Divide and conquer (D&C) algorithms

- General method
- Matrix multiplication
- Strassen matrix multiplication

General method

- Given a function to compute on n inputs, the divide and conquer strategy suggests splitting the input in to k distinct subsets, yielding k subproblems.
- These sub-problems must be solved, then a method must be found to combine sub-solutions into a solutions of the whole.
- Often the subproblems are of the same type as the original problem. If the sub-problems are still large, apply D&C to the sub-problem (use recursion).
- Smaller and smaller subproblems are produced until the problem is small enough, which can be solved without splitting.

Algorithm D&C(P)

if Small(P) return Solve(P)

else{

Divide P into smaller instances $P_1, P_2, \ldots P_k$

Apply D&C to each of these subproblems

return Combine(D&C(P_1*), D&C(*P_2*), …,D&C(*P_k*))*

}

- *Small(P)* is a Boolean valued function that determines whether the problem is small enough.
- If yes, the function *Solve(P)* is invoked.
- otherwise, each of subproblems is solved by *D&C* algorithm.
- *Combine* is a function that determines the solution of *P* using the solutions to *k* sub-problems.

Matrix multiplication

Strassen Matrix multiplication

Convex hull

Master theorem

Divide and Conquer

max-min

Selection

Multiplication of two integers

Matrix multiplication

- Example: Calculate

$$C=AB=\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}\begin{pmatrix}5 & 6\\ 7 & 8\end{pmatrix}$$

$$=\begin{pmatrix}1*5+2*7 & 1*6+2*8\\ 3*5+4*7 & 3*6+4*8\end{pmatrix}=\begin{pmatrix}19 & 22\\ 43 & 50\end{pmatrix}$$

- How many multiplications: 8
- How many additions?
- If matrices are 4 by 4, how many multiplications and additions?
- If matrices are n by n, how many multiplications and additions?

- **Definition** If A is an $m \times n$ matrix and B is an $n \times p$ matrix,

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{pmatrix}$$

- The matrix product $C = AB$ is defined to be the $m \times p$ matrix

$$C = \begin{pmatrix} c_{11} & \cdots & c_{1p} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mp} \end{pmatrix}$$

- Such that $c_{ij} = a_{i1}\, b_{1j} + a_{i2}\, b_{2j} + \ldots + a_{in}\, b_{nj}$

$$= \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

- We will consider the case where $n = m = p$, for simplicity

Algorithm MatMult(A, B, C: matrix)

Input: matrices A; B

output: matrix C

```
for(i= 1; i <=n;  i+ +) {
    for(j= 1; j <=n; j+ +){
            C[i; j] = 0;
        for(k= 1; k <=n; k+ +){
            C[i, j] =C[i, j] +A[i, k]*B[k, j];
        }
    }
}
```

- Complexity: $T(n) = O(n^3)$

In which order Matrix C is filled?

$$\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} & & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & & c_{43} & c_{44} \end{bmatrix}$$

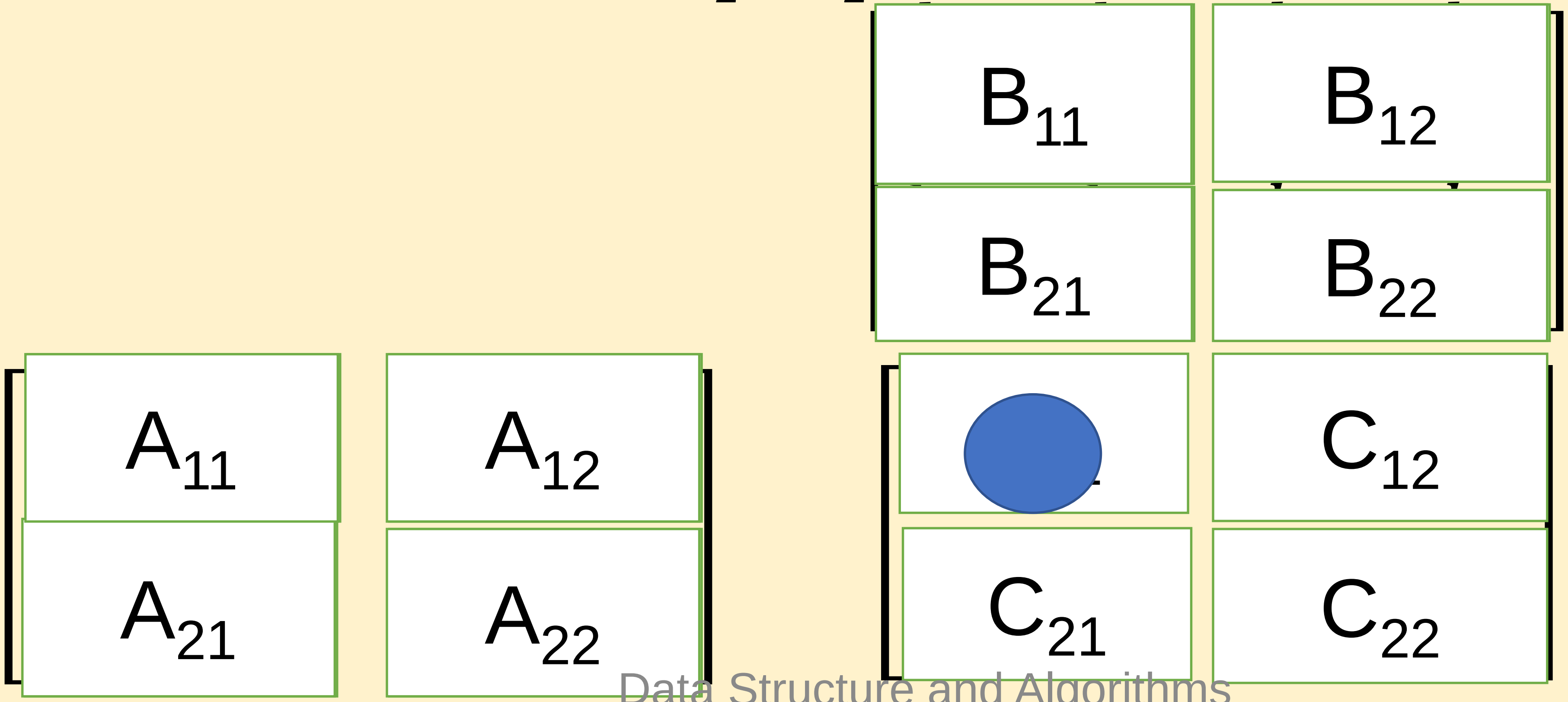
D &C Matrix Multiplication

$n = \{2,4,8,16,32,64\ldots\}$

- Divide problem by partitioning $C=AB$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

where each of A_{ij}, B_{ij}, C_{ij} is $\frac{n}{2}$ by $\frac{n}{2}$ matrices,



- Divide problem by partitioning C=AB as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

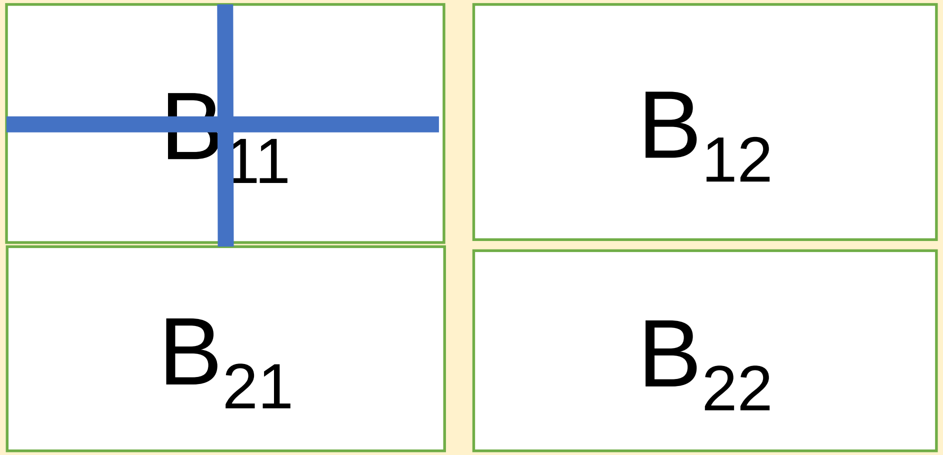
where each of A_{ij} , B_{ij} , C_{ij} is $\frac{n}{2}$ by $\frac{n}{2}$ matrices, with

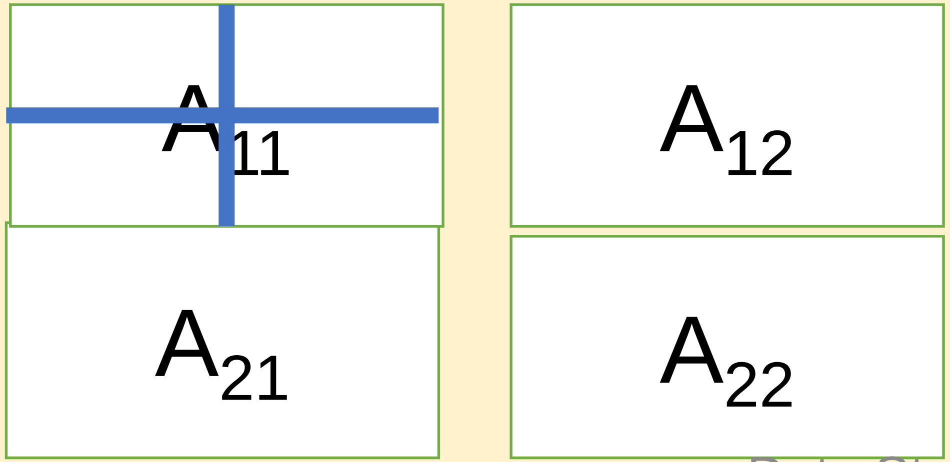
- Combine solutions

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

- For simplicity, we assume that $n = 2^k$, then each of the matrix products $A_{ij}\ B_{ij}$ can be recursively calculated using the same formula by D&C.

