1:
Entrada: Conjunto de pontos P no plano
Saída: Uma lista de vértices da EC em ordem
1. E = ø
2. FOR todos pares (p,q)∈PxP (p!=q)
3. DO valido = true
4. FOR todos pontos r∈P(r!=p and r!=q)
5. IF r está a direita da reta pq
6. THEN valido = false
7. IF valido THEN adicione pq para E
8. Para o conjunto E de arestas, construa uma lista de vertices classificados em ordem cronológica

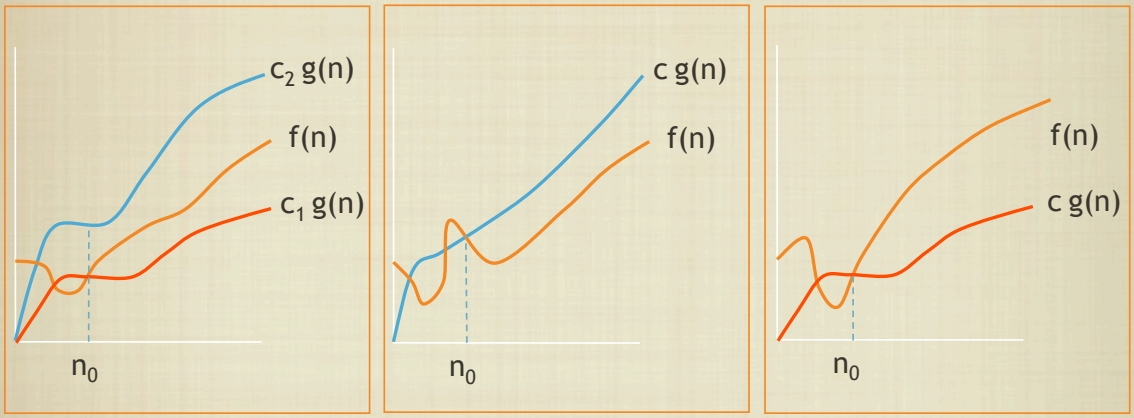


$n^3 - 3n^2 + 2n + n^2 = O(n^3)$

Complexidade de Algoritmos

- Modelo: máquina de acesso randômico
  - instruções sequenciais, sem ops. concorrentes
- Tamanho da entrada
- Tempo de execução:
  - número de operações
    - Linear ($A*N+B$)
    - Quadrático ($A*N^2+B*N+C$)
    - Cúbico ($A*N^3+B*N^2+C*N+D$)
    - Logarítmico: $A * \log N$
- Tamanho da saída

Complexidade de Algoritmos



$f(n) = \Theta(g(n))$ $f(n) = O(g(n))$ $f(n) = \Omega(g(n))$

Primitivas Geométricas

- Como testar se um ponto esta a esquerda ou a direita de uma linha ?
  - Ex. Ponto (5,7) e linha conectando pontos (2,3)e (3,8)
  - Ex. ponto (5,7) e linha 2x + 3y = 5

Primitivas Geométricas

- Linhas;
  - y = mx + b
  - ax + by + c = 0

```
typedef struct {
        double a;               /* x-coefficient */
        double b;               /* y-coefficient */
        double c;               /* constant term */
} line;
```

Primitivas Geométricas

```
points_to_line(point p1, point p2, line *l)
{
        if (p1[X] == p2[X]) {
                l->a = 1;
                l->b = 0;
                l->c = -p1[X];
        } else {
                l->b = 1;
                l->a = -(p1[Y]-p2[Y])/(p1[X]-p2[X]);
                l->c = -(l->a * p1[X]) - (l->b * p1[Y]);
        }
}

point_and_slope_to_line(point p, double m, line *l)
{
        l->a = -m;
        l->b = 1;
        l->c = -((l->a*p[X]) + (l->b*p[Y]));
}
```

REPRESENTAÇÃO CANÔNICA

Primitivas Geométricas

```
bool parallelQ(line l1, line l2)
{
     return ( (fabs(l1.a-l2.a) <= EPSILON) &&
              (fabs(l1.b-l2.b) <= EPSILON) );
}

bool same_lineQ(line l1, line l2)
{
     return ( parallelQ(l1,l2) && (fabs(l1.c-l2.c) <= EPSILON) );
}
```

Primitivas Geométricas

Interseção entre duas linhas

$$x = \frac{b_2 - b_1}{m_1 - m_2}, \qquad y = m_1 \frac{b_2 - b_1}{m_1 - m_2} + b_1$$

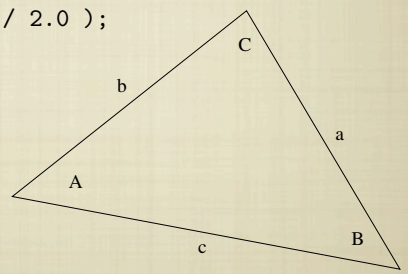
```
intersection_point(line l1, line l2, point p)
{
     if (same_lineQ(l1,l2)) {
         printf("Warning: Identical lines, all points intersect.\n");
         p[X] = p[Y] = 0.0;
         return;
     }

     if (parallelQ(l1,l2) == TRUE) {
         printf("Error: Distinct parallel lines do not intersect.\n");
         return;
     }

     p[X] = (l2.b*l1.c - l1.b*l2.c) / (l2.a*l1.b - l1.a*l2.b);

     if (fabs(l1.b) > EPSILON)       /* test for vertical line */
             p[Y] = - (l1.a * (p[X]) + l1.c) / l1.b;
     else
             p[Y] = - (l2.a * (p[X]) + l2.c) / l2.b;
}
```

Área de um triângulo

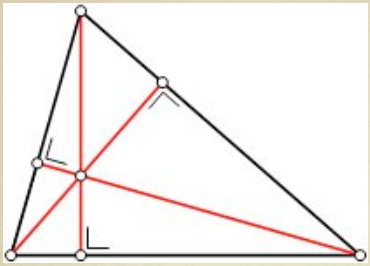
- **AREA(T), onde T é definido por pontos a, b, c**

$$2 \cdot A(T) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = a_x b_y - a_y b_x + a_y c_x - a_x c_y + b_x c_y - c_x b_y$$

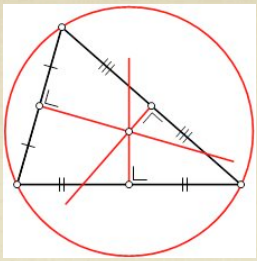
```
double signed_triangle_area(point a, point b, point c)
{
        return( (a[X]*b[Y] - a[Y]*b[X] + a[Y]*c[X]
                - a[X]*c[Y] + b[X]*c[Y] - c[X]*b[Y]) / 2.0 );
}

double triangle_area(point a, point b, point c)
{
        return( fabs(signed_triangle_area(a,b,c)) );
}
```



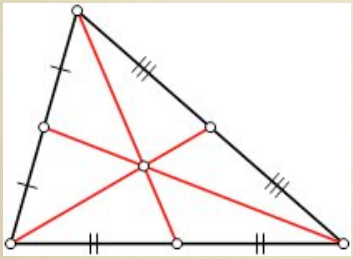
Propriedades de Triângulos



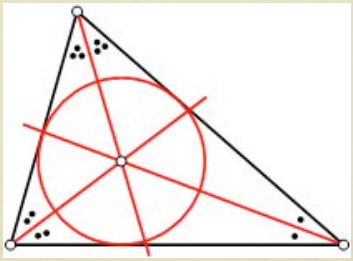
Ortocentro: Ponto de interseção das alturas



Circumcentro: Centro da circunferência circunscrita



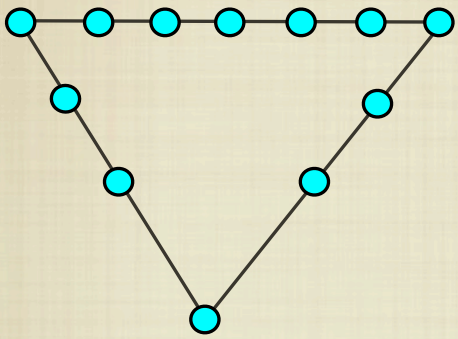
Baricentro: Ponto de interseção das medianas



Incentro: Ponto de interseção das 3 bissetrizes

Primitivas Geométricas

- O que acontece se os pontos estiverem sobre as linhas ? **CASOS DEGENERADOS**

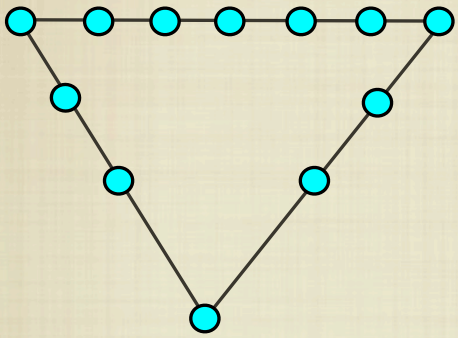


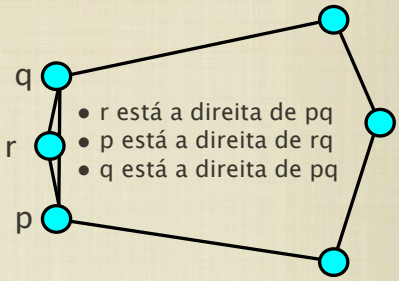
Uma aresta pq é uma aresta da envoltória convexa se e somente se todos os outros pontos da entrada estão estritamente a direita da linha dirigida entre p e q, ou eles se ele estão sobre a aresta pq

Primitivas Geométricas

- O que acontece se os pontos estiverem sobre as linhas ?

