MATRIX MULTIPLICATION 420 - 422

- Let $A$ and $B$ be two $n \times n$ matrices ($n = 2^k$)
- Let $C = A \times B$,
- i.e., $C_{i,j} = \sum_{k=1}^{n} a_{i,k}b_{k,j}$ .... takes $O(n^3)$ time.

Partition matrices into submatrices of size $\frac{n}{2} \times \frac{n}{2}$

\[
\begin{bmatrix}
  C_{1,1} & C_{1,2} \\
  C_{2,1} & C_{2,2}
\end{bmatrix} \leftarrow \begin{bmatrix}
  A_{1,1} & A_{1,2} \\
  A_{2,1} & A_{2,2}
\end{bmatrix} \begin{bmatrix}
  B_{1,1} & B_{1,2} \\
  B_{2,1} & B_{2,2}
\end{bmatrix}
\]

where,

\[
\begin{align*}
  C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\
  C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\
  C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\
  C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2}
\end{align*}
\]
Time Complexity is given by $T(n)$ as follows

$$T(n) = \begin{cases} 
  m + a, & n = 2 \\
  mT(n/2) + a(n^2/4), & n > 2 
\end{cases}$$

Solution is $O(n^{\log_2 m})$, where
- $m$: # of $n/2 \times n/2$ matrix multiplications, and
- $a$: # of $n/2 \times n/2$ matrix additions.

Solution is $O(n^{\log_2 m})$

In this case $m = 8$ and $a = 4$, so the time complexity bound is $O(n^3)$
\[ M_1 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2}) \]
\[ M_2 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \]
\[ M_3 = (A_{1,1} - A_{2,1})(B_{1,1} + B_{1,2}) \]
\[ M_4 = (A_{1,1} + A_{1,2})B_{2,2} \]
\[ M_5 = A_{1,1}(B_{1,2} - B_{2,2}) \]
\[ M_6 = A_{2,2}(B_{2,1} - B_{1,1}) \]
\[ M_7 = (A_{2,1} + A_{2,2})B_{1,1} \]

\[ C_{1,1} = M_1 + M_2 - M_4 + M_6 \]
\[ C_{1,2} = M_4 + M_5 \]
\[ C_{2,1} = M_6 + M_7 \]
\[ C_{2,2} = M_2 - M_3 + M_5 - M_7 \]
Mathematica: ECI or CSIL (math)

\[ In[1] := \quad M1 = (A_{12} - A_{22})(B_{21} + B_{22}) \]
\[ In[2] := \quad M2 = (A_{11} + A_{22})(B_{11} + B_{22}) \]

... ...

\[ In[8] := \quad \text{Simplify}[C_{11} = M1 + M2 - M4 + M6] \]
\[ Out[8] = \quad A_{11} \, B_{11} + A_{12} \, B_{21} \]

\[ In[9] := \quad \text{Simplify}[C_{12} = M4 + M5] \]
\[ Out[9] = \quad A_{11} \, B_{12} + A_{12} \, B_{22} \]

\[ In[10] := \quad \text{Simplify}[C_{21} = M6 + M7] \]
\[ Out[10] = \quad A_{21} \, B_{11} + A_{22} \, B_{21} \]

\[ In[11] := \quad \text{Simplify}[C_{22} = M2 - M3 + M5 - M7] \]
\[ Out[11] = \quad A_{21} \, B_{12} + A_{22} \, B_{22} \]
\[
T(n) = \begin{cases} 
7 + 18, & n = 2 \\
7T(n/2) + 18(n^2/4), & n > 2 
\end{cases}
\]

So the solution is \(O(n^{\log_2 7}) = O(n^{2.81})\). Faster than classical algorithm when \(n \geq 16\). If \(n\) is not a power of two then same bound holds by using standard techniques.


Hopcroft and Kerr multiplying matrices of size 2 by 2 require 7 multiplication. But bound is not known for matrices of size 3 by 3, or higher.

- Victor Pan \(O(n^{2.79})\)
- Bini et. al. ... \(O(n^{2.77})\) ...
- ... Victor Pan ... \(O(n^{2.5})\)
- Coppersmith and Winograd ... \(O(n^{2.376})\)
• Coppersmith and Winograd ... \( O(n^{2.375477}) \)
• Stothers ... \( O(n^{2.373}) \)
• Williams ... \( O(n^{2.3728642}) \)
• Le Gal ... \( O(n^{2.3728639}) \)

• Cohn, Klienberg, Szegedy and Umans show that improvements via two different conjectures would imply an \( O(n^2) \) algorithm, but show this would be incompatible with the sunflower conjecture.