- $n=2^k$ divide into eight 2^{k-1} multiplications
- Each submatrix is divided in the same manner, recursively, until its size is $n=2$, so 2 by 2, which is direct solved





- *The algorithm Solve C=AB by D&C*
- *Small(P) is true for n≤2: direct solve*
- *Divide into subproblems* **e.g** $A_{11}\ B_{11}$ = D&CMatMul(A_{11}, B_{11})
- *Combine the solutions*

Algorithm D&CMatMult(A, B)

If (n <= 2) return(C_{11} ,C_{12} , C_{21} , C_{22});

else{

Recursively compute all combinations;

$C_{11} = A_{11}\ B_{11} + A_{12}\ B_{21}$

$C_{12} = A_{11}\ B_{12} + A_{12}\ B_{22}$

$C_{21} = A_{21}\ B_{11} + A_{22}\ B_{21}$

$C_{22} = A_{21}\ B_{12} + A_{22}\ B_{22}$

return(C_{11} ,C_{12} , C_{21} , C_{22});

}

}

- eight multiplications of size $\frac{n}{2}$ by $\frac{n}{2}$ matrices.
  - Four additions of $\frac{n}{2}$ by $\frac{n}{2}$ which is $O(n^2)$.
- So $T(n) = 8T(\frac{n}{2}) + c\,n^2$ for $n > 2$

Strassen Matrix Multiplication

- Matrix multiplication is more expensive than additions ($O(n^3)$ versus $O(n^2)$).
- Strassen's Matrix Multiplication approach is to reduce the number of multiplications at the expense in additions
- D&C Matrix Multiplication approach
  - 8 multiplications
  - 4 additions or subtractions
- Strassen's Matrix Multiplication approach
  - 7 multiplications
  - 18 additions or subtractions

1. Divide problem by partitioning $C=AB$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

2. Compute seven matrices as

$$P = (A_{11} + A_{22}) \boxed{*} (B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22}) \boxed{*} B_{11}$$
$$R = A_{11} \boxed{*} (B_{12} - B_{22})$$
$$S = A_{22} \boxed{*} (B_{21} - B_{11})$$
$$T = (A_{11} + A_{12}) \boxed{*} B_{22}$$
$$U = (A_{21} - A_{11}) \boxed{*} (B_{11} + B_{12})$$
$$V = (A_{12} - A_{22}) \boxed{*} (B_{21} + B_{22})$$

3. Combine the solutions as

$$C_{11} = P + S - T + V$$
$$C_{21} = Q + S$$
$$C_{12} = R + T$$
$$C_{22} = P + R - Q + U$$

- 7 multiplications
- 18 additions or subtractions

- Checking if $C_{11} = P + S - T + V$ is correct

$P = (A_{11} + A_{22}) * (B_{11} + B_{22}) =$

$S = A_{22} * (B_{21} - B_{11}) =$

$-T = -(A_{11} + A_{12}) * B_{22} =$

$V = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$$P+S-T+V = A_{11}B_{11} + A_{12}B_{21}$$
$$= C_{11}$$

Exercise: Checking if $C_{12} = R + T$ is correct

$$R = A_{11} * (B_{12} - B_{22})$$

$$T = (A_{11} + A_{12}) * B_{22}$$

$$R + T =$$

$$C_{12} =$$

Exercise: Checking if $C_{21} = Q + S$ is correct

$Q = (A_{21} + A_{22}) * B_{11}$

$S = A_{22} * (B_{21} - B_{11})$

$Q + S =$

$C_{21} =$

Exercise: Checking if $C_{22} = P + R - Q + U$ is correct

$P = (A_{11} + A_{22}) * (B_{11} + B_{22}) =$

$-Q = -(A_{21} + A_{22}) * B_{11} =$

$R = A_{11} * (B_{12} - B_{22}) =$

$U = (A_{21} - A_{11}) * (B_{11} + B_{12})$

$P + R - Q + U =$

$C_{22} =$

- Time complexity

$$T(n) = 7T\left(\frac{n}{2}\right) + an^2 \quad \text{for} \quad n > 2$$

$$T(n) = 1 \quad \text{for} \quad n \leq 2$$

- Solving recurrence

$$T(n) = a * n^2\left(1 + \frac{7}{4} + \left(\frac{7}{4}\right)^2 + \cdots \left(\frac{7}{4}\right)^{k-1}\right) + 7^k T\left(\frac{n}{2^k}\right)$$

$$\leq cn^2\left(\frac{7}{4}\right)^{\log n} + 7^{\log n}$$

(Where c is a constant)

$$= O\left(n^{\log 7}\right) = O\left(n^{2.81}\right)$$

- In practice overhead of implementation outweighs benefit.

Divide and conquer (D&C) algorithms

- General method
- Multiplication of two binary integers
- Convex hull

General method

- Given a function to compute on n inputs, the divide and conquer strategy suggests splitting the input in to k distinct subsets, yielding k subproblems.
- These sub-problems must be solved, then a method must be found to combine sub-solutions into a solutions of the whole.
- Often the subproblems are of the same type as the original problem. If the sub-problems are still large, apply D&C to the sub-problem (use recursion).
- Smaller and smaller subproblems are produced until the problem is small enough, which can be solved without splitting.

Algorithm D&C(P)

if Small(P) return Solve(P)

else{

Divide P into smaller instances $P_1, P_2, \ldots P_k$

Apply D&C to each of these subproblems

return Combine(D&C(P_1*), D&C(*P_2*), ...,D&C(*P_k*))*

}

- *Small(P)* is a Boolean valued function that determines whether the problem is small enough.
- If yes, the function *Solve(P)* is invoked.
- otherwise, each of subproblems is solved by *D&C* algorithm.
- *Combine* is a function that determines the solution of *P* using the solutions to *k* sub-problems.

Matrix multiplication

Strassen Matrix multiplication

Convex hull

Master theorem

Divide and Conquer

max-min

Selection

Multiplication of two integers

Multiplication of two binary integers

- For 12*9=108, if in binary arithmetic $1100_2 * 1001_2 = 1101100_2$
- For two n-bit numbers a and b, standard method requires $O(n^2)$ bit steps
- High-school approach to binary number multiplication result is at most $2n$ bits long, n^2 bits multiples, $2n$-bit additions.

$$
\begin{array}{r}
1100 \\
\underline{1001} \\
1100 \\
00000 \\
000000 \\
\underline{1100000} \\
1101100
\end{array}
$$

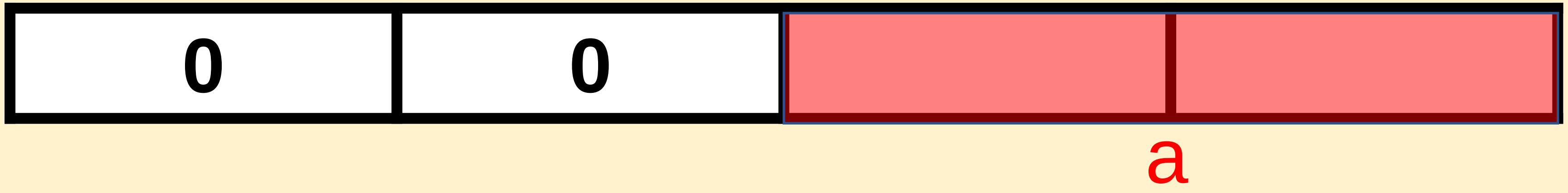
- Observe that a n-bit number can be expressed as

$$a * 2^{n/2} + b$$

e.g. $9 = 1001_2 = 1000_2 + 0001_2$

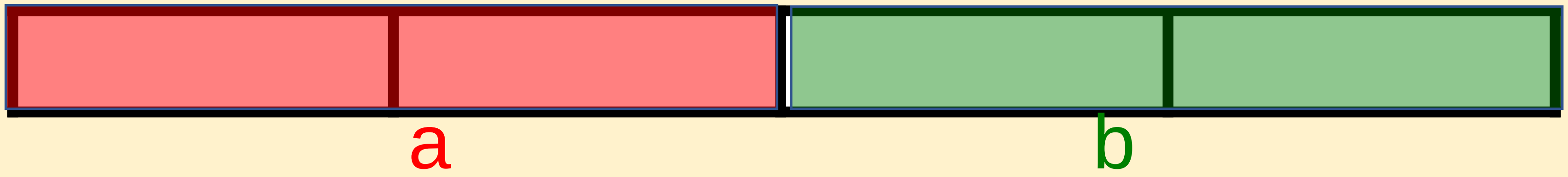
$$= 10_2 * 2^2 + 01_2$$

- To obtain a, shift right by 2 digits

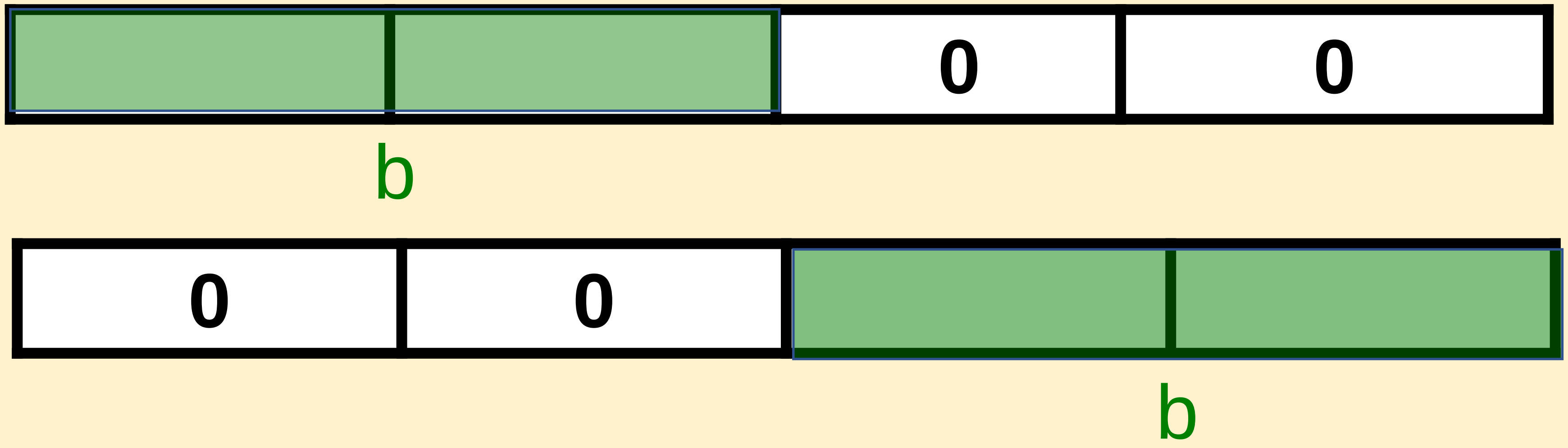


- Since shift right one digit is divided by 2

$a * 2^{n/2} + b$



- To obtain b shift left by 2 digits, then shift right by two digits



- Since shift left one digit is to multiply by 2

- Also note the product *uv* for

$$u = a * 2^{\frac{n}{2}} + b$$

$$v = c * 2^{\frac{n}{2}} + d$$

is given by

$$uv = \left(a * 2^{\frac{n}{2}} + b\right)\left(c * 2^{\frac{n}{2}} + d\right)$$

$$= ac * 2^n + (ad + bc)\, 2^{\frac{n}{2}} + bd$$

- Hence the problem is reduced to solving four $\frac{n}{2}$ bit multiplication instances.