CASOS DEGENERADOS



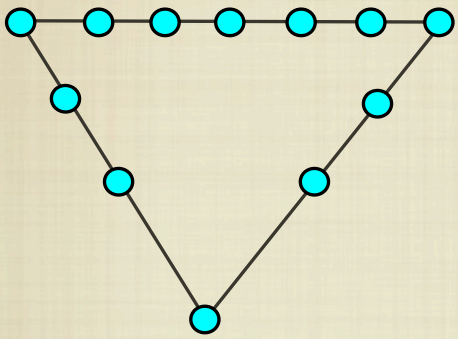


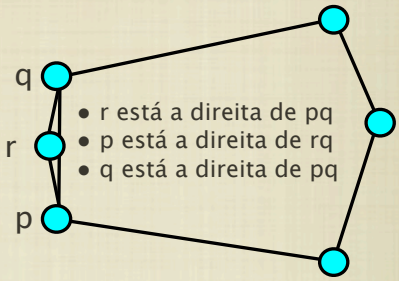
Uma aresta pq é uma aresta da envoltória convexa se e somente se todos os outros pontos da entrada estão estritamente a direita da linha dirigida entre p e q, ou eles se ele estão sobre a aresta pq

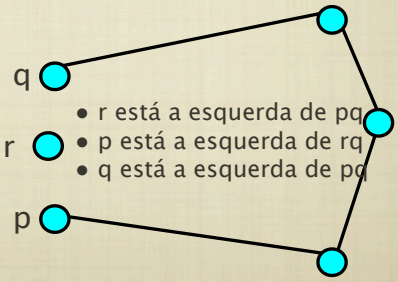
Primitivas Geométricas

- O que acontece se os pontos estiverem sobre as linhas ?

CASOS DEGENERADOS



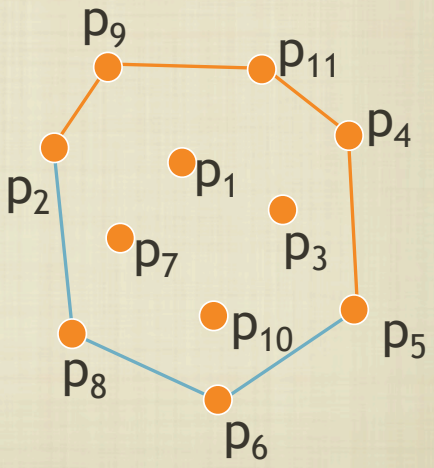




Uma aresta pq é uma aresta da envoltória convexa se e somente se todos os outros pontos da entrada estão estritamente a direita da linha dirigida entre p e q, ou eles se ele estão sobre a aresta pq

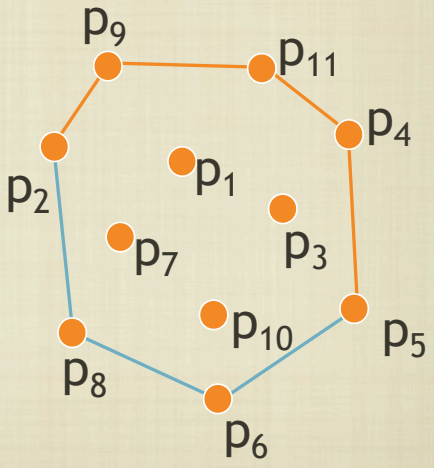
Envoltória Convexa Incremental

- Abordagem incremental: Adicionar pontos um por um e atualizar a solucao
-

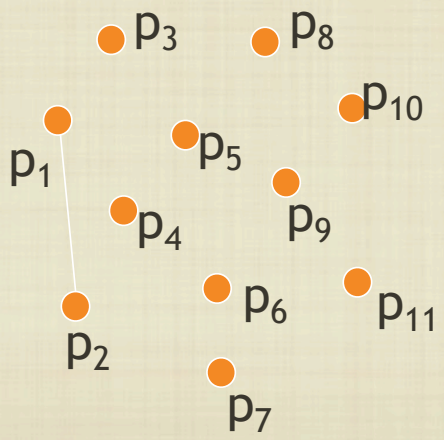


Envoltória Convexa Incremental

- Abordagem incremental: Adicionar pontos um por um e atualizar a solucao
- Classificar os pontos em x, obtendo uma sequencia ordenada $p_1, p_2, \ldots, p_n$
- Calcular a EC superior percorrendo a lista acima da esquerda para a direita
- Calcular a EC inferior percorrendo a lista acima da direita para a esquerda