• Raz showed an \( \Omega(n^2 \log n) \) for a restricted model of computation.
Triangular MM (notes.2)

- $T(n)$: Time complexity to multiply two $n \times n$ triangular matrices.
- $M(n)$: Time complexity to multiply two $n \times n$ matrices.
- Is $T(n) = \Omega(M(n))$?
- Answer is YES, because any algorithm that multiplies triangular matrices of size $3n \times 3n$ can be used to multiply matrices of size $n \times n$.

$$
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & A & 0
\end{array} \quad \times \quad 
\begin{array}{ccc}
0 & 0 & 0 \\
B & 0 & 0 \\
0 & 0 & 0
\end{array} \quad \rightarrow \quad 
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 0 \\
AB & 0 & 0
\end{array}
$$
Selecting $k^{th}$ Smallest Element Sec. 10.2.3

- Return $k^{th}$ smallest element in $a[0:n-1]$. Assume $a[n]$ is a dummy largest element and all elements distinct.

Method: Put first element in its correct position (wrt to ordering) (like in Quicksort) and then either you found the $k^{th}$ smallest element or it is to its left or it is to its right. In the latter two cases find the $k^{th}$ smallest element recursively.

```cpp
template<class T> T QSelect(T a[], int n, int k) {
    // Return k’th smallest element in a[0:n-1].
    // Assume a[n] is a dummy largest element.
    if (k < 1 || k > n) throw OutOfBounds();
    return qselect(a, 0, n-1, k); }
```
template<class T> T qselect(T a[], int l, int r, int k)
{ // Return k'th smallest in a[l:r].
    if (l >= r) return a[l];
    int i = l, // left to right cursor
        j = r + 1; // right to left cursor
    T pivot = a[l]; // Partition wrt pivot
    // swap elements >= pivot on left side
    // with elements <= pivot on right side
    while (true) {
        do { // find a[i] >= pivot on left side
                i = i + 1; } while (a[i] < pivot);
        do { // find a[j] <= pivot on right side
                j = j - 1; } while (a[j] > pivot);
        if (i >= j) break; // swap pair not found
        Swap(a[i], a[j]); // swap the pair
    }
    if (j - l + 1 == k) return pivot;
    a[l] = a[j]; a[j] = pivot;
    if (j - l + 1 < k)
        return qselect(a, j + 1, r, k - j + l - 1);
    else return qselect(a, l, j - 1, k); }
Worst case time complexity is $O(n^2)$. So fix by ...

**Find a Good Pivot**

- Partition into groups with $r$ ($r$ is odd, e.g. $r = 5$) elements (discard additional elements)
- Find the Median in each group
- Find Median of ($\lfloor n/r \rfloor$) Medians
- Use Median of Medians to Partition.

\[
\begin{array}{c}
\times \times \times \times \times \times \times \times \\
\times \times \times \times \times \times \times \times \\
\times \times \times \times \times \times \times \times \\
\times \times \times \times \times \times \times \times \\
\times \times \times \times \times \times \times \times \\
\end{array}
\]

- < median
- medians
- > median
- smaller than MM
- larger than MM

At least $\frac{r+1}{2} \frac{|n/r|}{2}$ elements are deleted (for odd $r > 1$).
We will show that when $r = 5$ the Time Complexity bound is ...

$$T(n) = \begin{cases} 
  cn, & n < 24 \\
  T(n/5) + T(3n/4) + cn, & n \geq 24 
\end{cases}$$

Will show has solution $T(n) = O(n)$

Practical when $n$ is in the hundreds.
int select2 (Type a[], int k, int low, int hi)
// Find $k^{th}$ smallest element in a[ low : hi ]
{
  int n = hi - low + 1; // r is 5
  if (n <= 23) Sort a[low:hi];
    return the $k^{th}$ smallest element;
  divide a[low:hi] into n/5 adjacent subsets; of
    size 5 each; ignore excess elements;
  Sort (in place) each group of 5 elements;
  let m[i], 1 <= i <= ⌊ n/5 ⌋ be the
    set of medians of the n/5 subsets;
  v = select2(m, ⌈⌊ n/5 ⌋ /2 ⌉, 1, ⌊ n/5 ⌋);
  // Assume v is at position j
  Swap ( a[low], a[j] );
  // Use qselect skipping last two ifs
  select(a, low, hi, k); // Last 2 ifs are skipped;
  Assume that v ends at position j in array a;
  if (k == j-low+1) return (v);
  if (k < j-low+1) return select2(a, k, low, j-1);
    else return select2(a, k-(j-low+1), j+1, hi);
}
Element deleted when $r = 5$ is $\geq 1.5\lfloor n/5 \rfloor$.