$$T(n) = 4T\left(\frac{n}{2}\right) + cn$$

- But also observe that

$$ad+bc = (a+b)(c+d) - ac - bd$$

- Thus $(ad+bc)$ can be computed with one multiply, 2 additions, and 2 subtractions
  (because ac and bd are already known)
- The cost of addition/subtraction is O(n), hence

$$T(n) = 3T\left(\frac{n}{2}\right) + cn \quad \text{if } \ n > 1$$

- Time complexity is

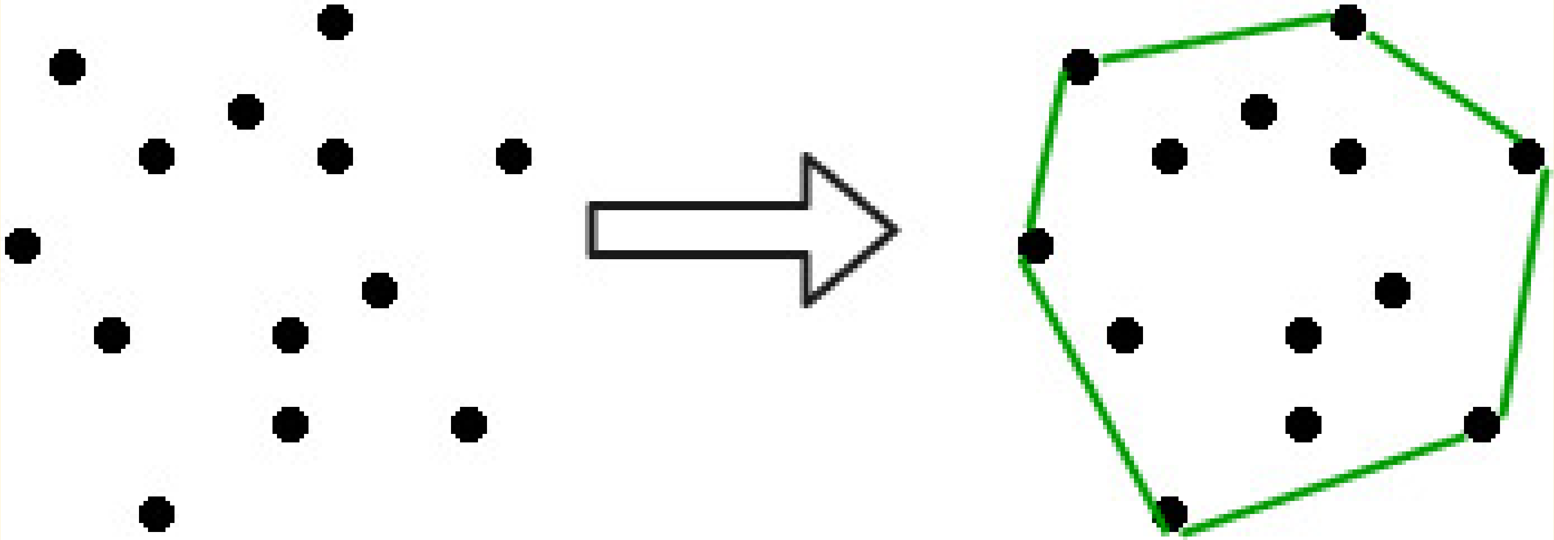
$$O\left(n^{\log 3}\right) = O\left(n^{1.58}\right)$$

- The algorithm solve n-bit multiplication between u *and* v by D&C
- Small(P), direct solve
- shiftleft(x,y) moves x by y bits to the left
- shiftright(x,y) moves x by y bits to the right
- *Recursively solve, and combine solution*

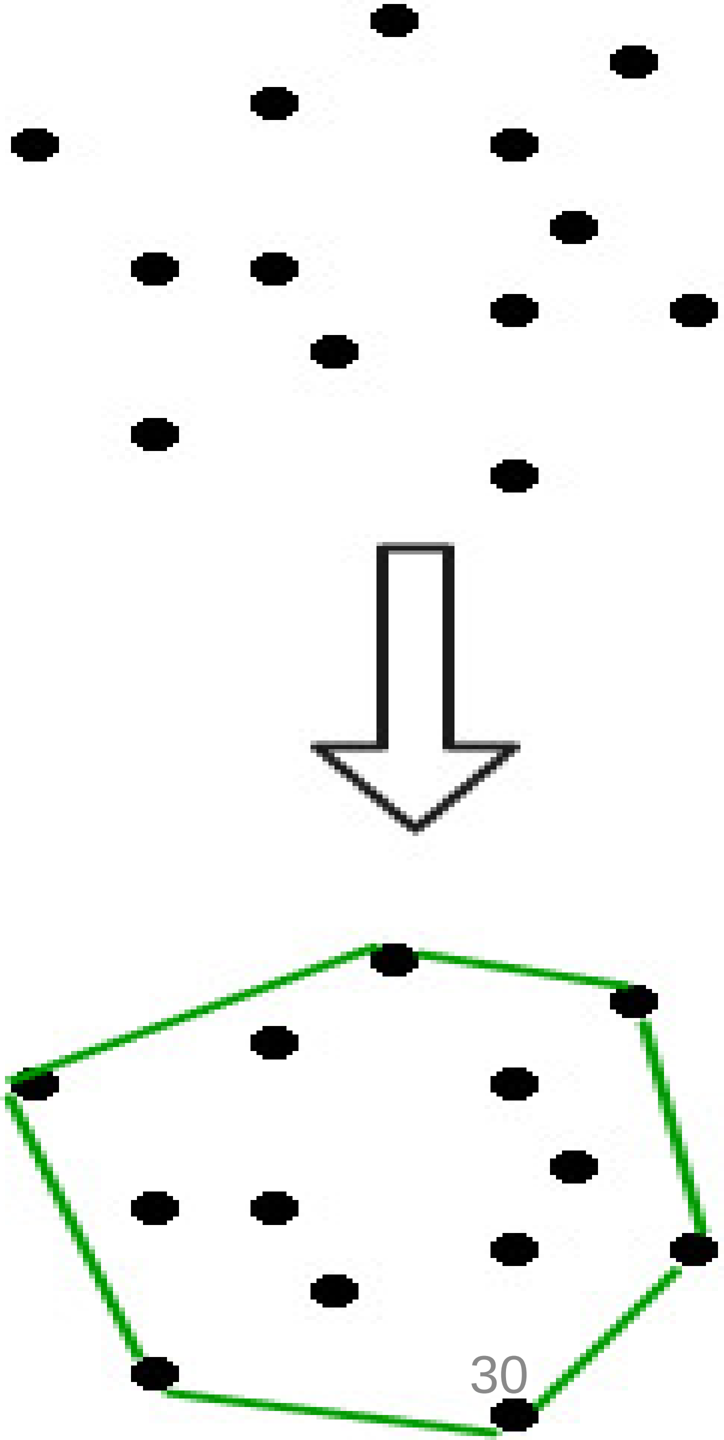
```
Algorithm multiply(u, v, word; n: integer)
{ if(n== 1) return(uv);
    else{
    p=n;
    m=n/2;
    a=shiftright(u, m);
    b=shiftright((shiftleft(u; m); m);
    c=shiftright(v; m);
    d=shiftright((shiftleft(v; m); m);
    r1 =multiply(a, c, n/2);
    r2 =multiply(b, d, n/2);
    r3 =multiply(a, d, n/2);
    r4 =multiply(b, c, n/2);
    return shiftleft(r1, p)+shiftleft(r3 +r4, m) +r2;
}
```

Convex Hull

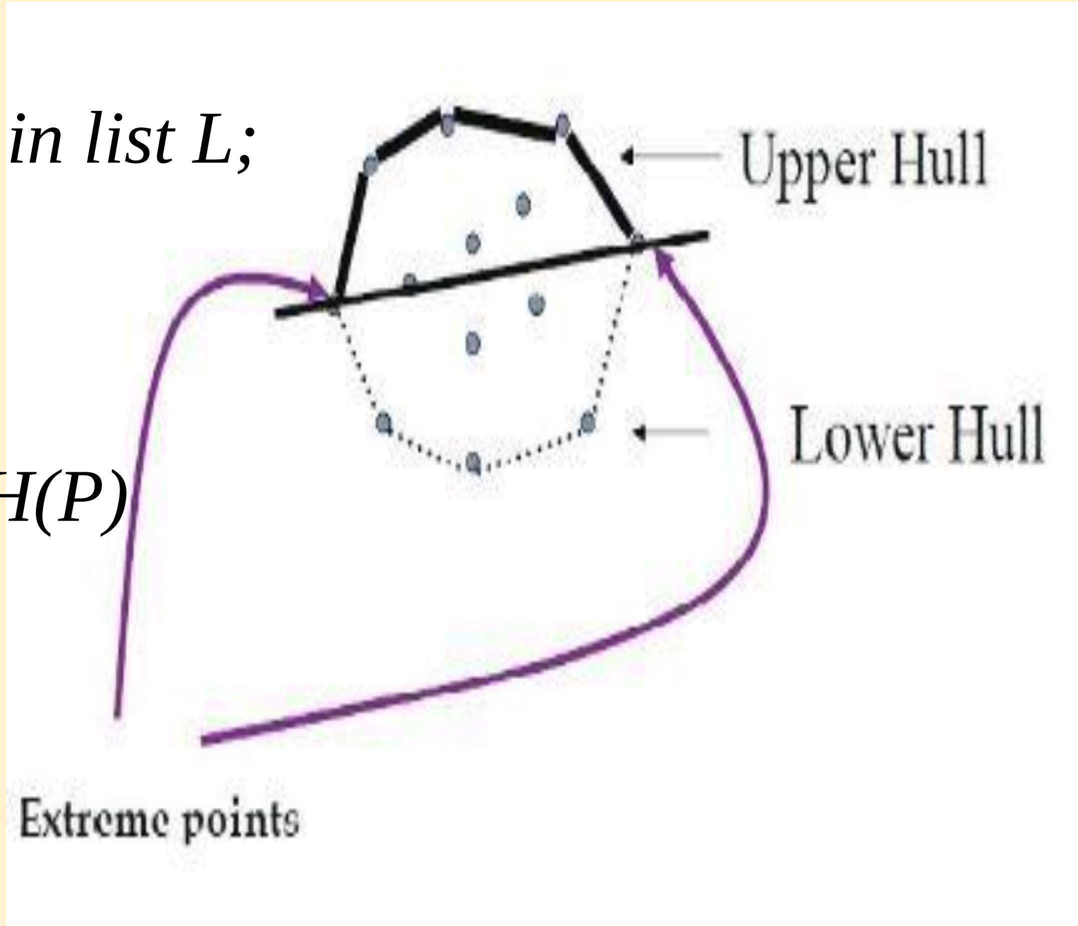
- **Definition:** The smallest boundary of a convex set S of points in the plane is defined to be the smallest convex polygon containing all the points of S.



- Applications:
  - Computer graphics (creating images and modeling scenes, creating realistic looking scenes taking light)
  - Robotics (design and use of robots, operating in a real 3-D world)
  - Geographic information systems (storage of physical systems such as mountains, vegetation, populations, roads)
  - CAD/CAM (design and manufacture of products)