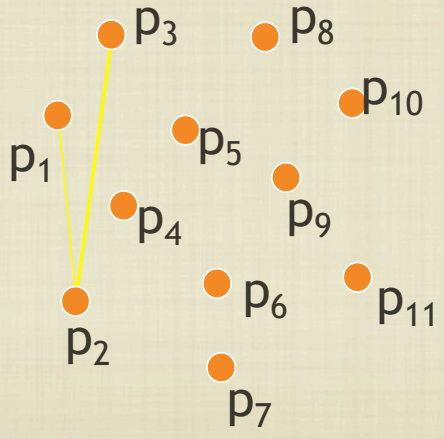


Envoltória Convexa Incremental

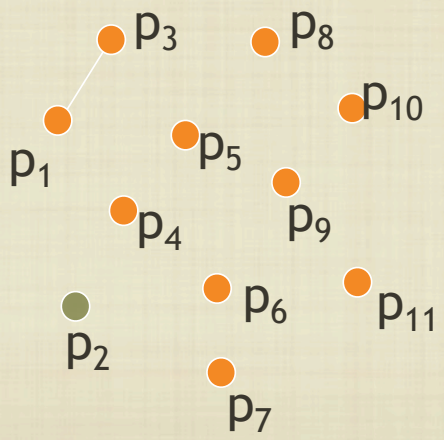


Envoltória Convexa Incremental

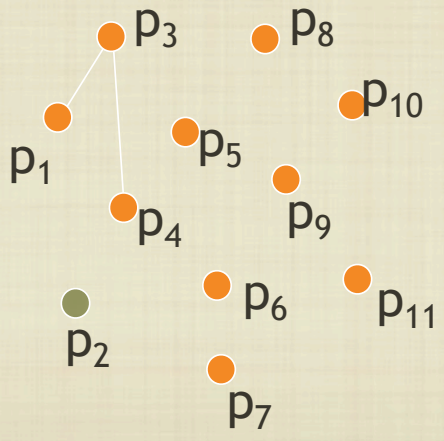
VIRADA P/ ESQ



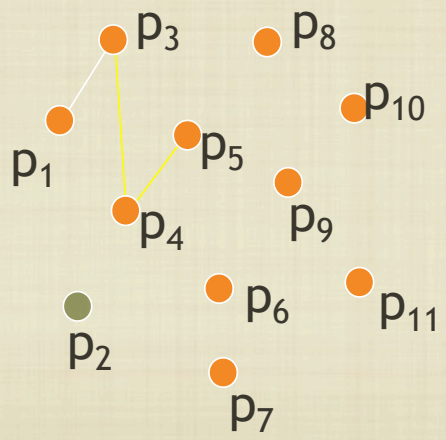
Envoltória Convexa Incremental



Envoltória Convexa Incremental

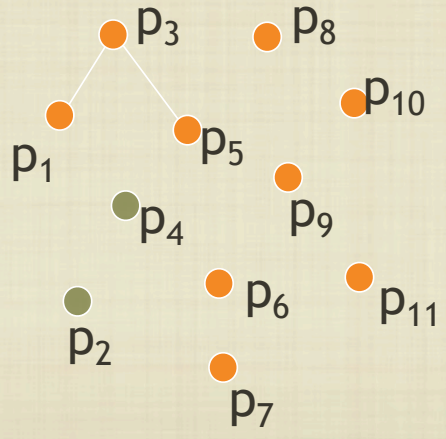


Envoltória Convexa Incremental



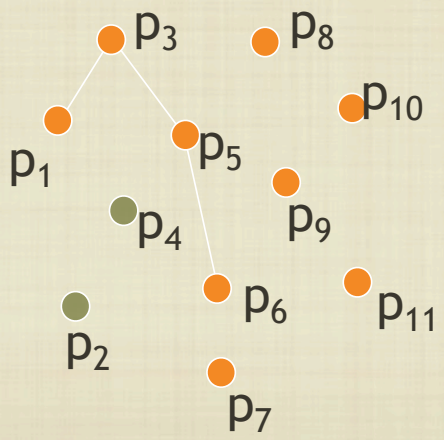
VIRADA P/ ESQ

Envoltória Convexa Incremental

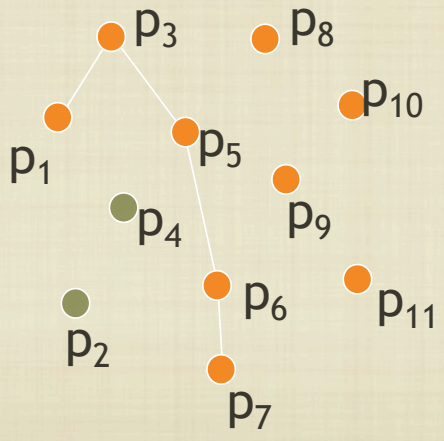


VIRADA P/ ESQ

Envoltória Convexa Incremental

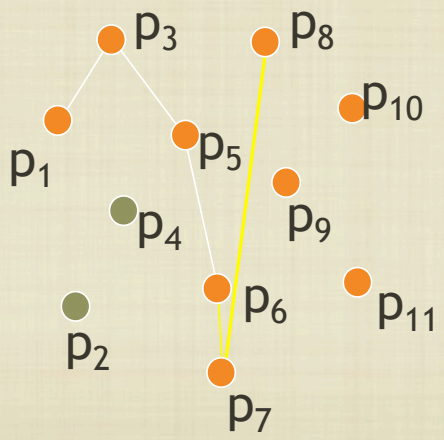


Envoltória Convexa Incremental



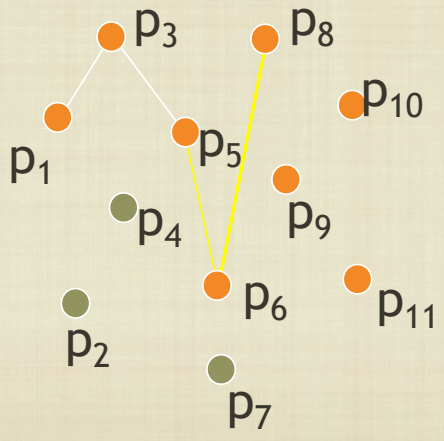
Envoltória Convexa Incremental

VIRADA P/ ESQ



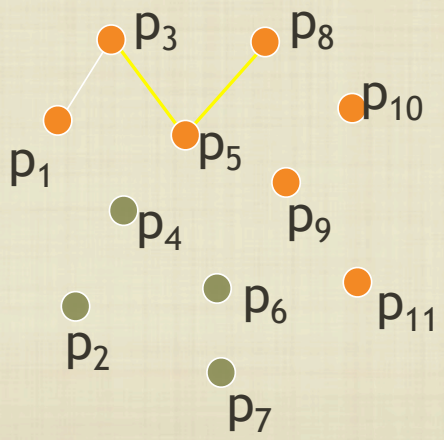
Envoltória Convexa Incremental

VIRADA P/ ESQ

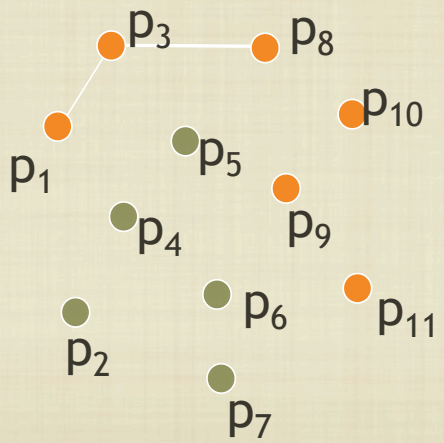


Envoltória Convexa Incremental

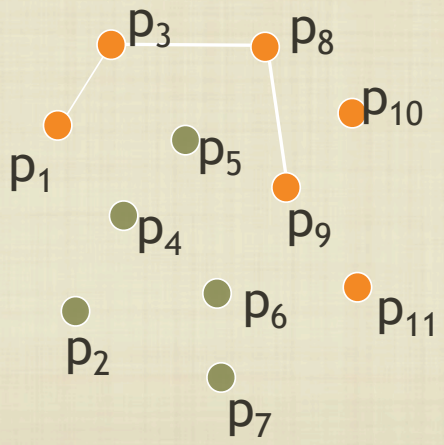
VIRADA P/ ESQ



Envoltória Convexa Incremental

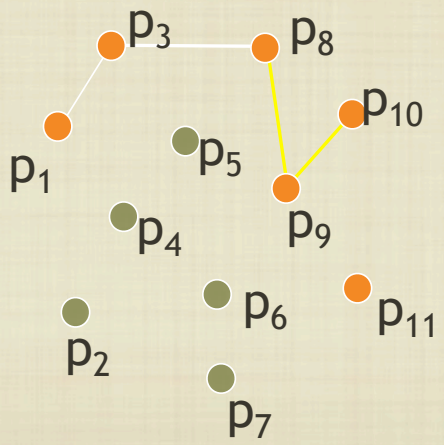


Envoltória Convexa Incremental

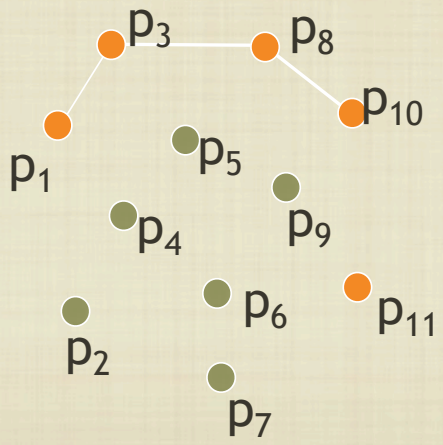


Envoltória Convexa Incremental

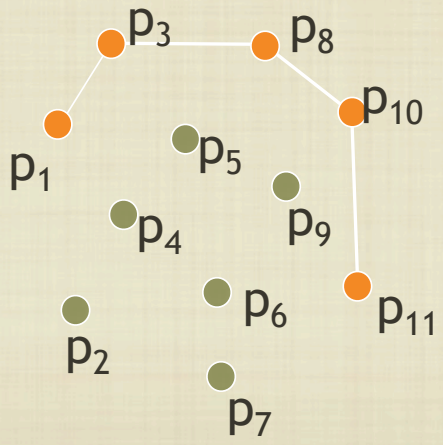
VIRADA P/ ESQ



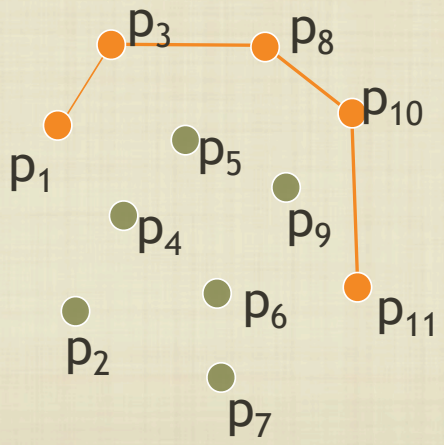
Envoltória Convexa



Envoltória Convexa Incremental



Envoltória Convexa Incremental



Envoltória Convexa 2 (Grahan's Scan 1972)

Algoritmo EC_GS:
Entrada: Conjunto de pontos P no plano
Saída: Uma lista de vértices da EC em ordem

1. Classificar os pontos pela coordenada x (p_1 .. p_n)
2. Colocar p_1 e p_2 na lista L_SUP
3. FOR i:=3 TO n
4. DO Adicionar p_i a L_SUP
5. WHILE L_SUP > 2 pontos e últimos três fazem uma virada para esquerda
6. DO delete o penultimo ponto de L_SUP
7. Coloque os pontos p_n e p_{n-1} na lista L_INF
8. FOR i: n-2 DOWNTO 1
9. Repita passos 4-6 com L_INF no lugar de L_SUP10.
10. Remover o primeiro e ultimo pontos de L_SUP e L_INF
11. RETURN Concatenacao de L_SUP e L_INF

Envoltória Convexa 2

Algoritmo EC_GS:
Entrada: Conjunto de pontos P no plano
Saída: Uma lista de vértices da EC em ordem

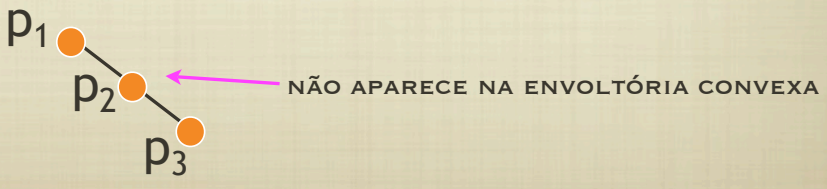
1. Classificar os pontos pela coordenada x (p_1 .. p_n)
2. Colocar p_1 e p_2 na lista L_SUP
3. FOR i:=3 TO n
4. DO Adicionar p_i a L_SUP
5. WHILE L_SUP > 2 pontos e últimos três fazem uma virada para esquerda
6. DO delete o penultimo ponto de L_SUP
7. Coloque os pontos p_n e p_{n-1} na lista L_INF
8. FOR i: n-2 DOWNTO 1
9. Repita passos 4-6 com L_INF no lugar de L_SUP10.
10. Remover o primeiro e ultimo pontos de L_SUP e L_INF
11. RETURN Concatenacao de L_SUP e L_INF

$$O(n\log n) + O(n) = O(n\log n)$$

Casos Degenerados

- O que acontece se 2 pontos possuem a mesma coordenada x ?
  - A ordenação na coordenada x é incorreta

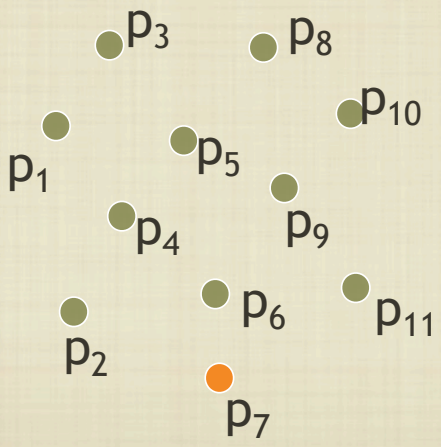
- O que acontece se 3 pontos são colineares ?