The total number of remaining elements is $\leq$:

$$n - 1.5\lfloor n/5 \rfloor \leq .7n + 1.2, \text{ iff } .3n \leq 1.5\lfloor n/5 \rfloor + 1.2, \text{ iff } n \leq 5\lfloor n/5 \rfloor + 4 \text{ True all the time}$$

$$\begin{aligned}
.7n + 1.2 &\leq 3n/4, \text{ iff } \\
1.2 &\leq 0.05n, \text{ iff } \\
24 &\leq n
\end{aligned}$$

Therefore, when $r = 5$ the time complexity is given by

$$T(n) = \begin{cases} 
  cn, & n < 24 \\
  T(n/5) + T(3n/4) + cn, & n \geq 24 
\end{cases}$$
Solution to Recurrence Relation

Solution is \( T(n) \leq 20cn \).

Obvious when \( n < 24 \).

Assume \( T(n) \leq 20cn \) for all \( n < m \) and let us now show that it holds for \( n = m \).

\[
T(n) \leq T(n/5) + T(3n/4) + cn \\
\leq 20c(n/5) + 20c(3n/4) + cn \\
= 4cn + 15cn + cn \\
= 20cn
\]

Therefore, the solution is \( T(n) \leq 20cn \).
Other Considerations

When \( r = 3 \) the recurrence relation is

\[
T(n) \leq T(n/3) + T(2n/3) + cn. \text{ Which is not } O(n).
\]

Solutions for Repeated Elements (See [Sa])

- Split the set into elements less than \( v \), elements equal to \( v \), and element greater than \( v \). Then if there are recursive calls they must either include all elements larger than \( v \) or smaller than \( v \).

- Careful analysis shows that the method takes linear time when \( r = 9 \), but does not take linear time when \( r \) is 5 or 7. The reason is that not enough elements are deleted. The book gives the recurrence relation for this case (r=9).
Finding a Closest Pair of Points in 2D (DC in 10.2.2)

Give \( n \) points in 2D Space, find a closest pair of points wrt Euclidean distance, i.e., the distance between points \((x_1, y_1)\) and \((x_2, y_2)\) is

\[
\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.
\]

Straight Forward Method

Find the distance between every pair of the \( n \) points, find the smallest of those distances \((\delta)\), and then find two points at a distance \( \delta \) of each other.

Since we need to consider about \( n^2 \) distances, the above algorithm takes \( O(n^2) \) time.
Sweep Line Method

Sort the points in ascending order wrt to their $x$-coordinate value.

Then “sweep” the points from left to right.

When visiting the $i^{th}$ point you have a closest pair of points among the first $i - 1$ points. This is updated to a closest pair of points among the first $i$ points by looking at all the distances between the $i^{th}$ point and the previous $i - 1$ points.

The time complexity for this method is also $O(n^2)$. 
Divide and Conquer (10.2.2)
Partition Into Two Subproblems

Solve Each Subproblem

DIVIDE

\[ d_1 = \text{distance between closest pair in } P_1 \]
\[ d_2 = \text{distance between closest pair in } P_2 \]

So far \( T(n) = 2T(n/2) \) so the remaining time that we can use is only \( cn \).
Solve Subproblem in the Band

"BAND"

\[ d = \min( d_1, d_2 ) \]

\( d_3 \) closest pair in the 2d band
(one point in \( P_1 \) and the other in \( P_2 \))

Compare the topmost on the left side of the Band against the ones in the right side of the Band, then the next one on the left ..., etc.

What if all the points in the band? The procedure will take \( n^2 \) time!
**Windows**

Just compare them to the ones in the "Window".

Can all the points be inside the window?