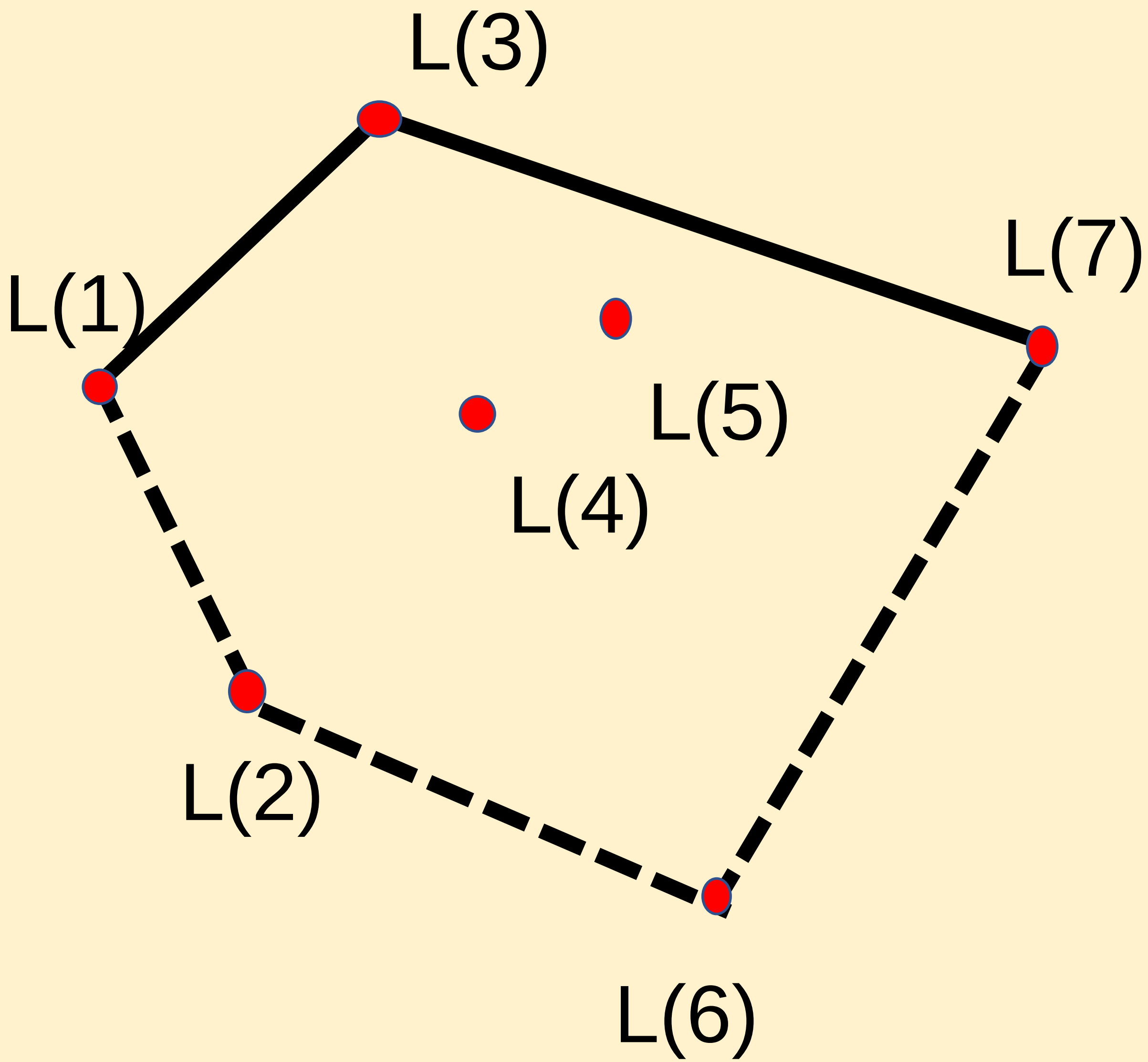


Algorithm ConvexHull (P: set of points)
{
Sort the points by the x coordinate and place in list L;
UpperHull(L);
LowerHull(L);
Remove the first and last point from Llower
Return the union of Llower and Lupper as CH(P)
}



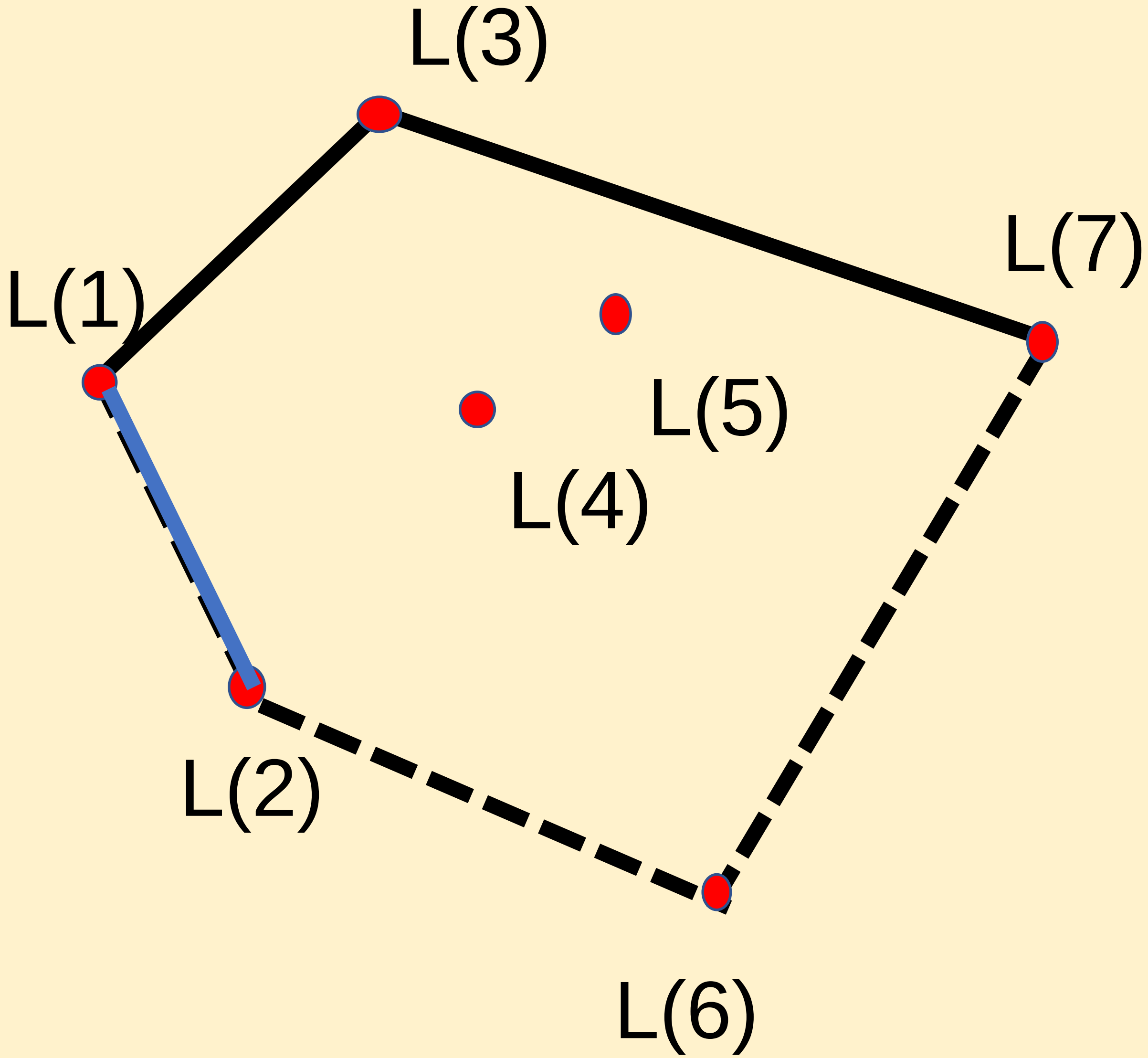
- *The Algorithm constructs a convex hull*
- *Input: a set of points from 2D*
- *Output: a set of CH(P) the vertices convex hull in clockwise order*
- *Lupper is the list for upper hull*
- *Llower is the list for lower hull*
- *Combine the two hulls*

Example



Algorithm UpperHull(P: set of points)
{
Put points L(1) and L(2) into a list Lupper;
for(i= 3;i <=n;i+ +){add L(i) to Lupper
while Lupper contains more than two points AND the last three points do not make right turn {
Delete the middle of the last three points from Lupper}
}
return Lupper;
}

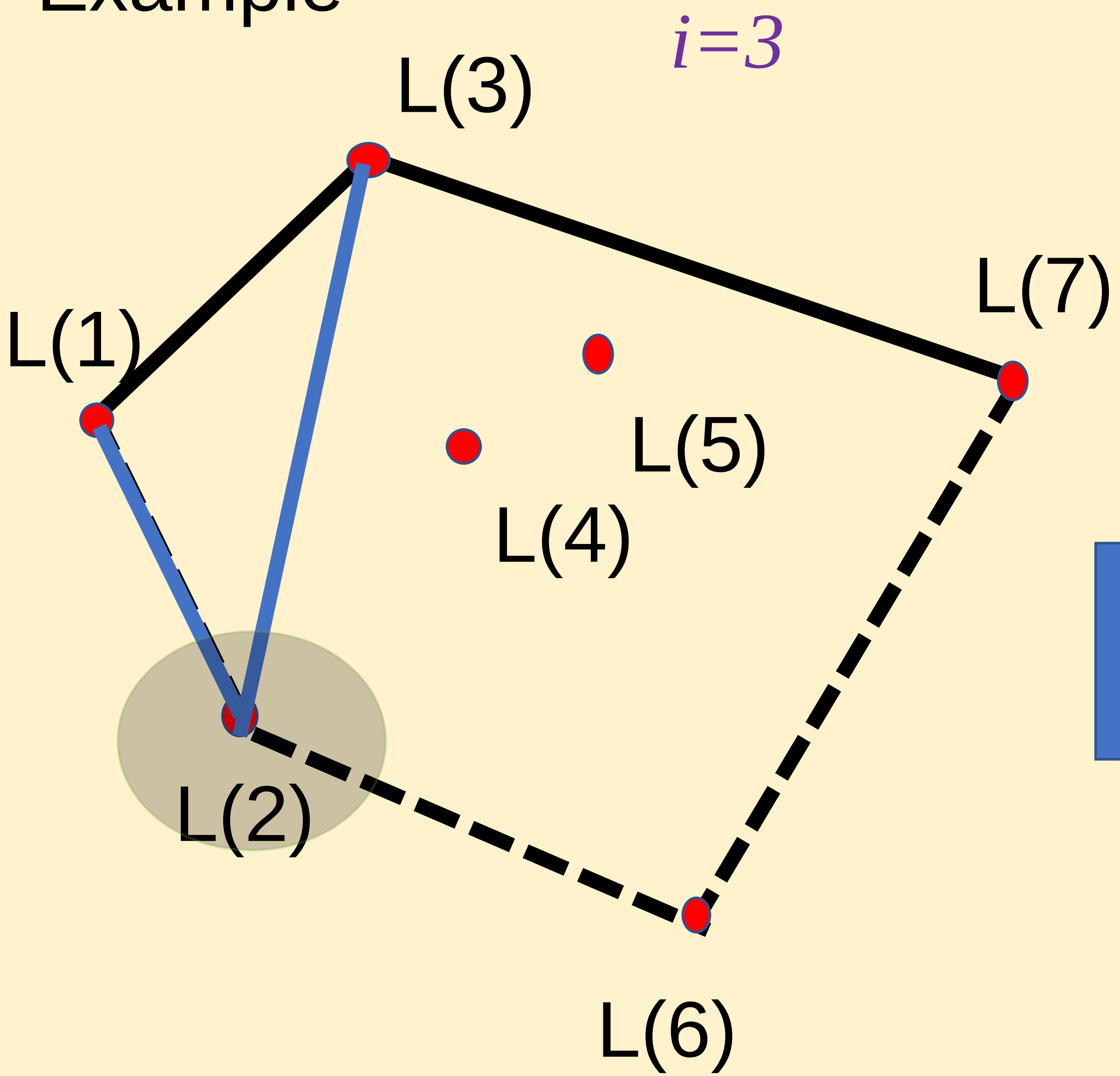
Example



Algorithm UpperHull(P: set of points)
{
Put points L(1) and L(2) into a list Lupper;

Lupper:
- L(1), L(2)