Problema 1

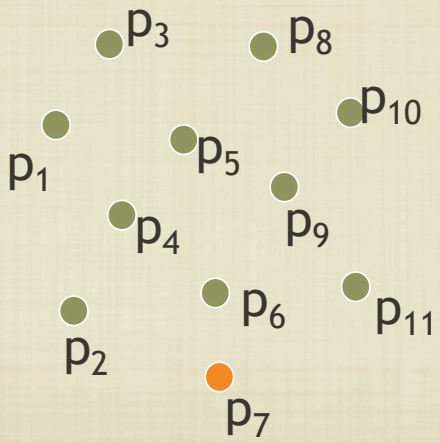
- Resolver Ex. 681: Convex Hull Finding
  - http://acm.uva.es/p/v6/681.html
  - Data 18/3 (Mandar e-mail com aceitação Em Valladolid

Gift Wrapping



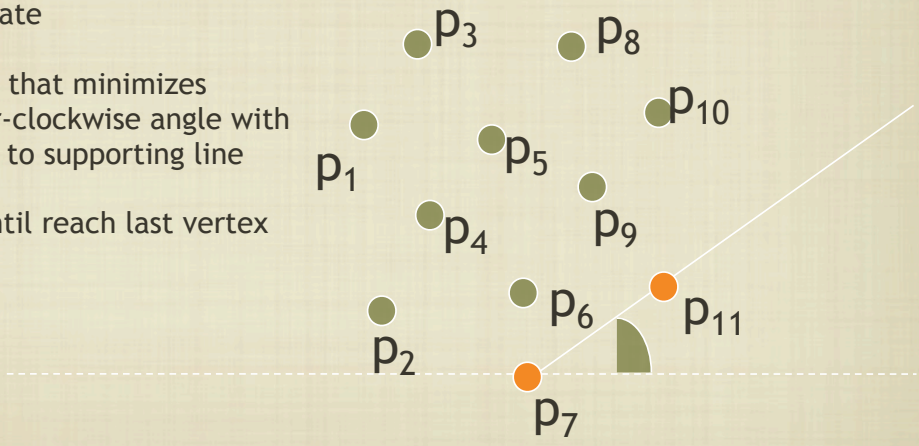
Gift Wrapping

1. Find point with smallest y coordinate



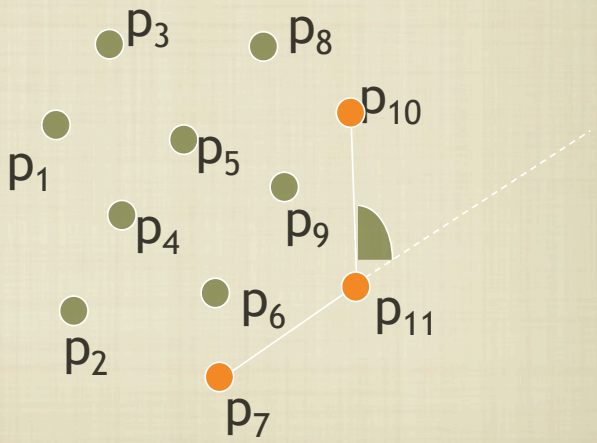
Gift Wrapping

1. Find point with smallest y coordinate
2. Find edge that minimizes counter-clockwise angle with respect to supporting line
3. Repeat until reach last vertex



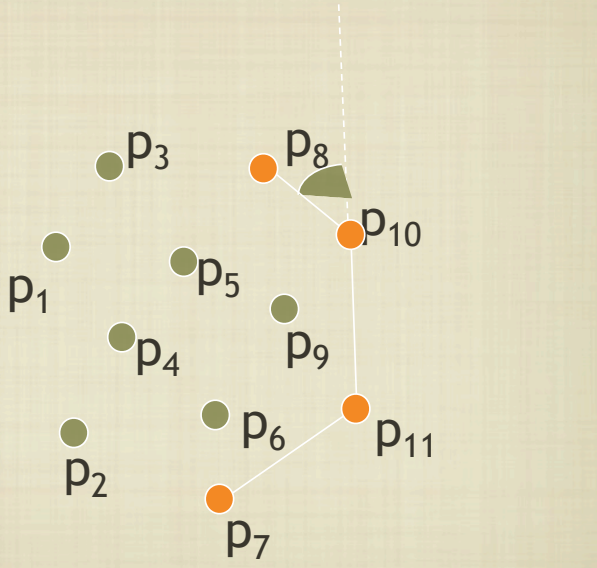
Gift Wrapping

1. Find point with smallest y coordinate
2. Find edge that minimizes counter-clockwise angle with respect to supporting line
3. Repeat until reach last vertex



Gift Wrapping

1. Find point with smallest y coordinate
2. Find edge that minimizes counter-clockwise angle with respect to supporting line
3. Repeat until reach last vertex

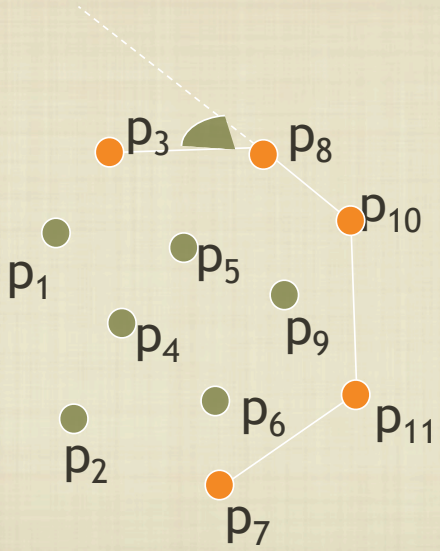


Gift Wrapping

1. Find point with smallest y coordinate

2. Find edge that minimizes counter-clockwise angle with respect to supporting line

3. Repeat until reach last vertex

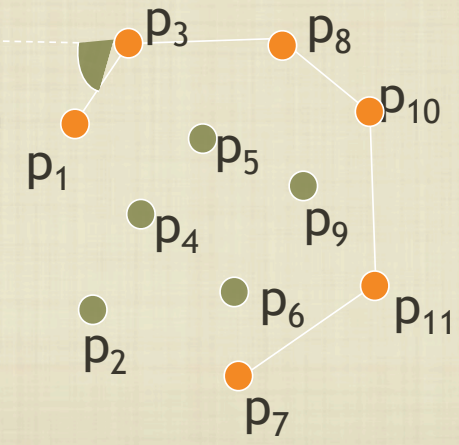


Gift Wrapping

1. Find point with smallest y coordinate

2. Find edge that minimizes counter-clockwise angle with respect to supporting line

3. Repeat until reach last vertex

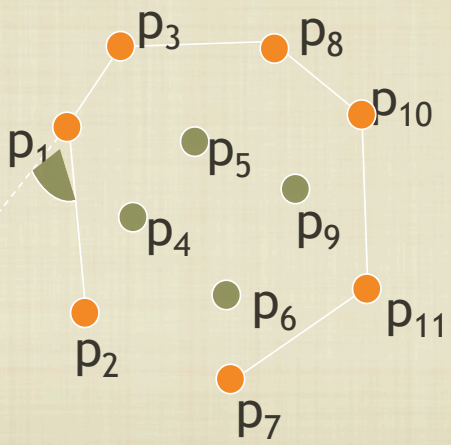


Gift Wrapping

1. Find point with smallest y coordinate

2. Find edge that minimizes counter-clockwise angle with respect to supporting line

3. Repeat until reach last vertex

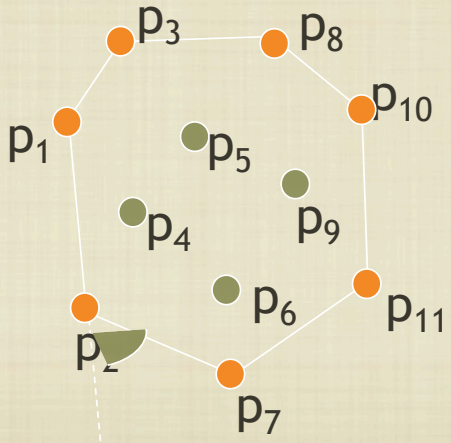


Gift Wrapping

1. Find point with smallest y coordinate

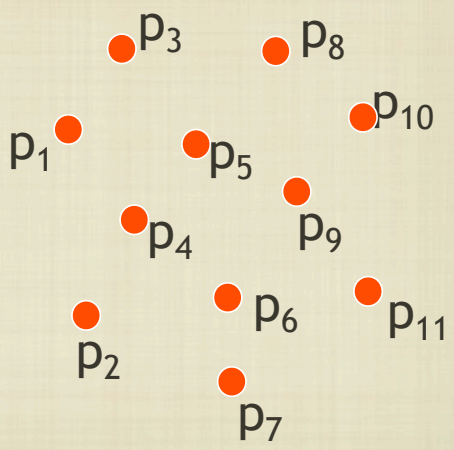
2. Find edge that minimizes counter-clockwise angle with respect to supporting line