**NO!!!** Compare with at most Six!!

Points need to be sorted with respect to $X$ and $Y$, which can be done initially.
Main Idea

MAIN Idea
if (n is small) use brute force and return;
Divide the point set into roughly equal parts A and B;
Determine the closest pair in A;
Determine the closest pair in B;
Determine the closest pair in “the band”;
Find the closest pair among the previously computed three closest pairs;
End
class Point2;

class Point1 {
    friend float dist(const Point1&, const Point1&);
    friend void close(Point1 *, Point2 *, Point2 *,
    int, int, Point1&, Point1&, float&);
    friend bool closest(Point1 *, int, Point1&,
    Point1&, float&);
    friend void main();

public:
    int operator<=(Point1 a) const
    {return (x <= a.x);}

private:
    int ID;  // point identifier
    float x, y;  // point coordinates
};
class Point2 {
friend float dist(const Point2&, const Point2&);
friend void close(Point1 *, Point2 *, Point2 *,
    int, int, Point1&, Point1&, float&);
friend bool closest(Point1 *, int, Point1&,
    Point1&, float&);
friend void main();
public:
    int operator<=(Point2 a) const
    {return (y <= a.y);}
private:
    int p;  // index to same point in array X
    float x, y;  // point coordinates
};
template<class T>
inline float dist(const T& u, const T& v)
{ // Distance between points u and v.
    float dx = u.x - v.x;
    float dy = u.y - v.y;
    return sqrt(dx * dx + dy * dy); }
void close(Point1 X[], Point2 Y[], Point2 Z[],
  int l, int r, Point1& a, Point1& b, float& d)
{// X[l:r] is sorted by x-coordinate.
  // Y[l:r] is sorted by y-coordinate.
  if (r-l == 1) {// two points
    .........  return;}
  if (r-l == 2) {// three points
    // compute distance between all pairs
    .........  // find closest pair
    .........  return;}

  // more than 3 points, divide into two
  int m = (l+r)/2; // X[l:m] in A, rest in B
  //create y-sorted lists in Z[l:m] & Z[m+1:r]
  int f = l,  // cursor for Z[l:m]
      g = m+1;  // cursor for Z[m+1:r]
  for (int i = l; i <= r; i++)
    if (Y[i].p > m) Z[g++] = Y[i];
    else Z[f++] = Y[i];
// solve the two parts
close(X,Z,Y,l,m,a,b,d);
float dr;
Point1 ar, br;
close(X,Z,Y,m+1,r,ar,br,dr);

// make (a,b) closer pair of the two
if (dr < d) {a = ar;
            b = br;
            d = dr;}

Merge(Z,Y,l,m,r);// reconstruct Y

// put points within d of mid point in Z
int k = l; // cursor for Z
for (int i = l; i <= r; i++)
  if (fabs(X[m].x-Y[i].x) < d) Z[k++] = Y[i];
// search for closer category 3 pair
// by checking all pairs from Z[l:k-1]
for (int i = l; i < k; i++){
    for (int j = i+1;
        j < k && Z[j].y - Z[i].y < d; j++)
    {
        float dp = dist(Z[i], Z[j]);
        if (dp < d) {// closer pair
            d = dp;
            a = X[Z[i].p];
            b = X[Z[j].p];}
    }
}
bool closest(Point1 X[], int n, Point1& a, Point1& b, float& d)
{
    // Find closest pair from n >= 2 points.
    // Return false if fewer than two points.
    // Otherwise, return closest points in a and b.
    if (n < 2) return false;
    // sort on x-coordinate
    MergeSort(X, n);
    // create a point array sorted on y
    Point2 *Y = new Point2 [n];
    for (int i = 0; i < n; i++) {
        // copy point i from X to Y and index it
        Y[i].p = i; Y[i].x = X[i].x;
        Y[i].y = X[i].y; }
    MergeSort(Y, n); // sort on y-coordinate
    // create temporary array needed by close
    Point2 *Z = new Point2 [n];
    close(X, Y, Z, 0, n-1, a, b, d); // find closest pair
    // delete arrays and return
    delete [] Y; delete [] Z;
    return true; 
}
void main()
{
    Point1 X[100], a, b;
    int n;
    float d;
    cout << "Enter number of points" << endl;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cout << "Enter point " << (i + 1) << endl;
        cin >> X[i].x >> X[i].y;
        X[i].ID = i + 1;
    }
    cout << "Return status is " <<
         closest(X, n, a, b, d) << endl;
    cout << "Closest points are " << a.ID << " and " << b.ID << endl;
    cout << "Their distance is " << d << endl;
}
Convex Hull (Just ideas [Sa] and [HS])

For set of points $S$ in 2D is the Smallest Convex polygon containing $S$.

**Simplest Method**

$O(n^2)$ time.
QuickHull \( (O(n^2)) \) Avg. \( O(n \log n) \)

Largest area triangle.
Divide and Conquer $O(n \log n)$