Example



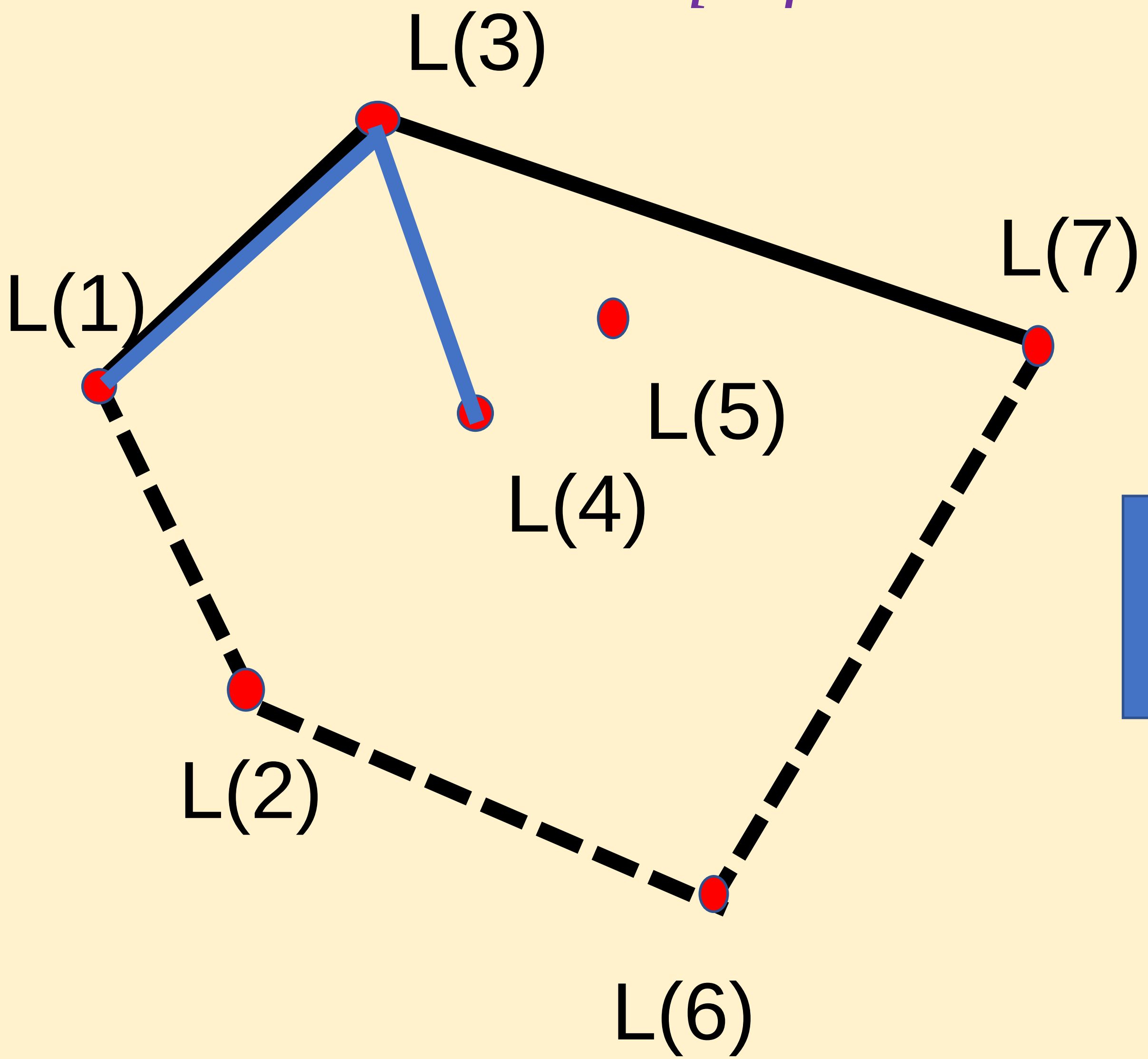
Algorithm UpperHull(P: set of points)
{
Put points L(1) and L(2) into a list Lupper;
for(i= 3;i <=n;i+ +){add L(i) to Lupper
while Lupper contains more than two points AND the last three points do not make right turn {
Delete the middle of the last three points from Lupper}

Lupper:

- L(1), L(2)
- L(1), ~~L(2)~~, L(3)

Example

i=4

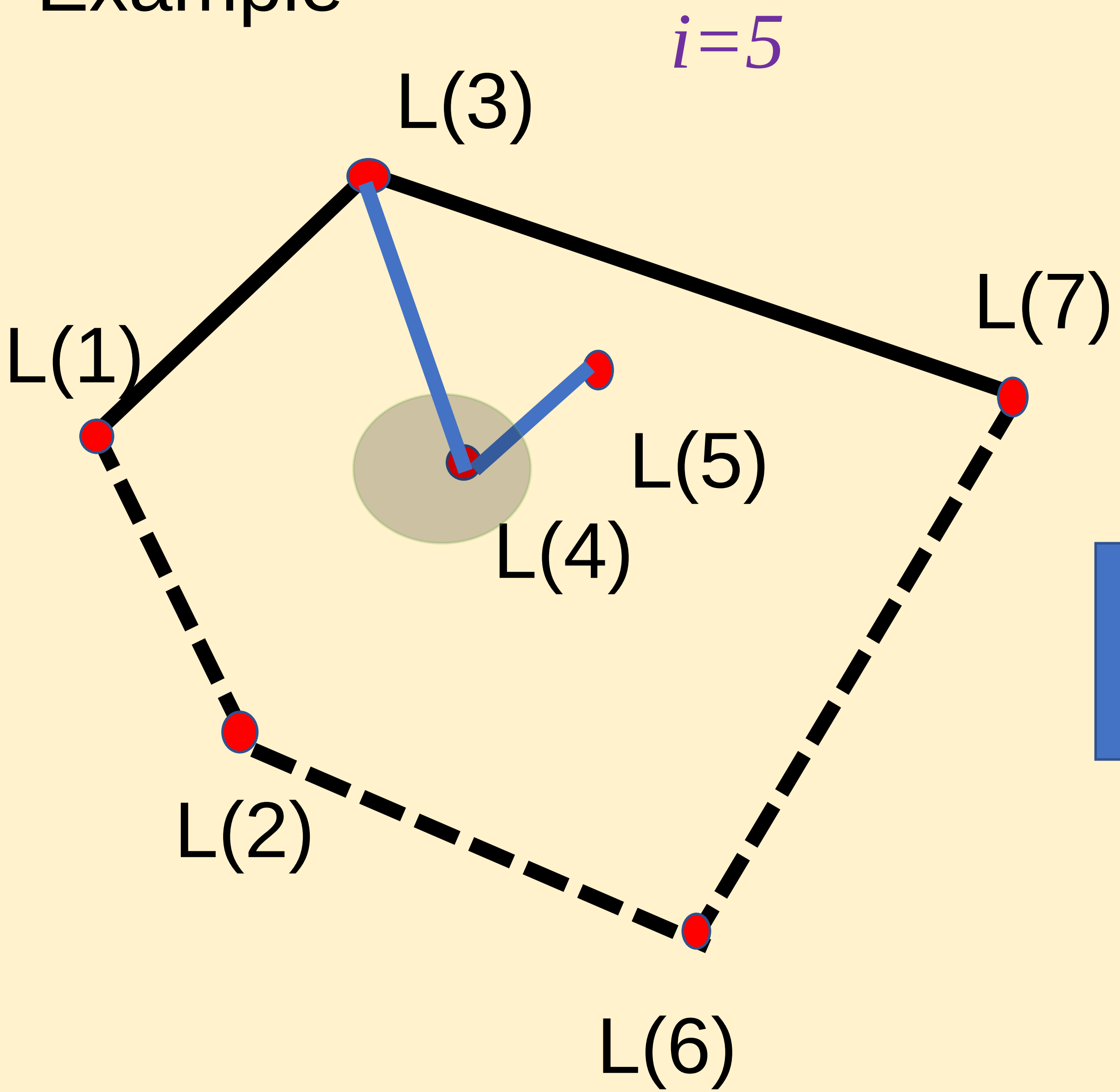


Algorithm UpperHull(P: set of points)
{
Put points L(1) and L(2) into a list Lupper;
for(i= 3;i <=n;i+ +){add L(i) to Lupper
while Lupper contains more than two points AND the last three points do not make right turn {
Delete the middle of the last three points from Lupper}

Lupper:

- L(1), L(3)
- L(1), L(3), L(4)

Example

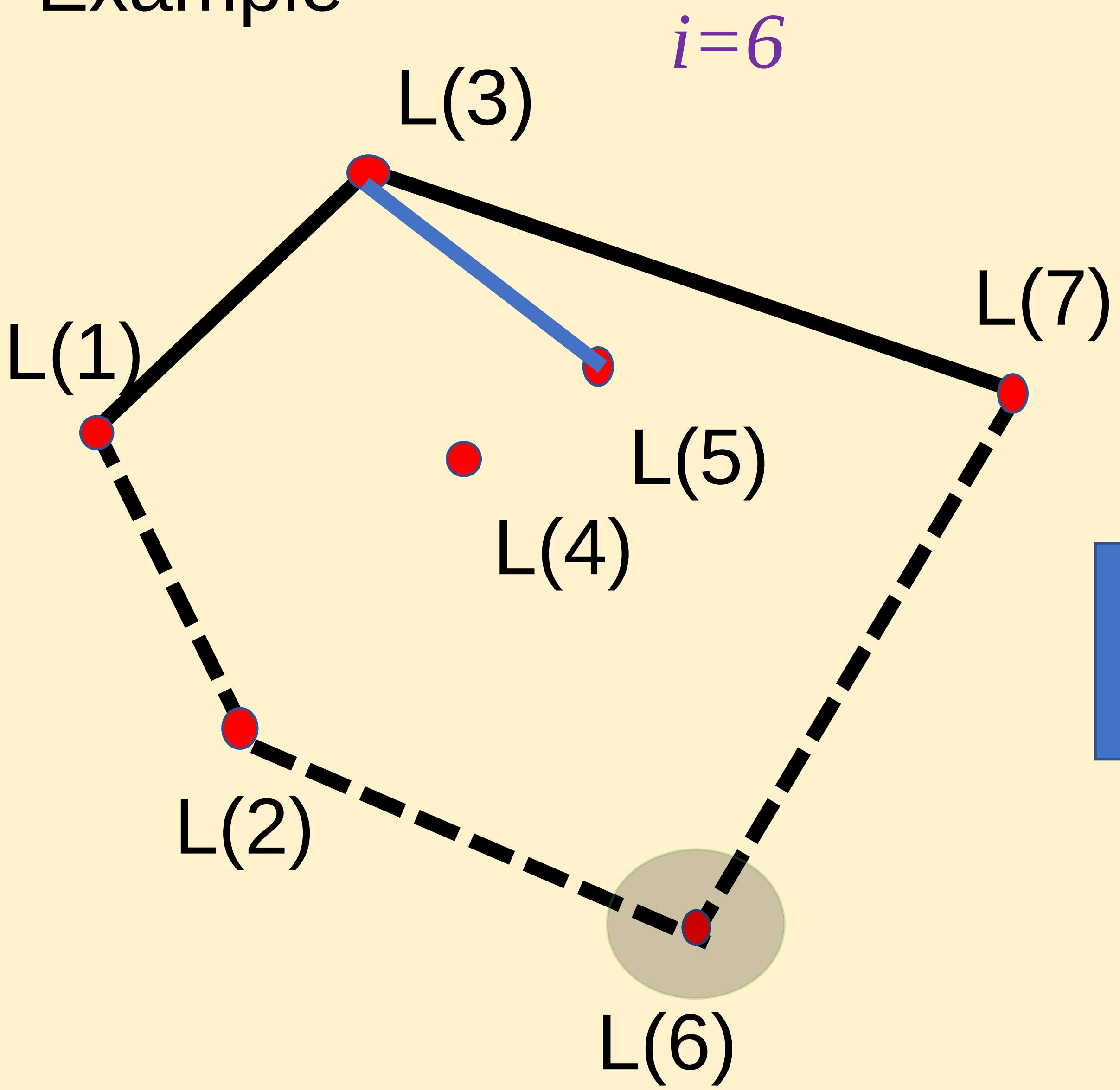


Algorithm UpperHull(P: set of points)
{
Put points L(1) and L(2) into a list Lupper;
for(i= 3;i <=n;i+ +){add L(i) to Lupper
while Lupper contains more than two points AND the last three points do not make right turn {
Delete the middle of the last three points from Lupper}