3. Repeat until reach last vertex



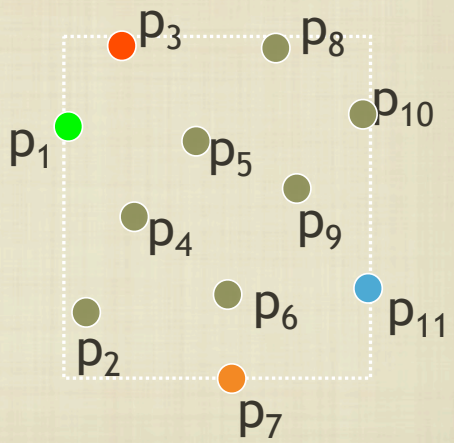
$O(n^2)$

QUICKHULL



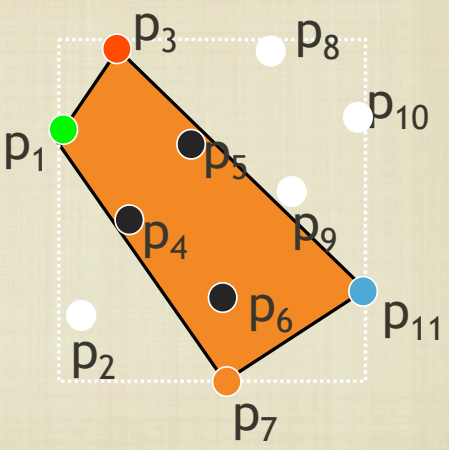
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



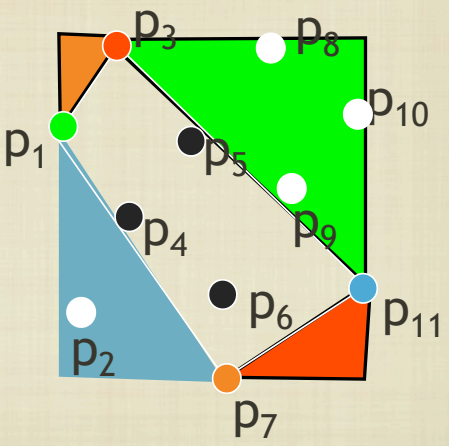
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points



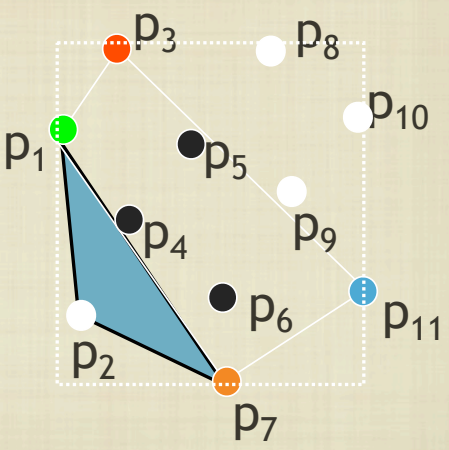
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively



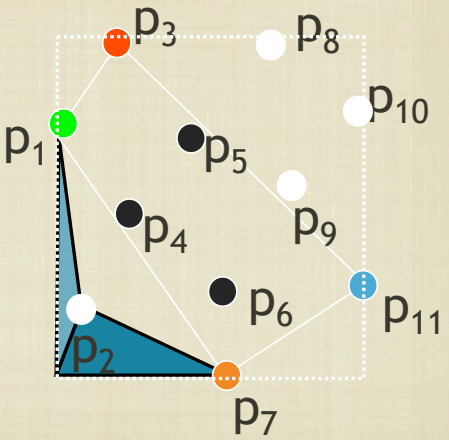
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles



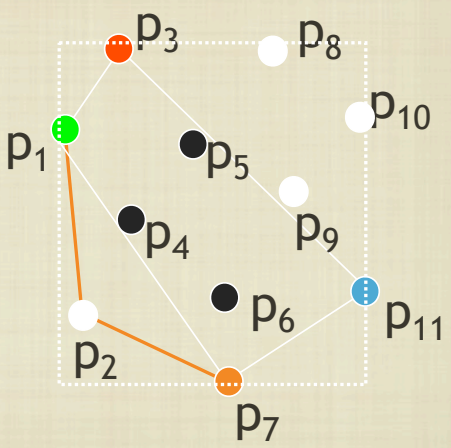
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



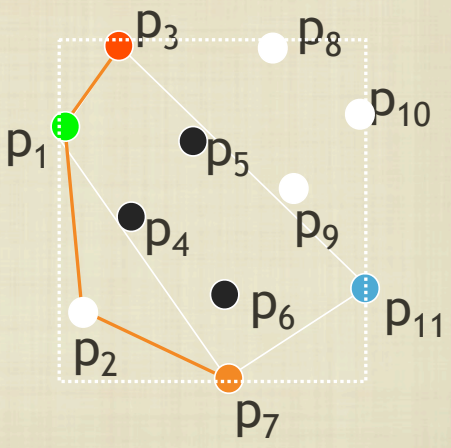
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



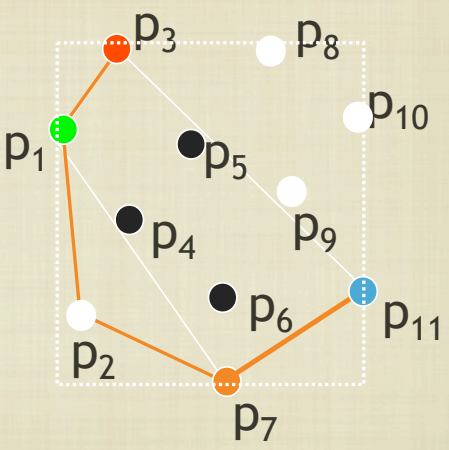
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



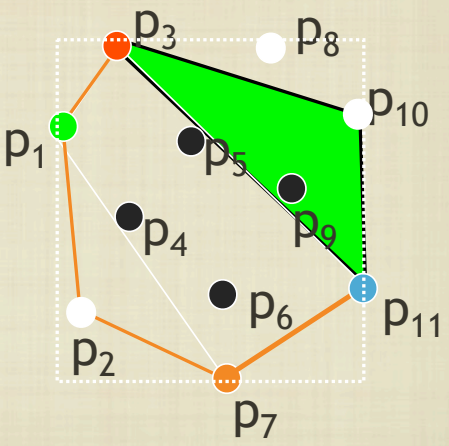
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



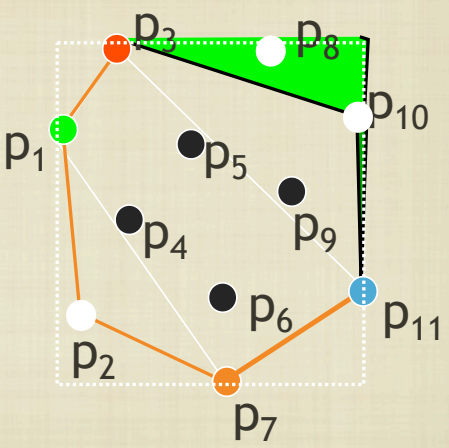
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



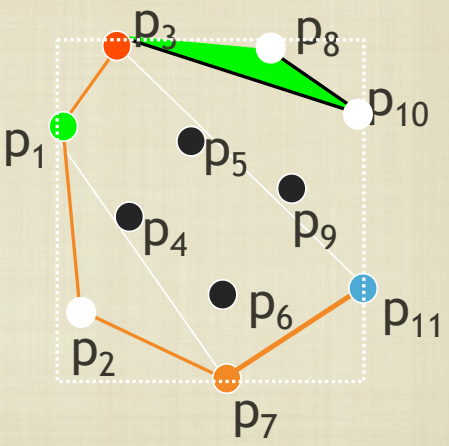
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



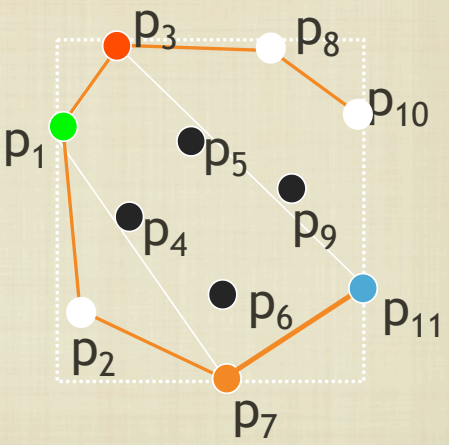
QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively

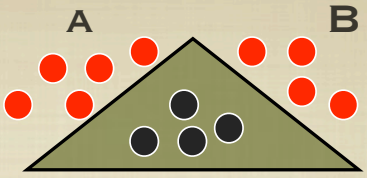


QUICKHULL

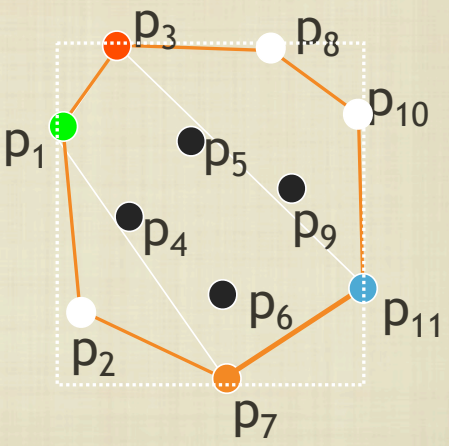
1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



QUICKHULL



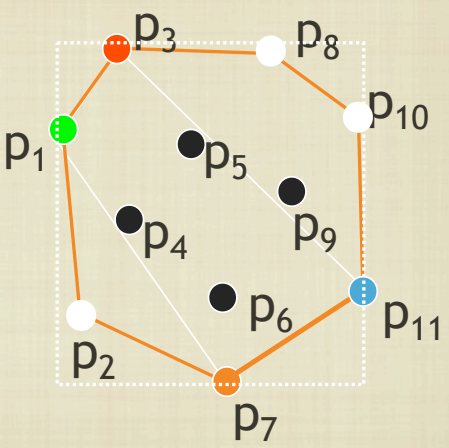
1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



QUICKHULL

$T(n) = O(n) + T(a) + T(b)$

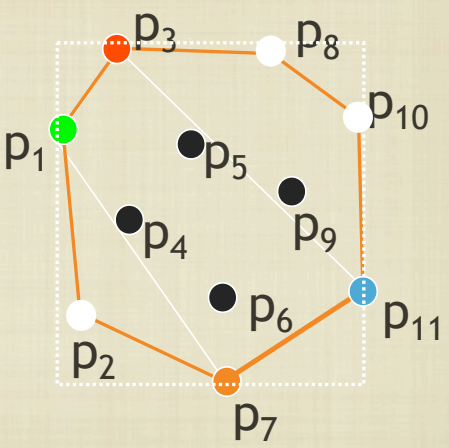
1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively



QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively

$T(n) = O(n) + T(a) + T(b)$

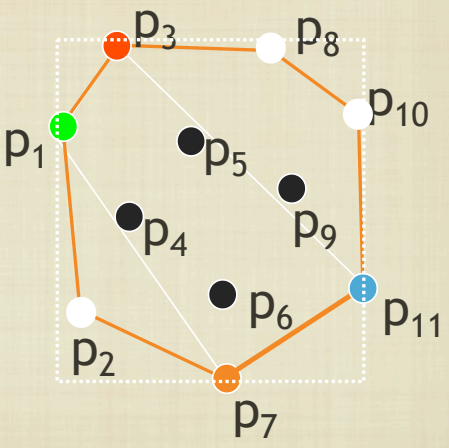


$T(n) = O(n) + 2T(n/2)$

QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively

$T(n) = O(n) + T(a) + T(b)$

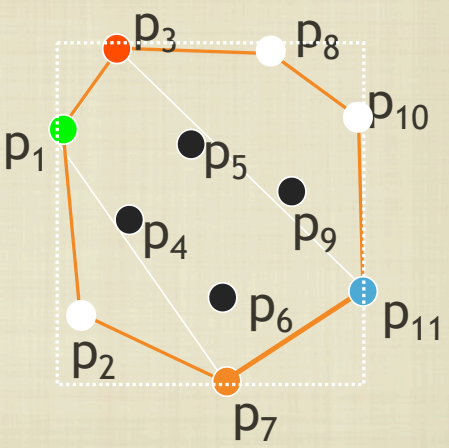


$T(n) = O(n) + 2T(n/2) \rightarrow O(n \log n)$

QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively

$T(n) = O(n) + T(a) + T(b)$



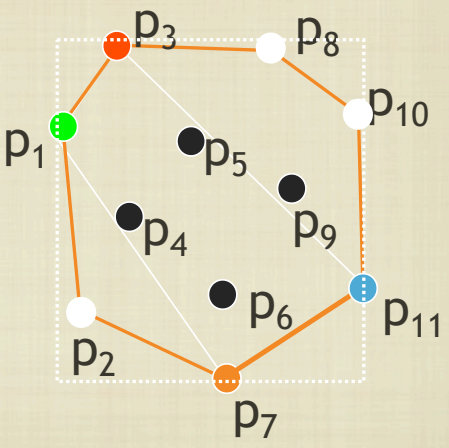
$T(n) = O(n) + 2T(n/2) \rightarrow O(n \log n)$

$T(n) = O(n) + T(n-1)$

QUICKHULL

1. Find extreme points above, below, left and right
2. Define a quadrilateral connecting extreme points, discard internal points
3. Process 4 triangles recursively
4. For each triangle, find vertex with greater distance from the triangle baseline.
5. Define another triangle and discard internal triangles
6. Process 2 triangles recursively

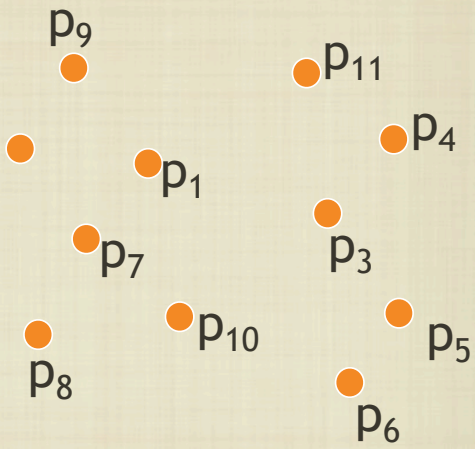
$T(n) = O(n) + T(a) + T(b)$



$T(n) = O(n) + 2T(n/2) \rightarrow O(n \log n)$

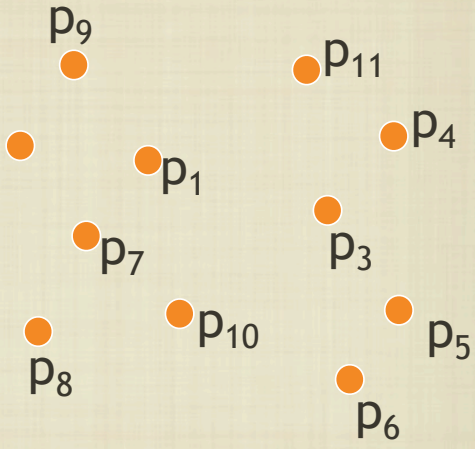
$T(n) = O(n) + T(n-1) \rightarrow O(n^2)$

Divide-and-Conquer



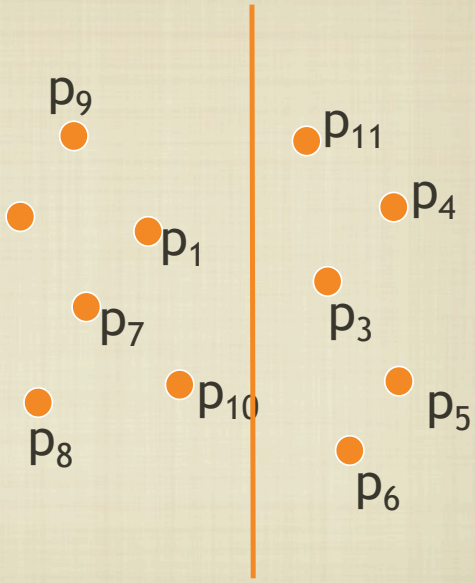
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull



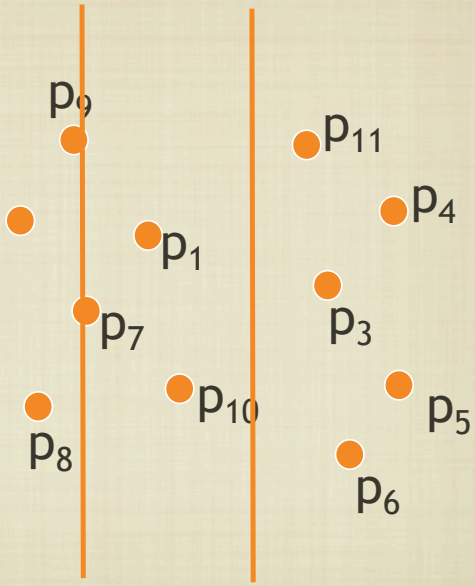
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull



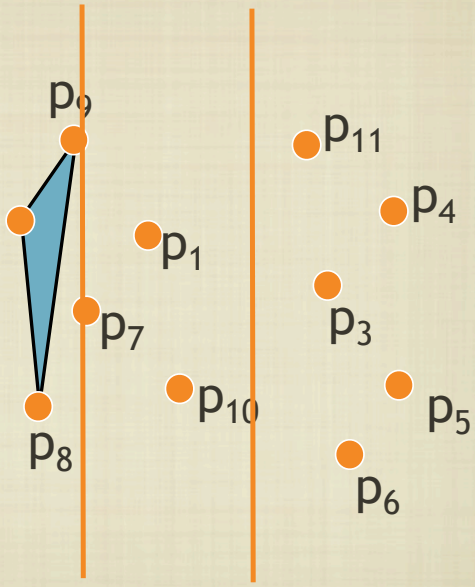
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull



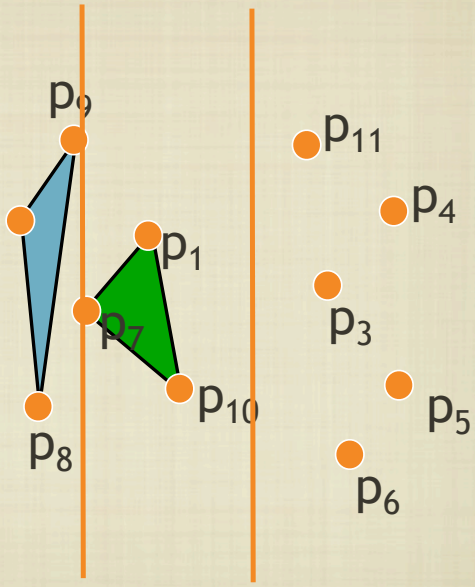
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull



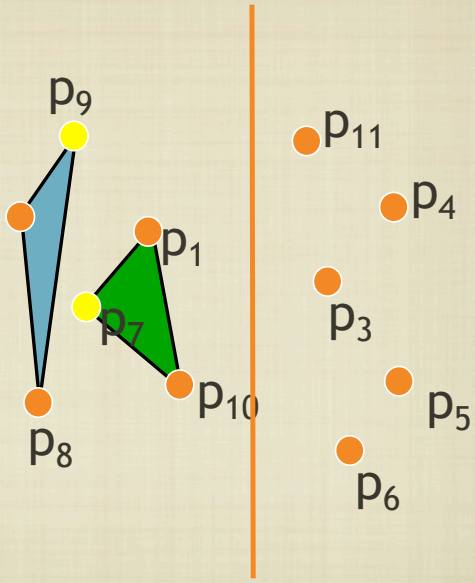
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull



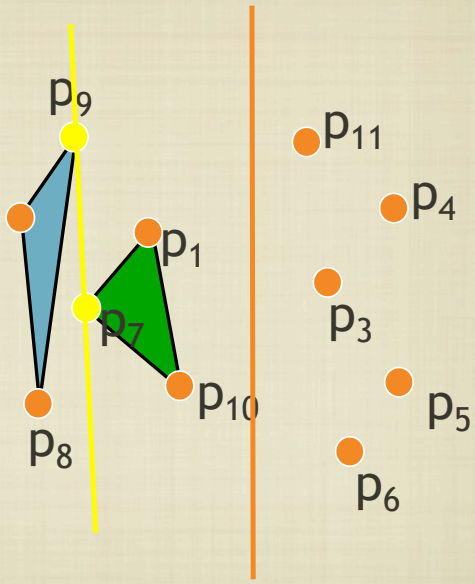
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



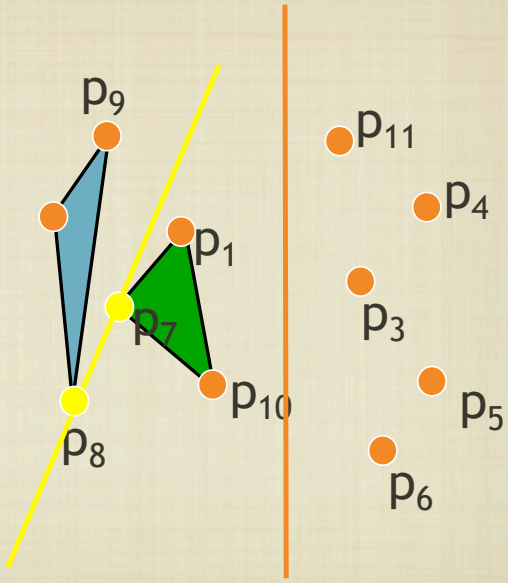
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



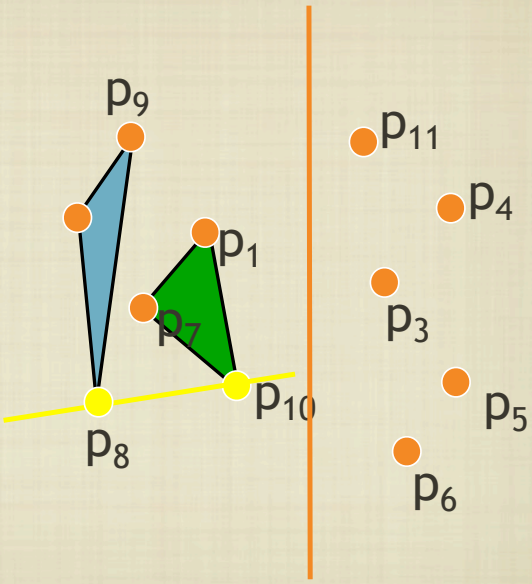
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



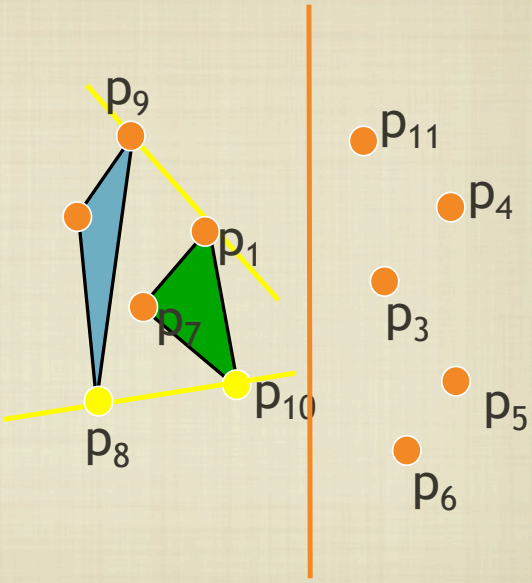
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



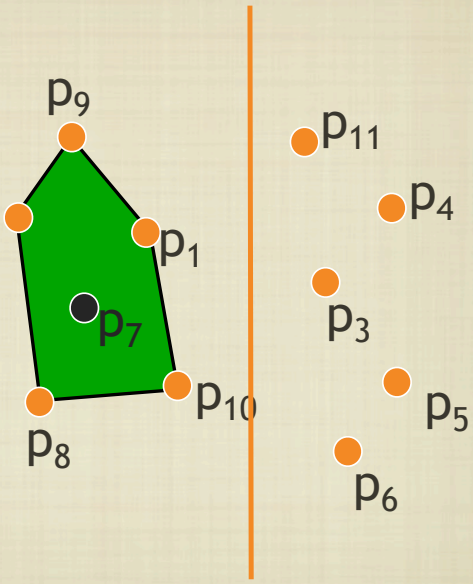
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



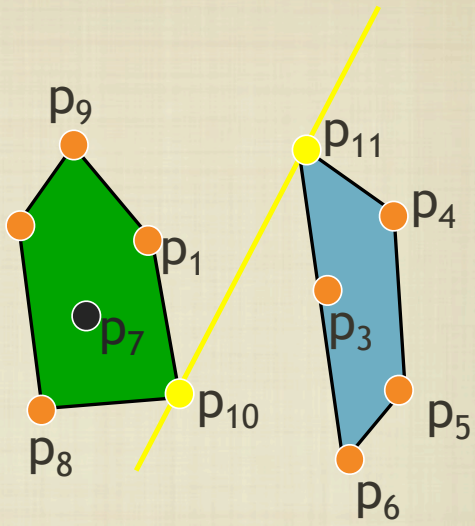
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



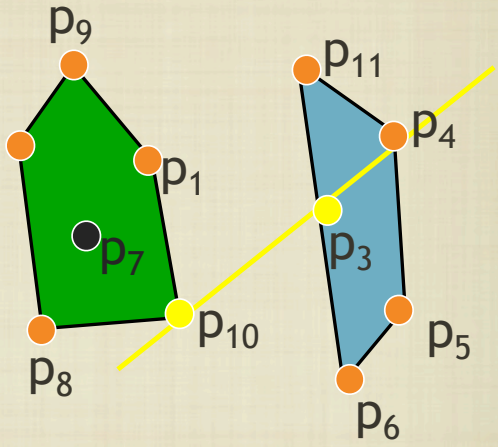
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