Lupper:

- L(1), L(3), L(4)
- L(1), L(3), ~~L(4)~~, L(5)

Example

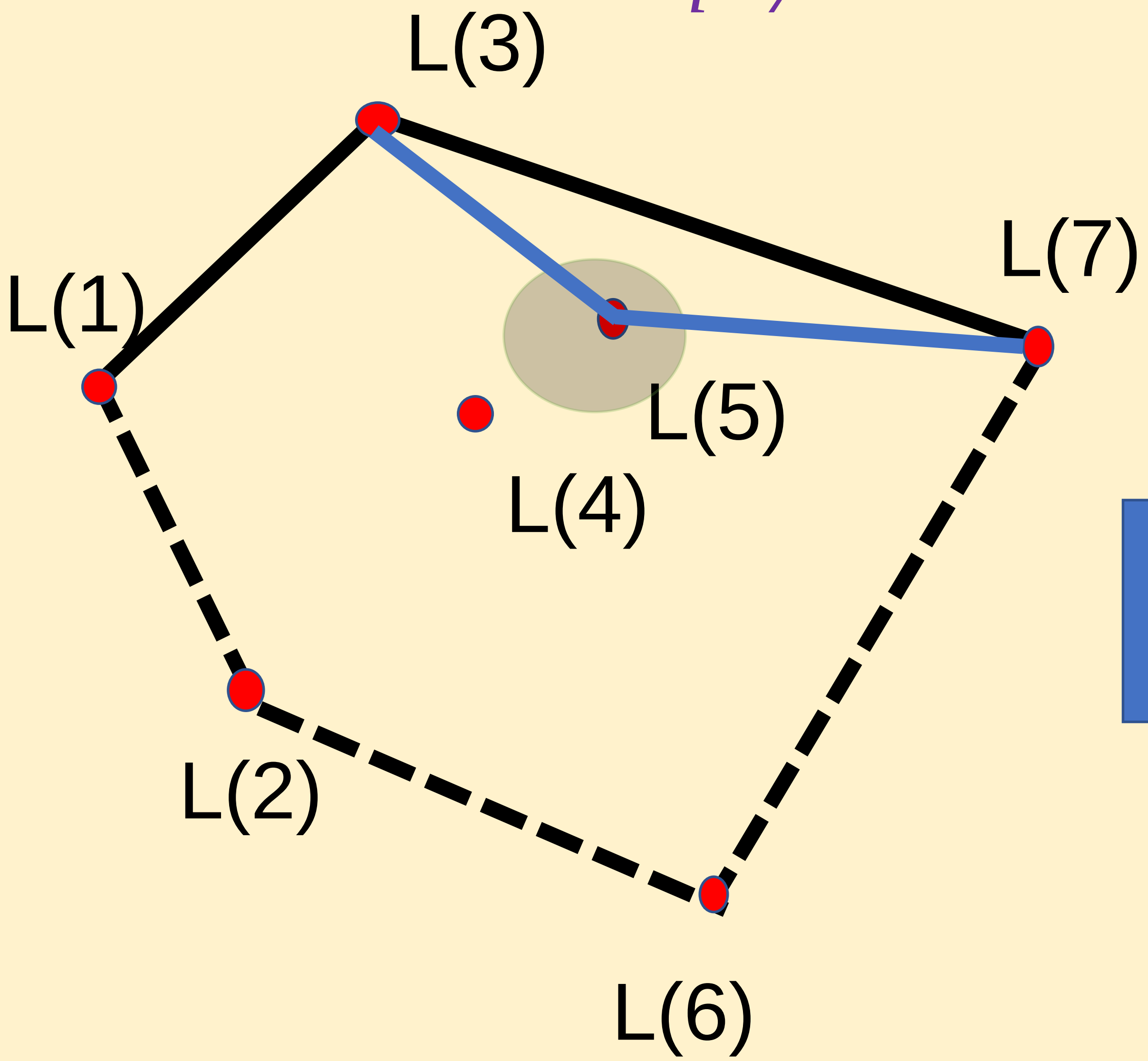


Algorithm UpperHull(P: set of points)
{
Put points L(1) and L(2) into a list Lupper;
for(i= 3;i <=n;i+ +){add L(i) to Lupper
while Lupper contains more than two points AND the last three points do not make right turn {
Delete the middle of the last three points from Lupper}

Lupper:

- L(1), L(3), L(5)
- L(1), L(3), L(5)

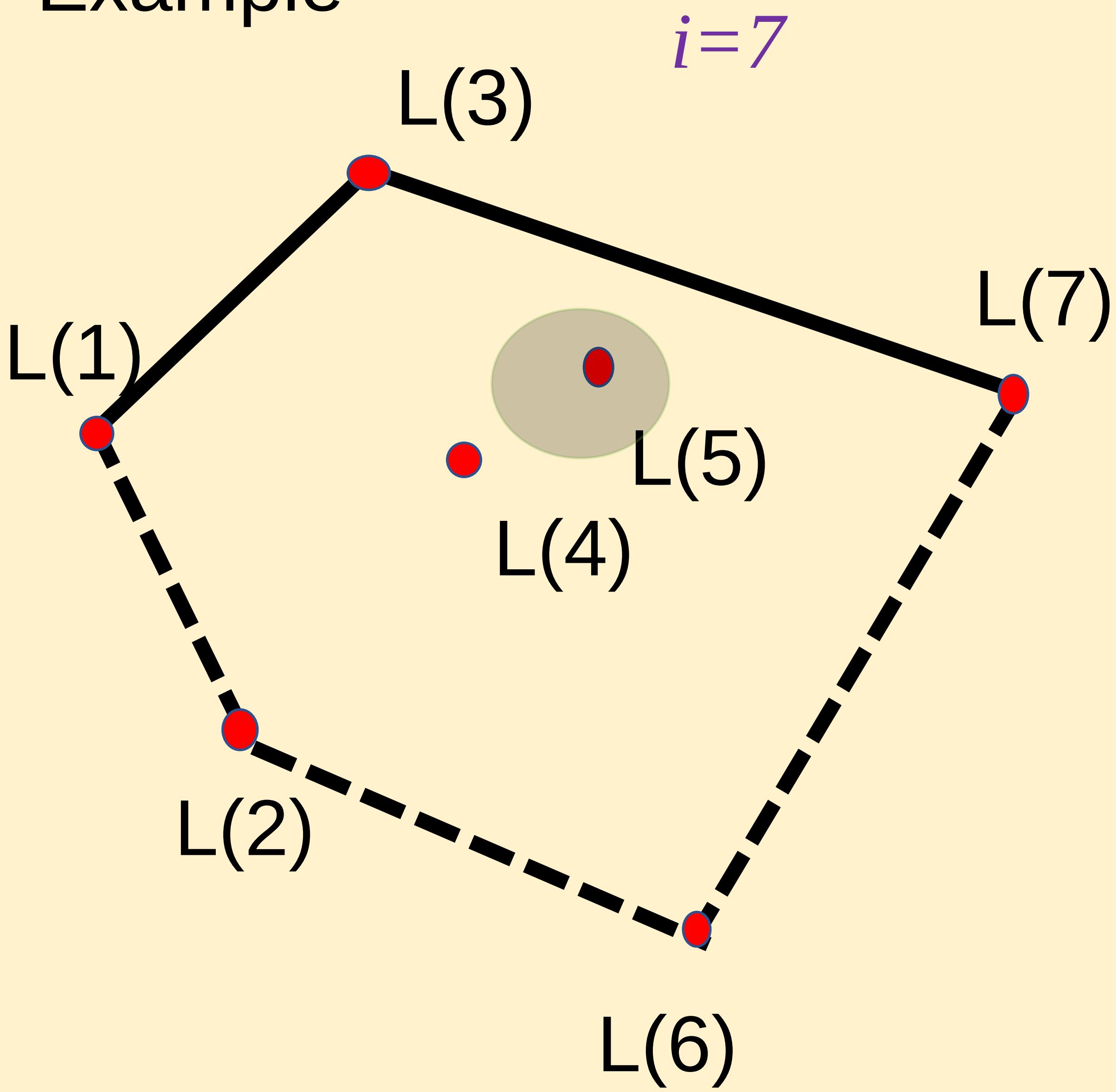
Example



Algorithm UpperHull(P: set of points)
{
Put points L(1) and L(2) into a list Lupper;
for(i= 3;i <=n;i+ +){add L(i) to Lupper
while Lupper contains more than two points AND the last three points do not make right turn {
Delete the middle of the last three points from Lupper}

Lupper:
- L(1), L(3), L(5)
- L(1), L(3), ~~L(5)~~,L(7)

Example

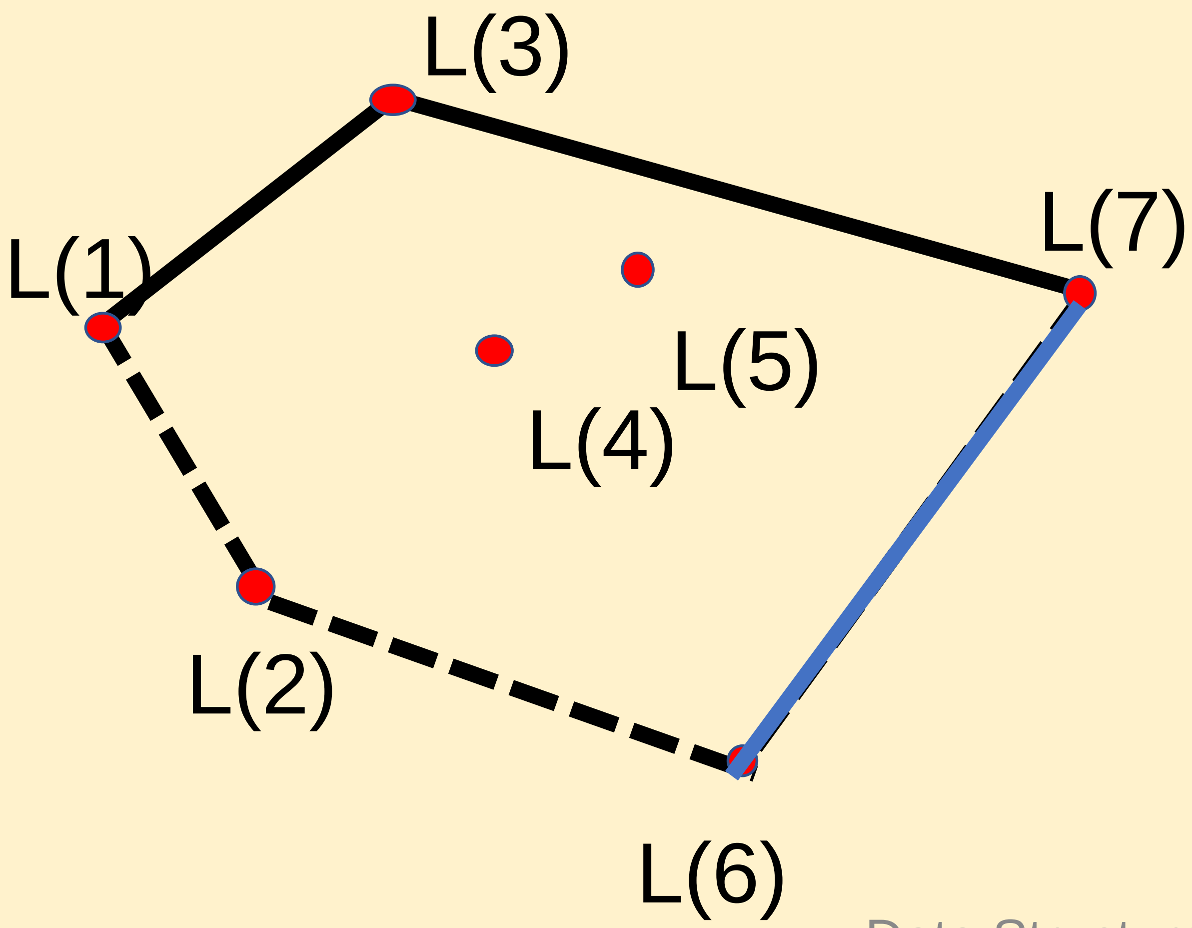


Algorithm UpperHull(P: set of points)
{
Put points L(1) and L(2) into a list Lupper;
for(i= 3;i <=n;i+ +){add L(i) to Lupper
while Lupper contains more than two points AND the last three points do not make right turn {
Delete the middle of the last three points from Lupper}
}
return Lupper;
}