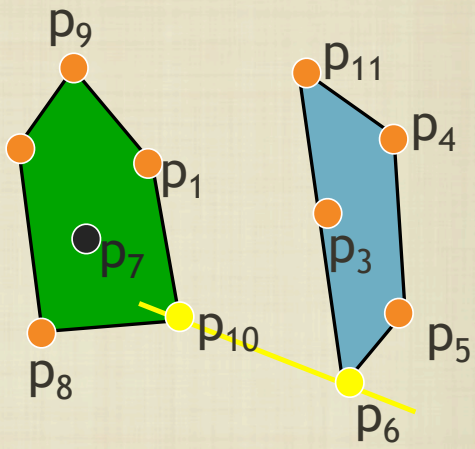
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



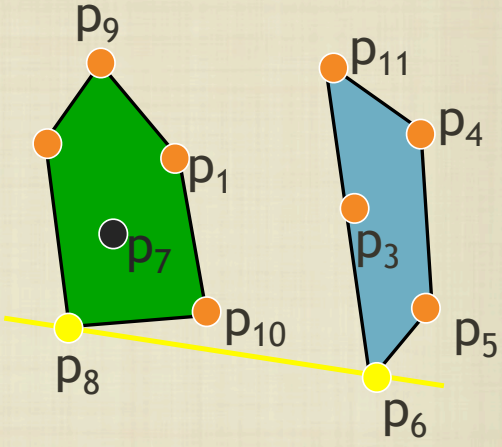
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



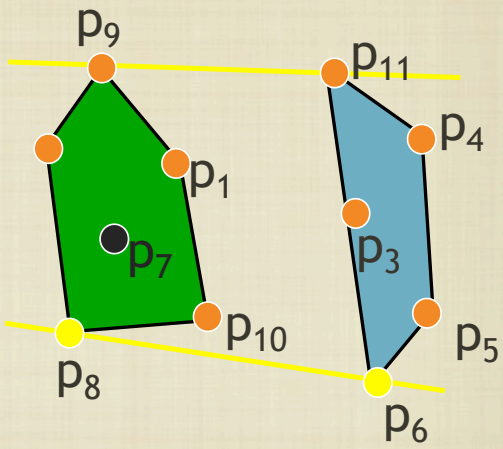
Divide-and-Conquer

1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b



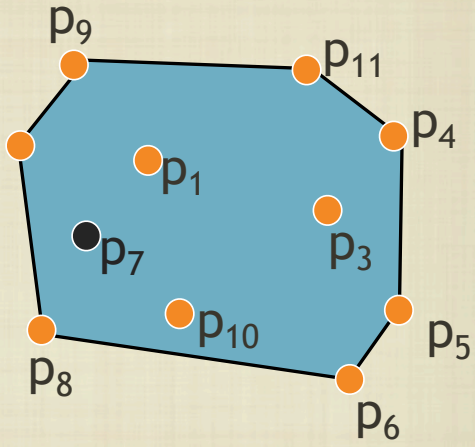
Divide-and-Conquer

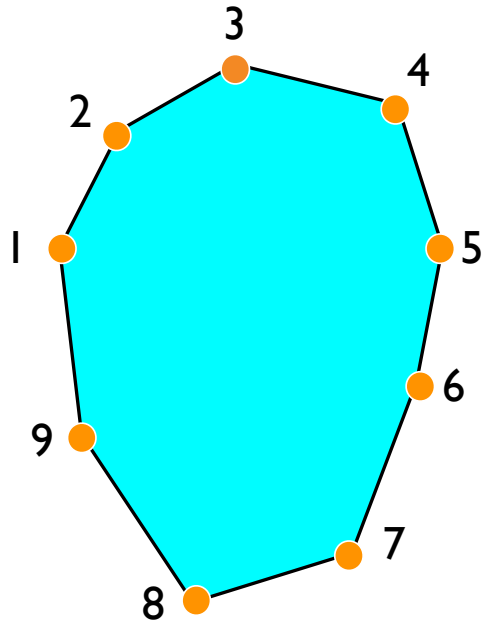
1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b

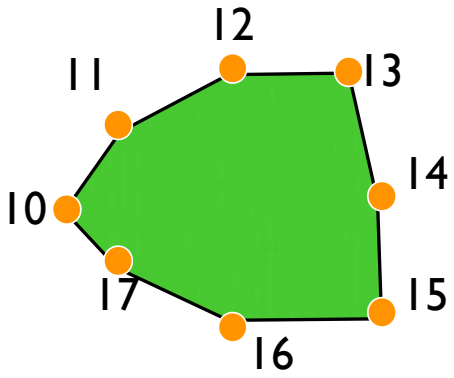


Divide-and-Conquer

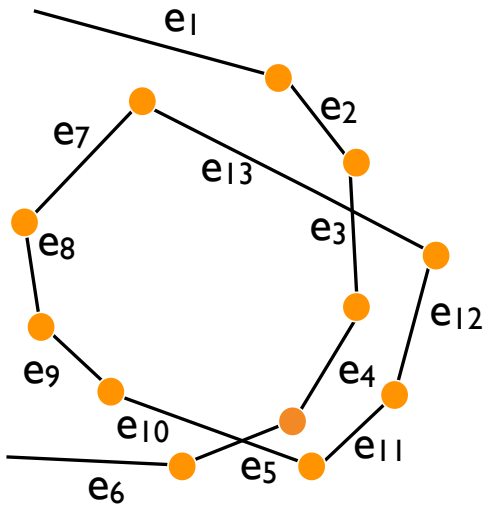
1. Sort points in X
2. Divide points in two sets A and B with similar number of elements
3. Compute Convex Hull of A and B recursively
4. Merge A and B to define convex hull
   - Find lower tangent common to A and B
   - Find rightmost point a of A and leftmost point b of B
   - While ab is not lower tangent to a, decrease a
   - While ab is not lower tangent of b, decrease b

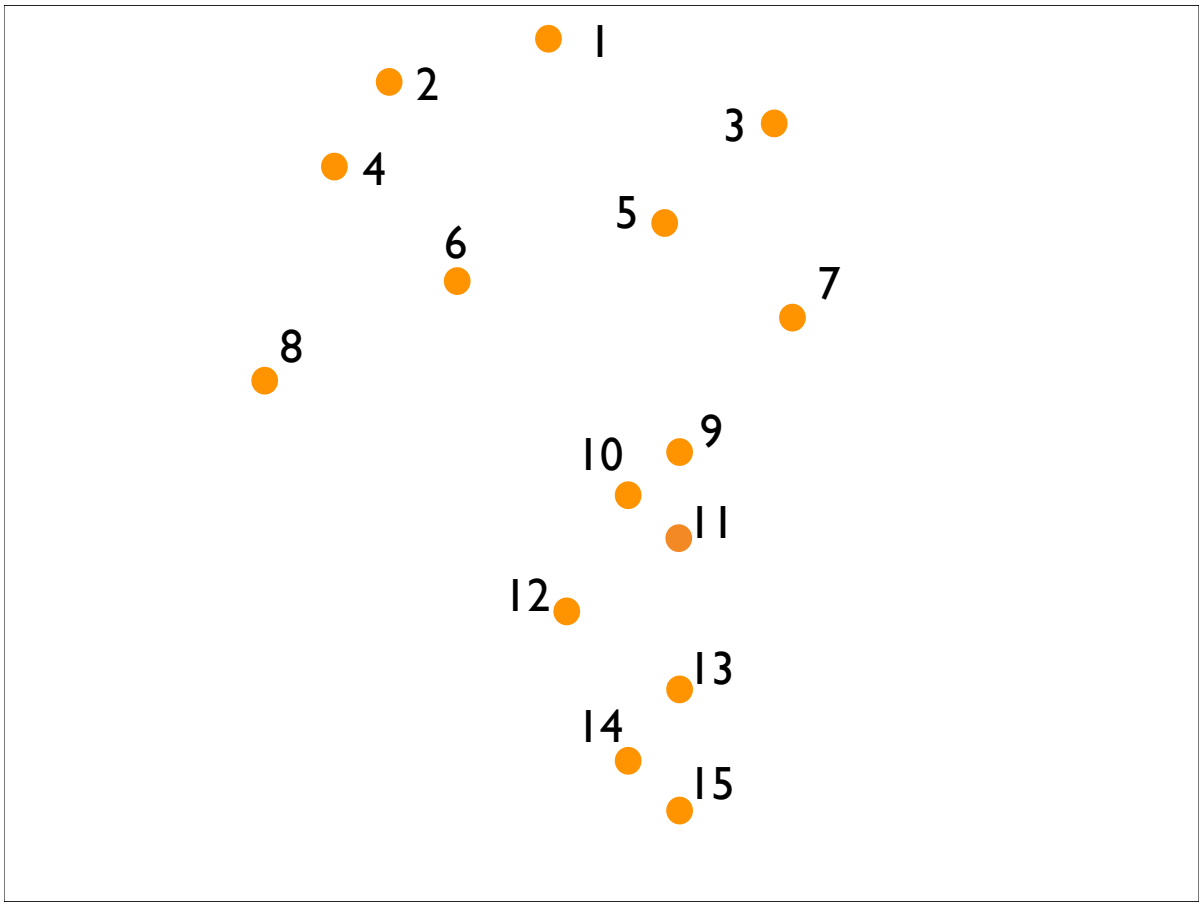






LEFT CH RIGHT CH




1
2
3
4
5
6
7
8
9
10
11
12
13
14
15