Lupper:

- L(1), L(3), L(7)

In class exercise: Find the sequence of Lower hull

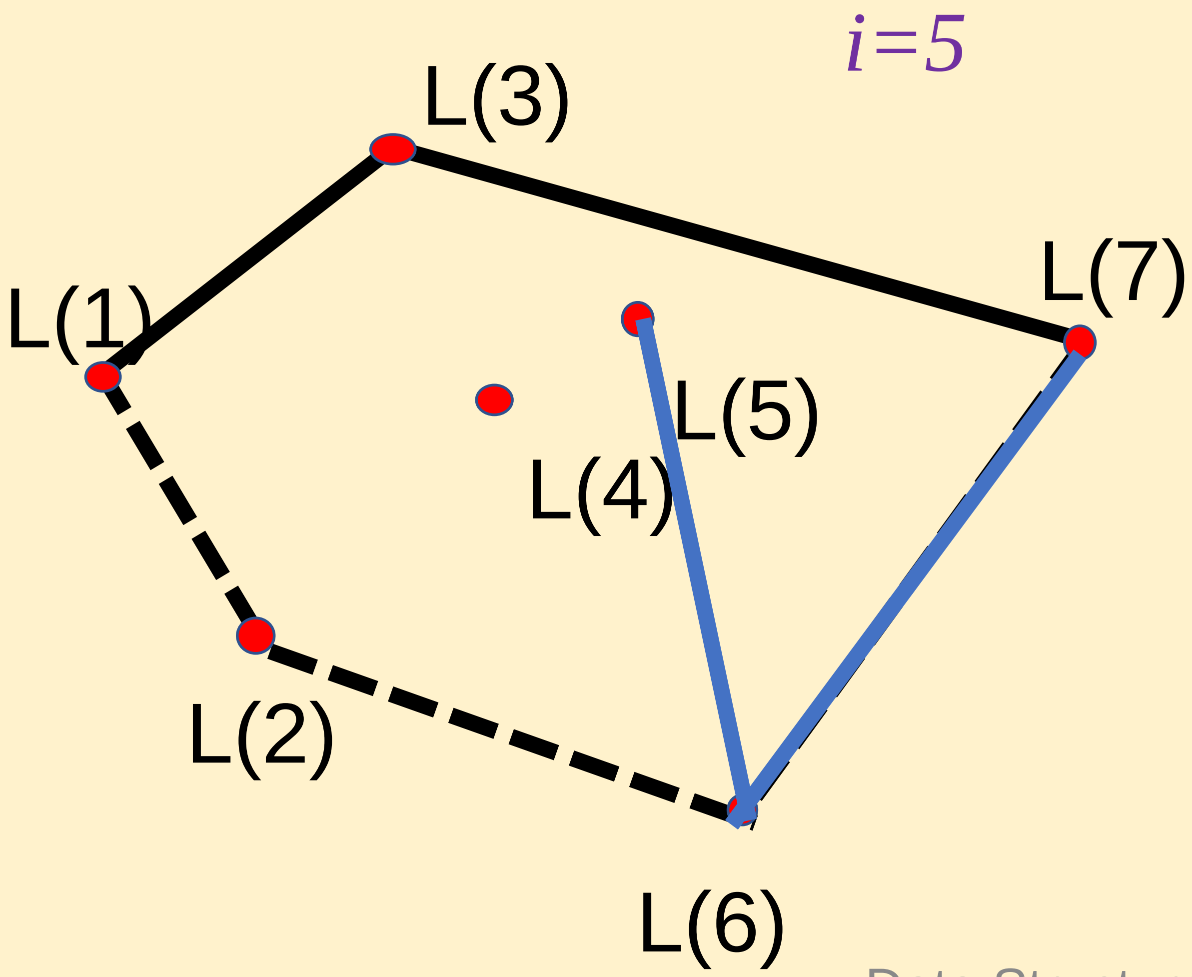


Algorithm LowerHull(P: set of points)
{
Put points L(n) and L(n-1) into a list Llower;
for(i= n-2;i >=1;i- -){add L(i) to Llower
while Llower contains more than two points AND the last three points do not make right turn {
Delete the middle of the last three points from Llower}
}
return Llower;
}

Llower:

- L(7), L(6)

In class exercise: Find the sequence of Lower hull



Algorithm LowerHull(P: set of points)
{
Put points L(n) and L(n-1) into a list Llower;
for(i= n-2;i >=1;i- -){add L(i) to Llower
while Llower contains more than two points AND the last three points do not make right turn {
Delete the middle of the last three points from Llower}
}
return Llower;
}

Llower:

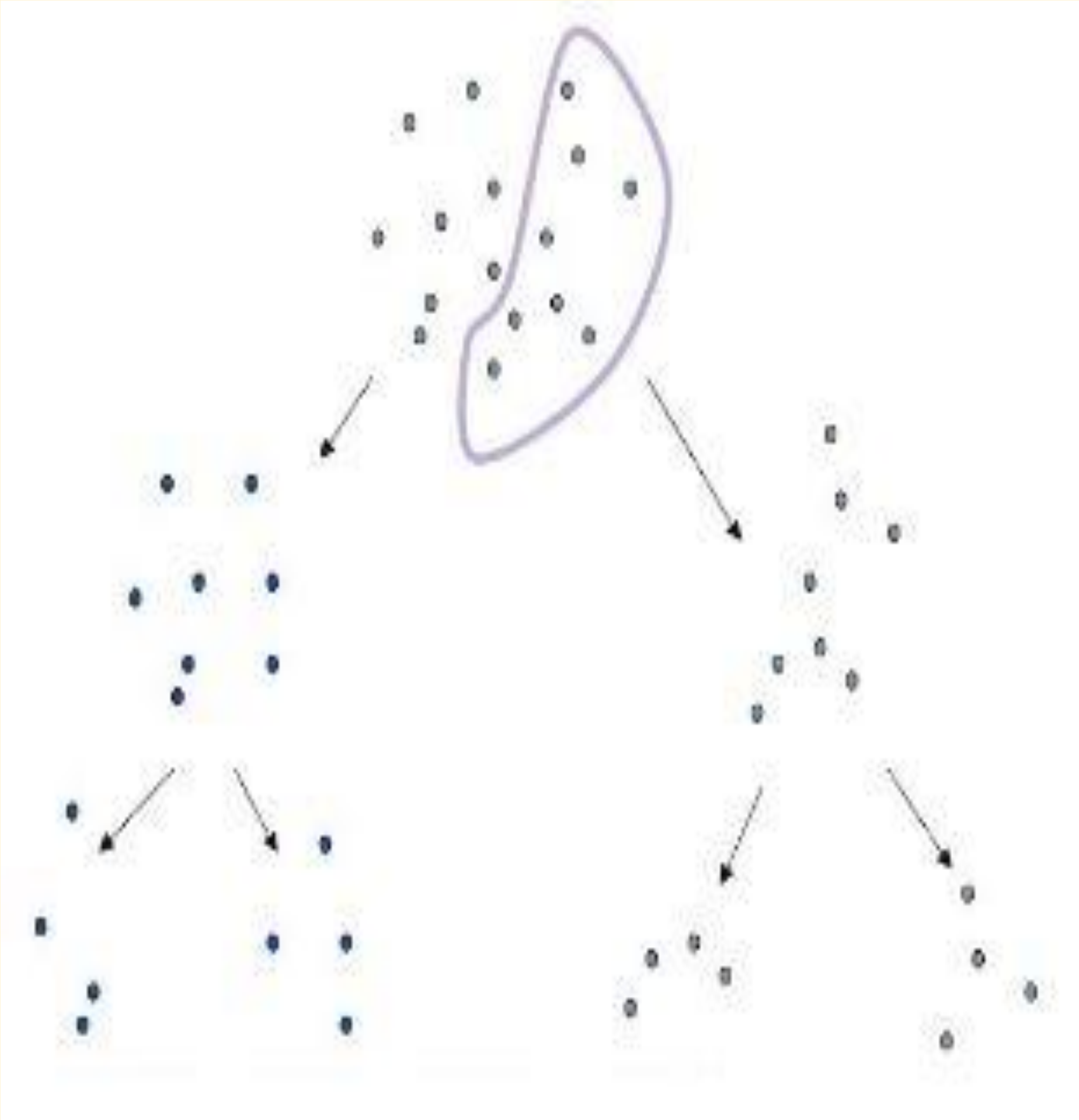
- L(7), L(6)
- L(7), L(6),L(5)

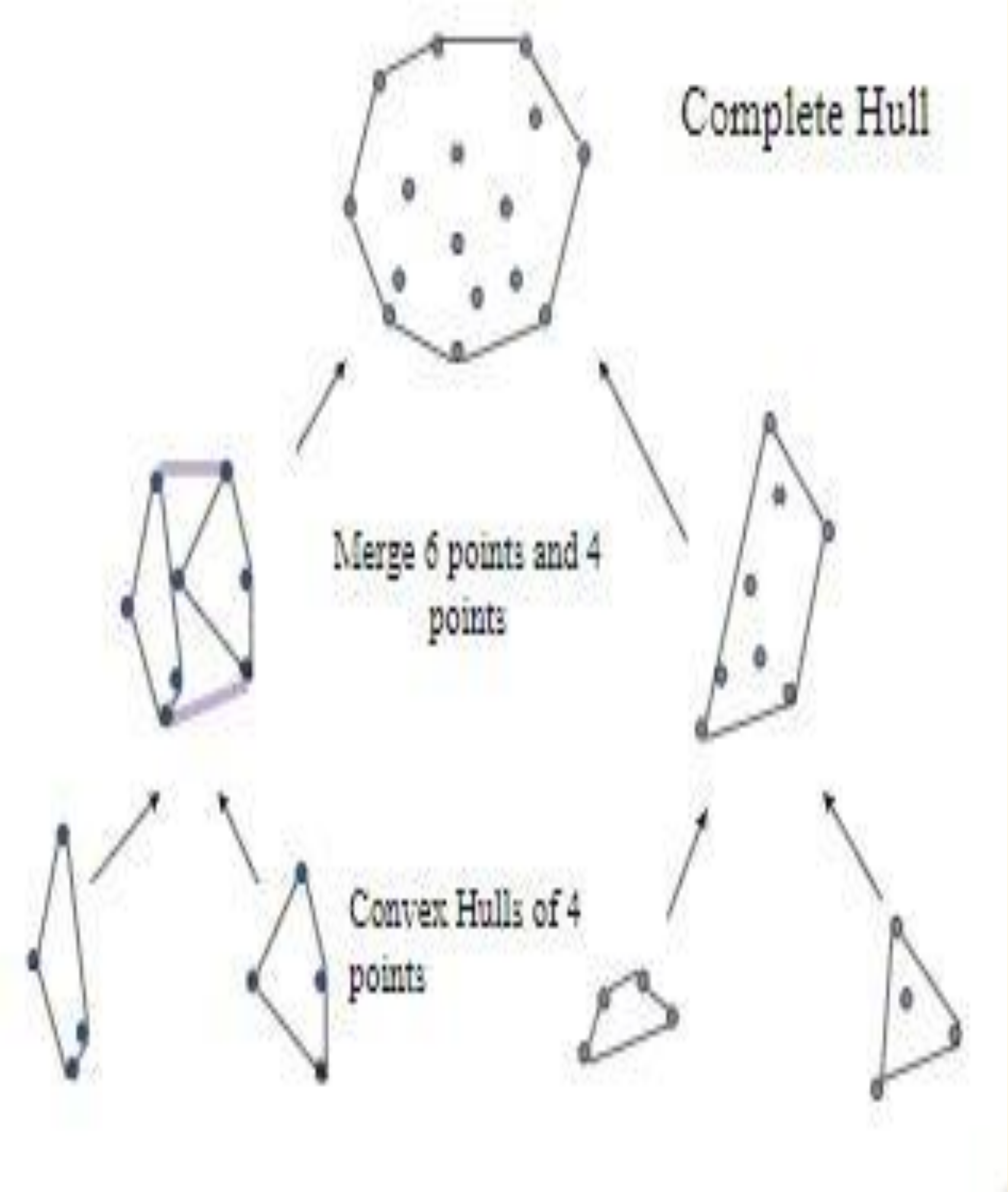
D&C convex hull:

```
Algorithm DCHull(P:set of points)
{
    if(size(P)< minsize
        return  ConvexHull(P)
  else{
      split(P, L1, L2 );
     \\divide P into two Lists L1, L2
    return Merge(DCHull(L1);  DCHull(L2 ))
   }
}
```

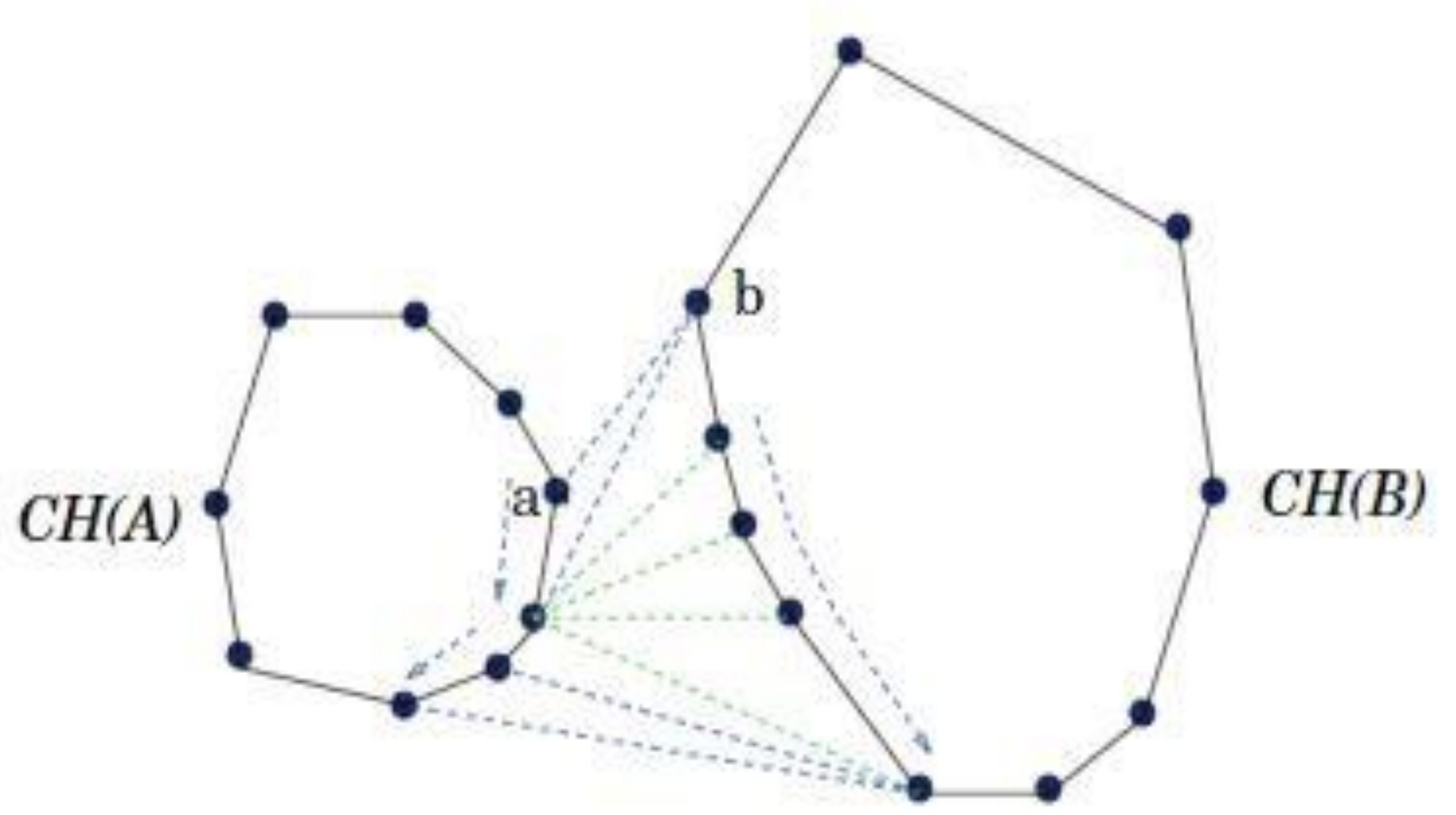
- Eliminate as many point as possible before doing a hull.
- How the list is split is crucial to efficiency.
- Merge: throws away points inside the two hulls form L_1, L_2

Let minsize = 4

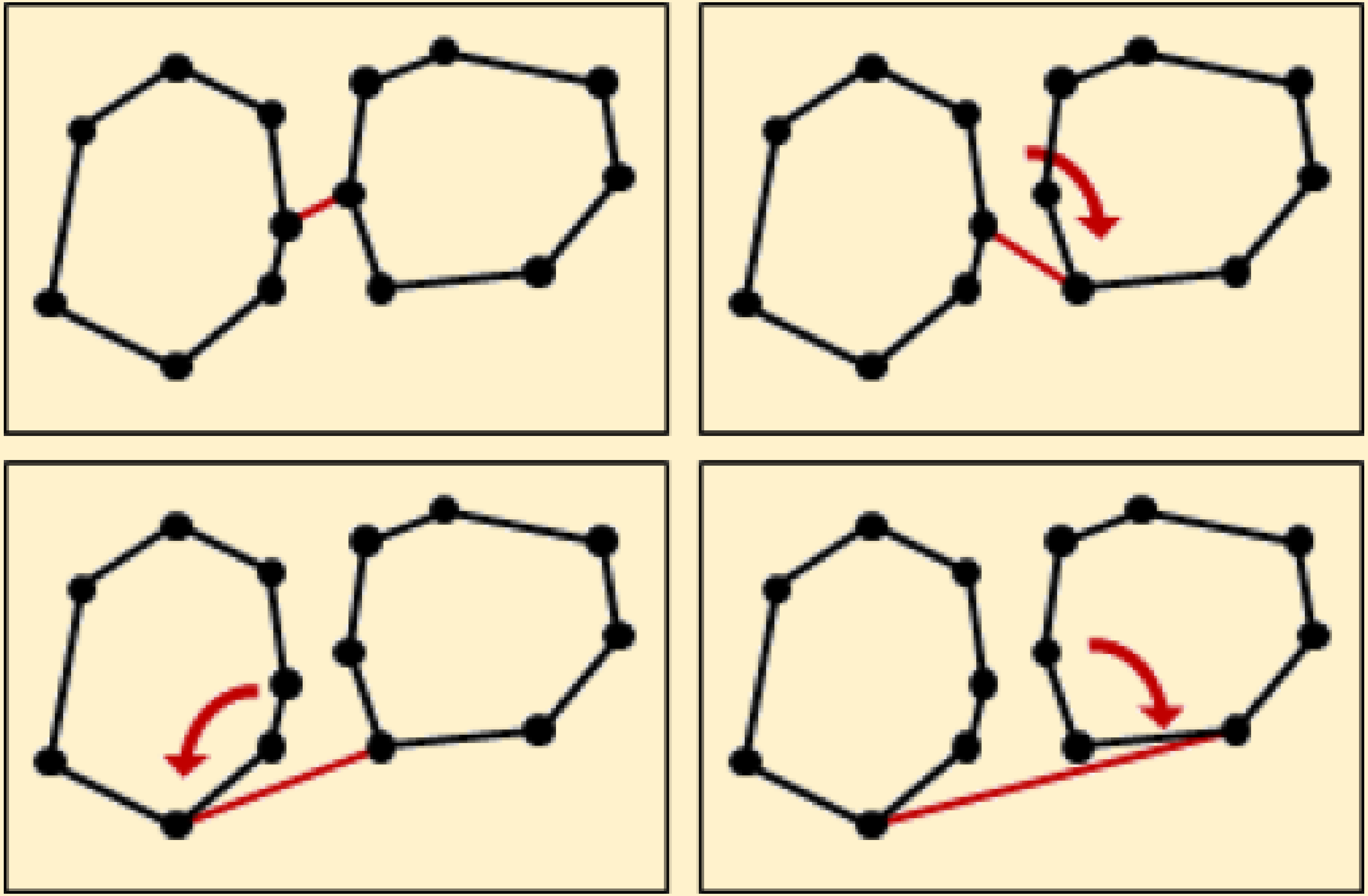




- To merge two convex hulls, the lower and higher tangent lines are found.



- All other points are on one side of the tangent lines.
- For example, search the lower tangent line, move a clockwise and b counter clockwise, until ab is lower tangent.
- Points between two tangent lines are dropped.



Complexity of DCHull

- Dividing the points into left and right half each containing $\frac{n}{2}$ points
    - Sort the points ($O(n\log n)$)
    - take the median

- Find the convex hull of each recursively and stich the hulls together
    - $T(n) = 2T(\frac{n}{2}) + cn$
    - $O(n\log n)$

D&C complexity summary

Problem	Algorithm	Time Complexity
Max-min	Direct	O(n)
	D&C	O(n)
Selection	Selection	$O(n^2)$
	Fast	O(n)
Convex Hull	Incremental	$O(n^2)$
	DCHull	O(nlogn)
Matrix Multiplication	D&C	$O(n^3)$
	Strassen's	$O(n^{2.81})$

Matrix multiplication

Strassen Matrix multiplication

Convex hull

Master theorem

Divide and Conquer

max-min

Selection

Multiplication of two integers