DIVIDE AND CONQUER GENERAL STRATEGY

```
DANDC (p,q)
{
if (small (p,q))
        return G(p,q);
else
        m=divide (p,q);
        return (Combine (DANDC(p,m),DANDC(m+1,q));
}
```

PARTITION ALGORITHM

```
int Partition (T[i..j])
{
    p=T[i]; l=i; u=j+1;
    do
    {
            repeat l++ until T[l]>p or k>=j;
            repeat u– until T[u]<=p;
            if (l<u) swap (T[l], T[u]);
    }
    While (l<u);
Swap (T[i], T[u]);
return (u);
}
```

MERGE SORT

```
mergesort (i,j)
{
    if (j-i<=16) insertionsort();
   else
    {
           mid=( i+j)/2
           mergesort (i,mid);
           mergesort (mid+1,j);
           merge (i,mid,j);
    }
}
```

MERGE SORT USING LINKS

```
mergesort (i,j)
{
   if (j-i<=16) insertionsort();
   else
   {
        mid=( i+j)/2
        q=mergesort (i,mid);
        r=mergesort (mid+1,j);
        merge (q,r);
   }
}
```

```
Merge1(q,r)
{
   i=q; j=r; k=0;
   while ((i!=0) and (j!=0))
           if (A[i]<= A[j])
                    link[k]=i;k=i; i=link[i];
           else
                    link[k]=j; k=j; j=link[j];
if (i==0) link[k]=j;
else link[k]=I;
return link[0];
}
```

MATRIX MULTIPLICATION

Multiply 2 x 2 Matrices:

$$\begin{vmatrix} r & s \\ t & u \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \begin{vmatrix} e & g \\ f & h \end{vmatrix}$$

r = a*e + b*f

s = a*g + b*h

t = c*e + d*f

u = c*g + d*h

Strassen's Algorithm

Multiply 2 x 2 Matrices:

$$\begin{vmatrix} r & s \\ t & u \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \begin{vmatrix} e & g \\ f & h \end{vmatrix}$$

$r = p_1 + p_4 – p_5 + p_7$

$s = p_3 + p_5$

$t = p_2 + p_5$

$u = p_1 + p_3 – p_2 + p_7$

Where:

$p_1 = (b + d)(f + g)$

$p_2 = (c + d)e$

$p_3 = a(g – h)$

$p_4 = d(f – e)$

$p_5 = (a – b)h$

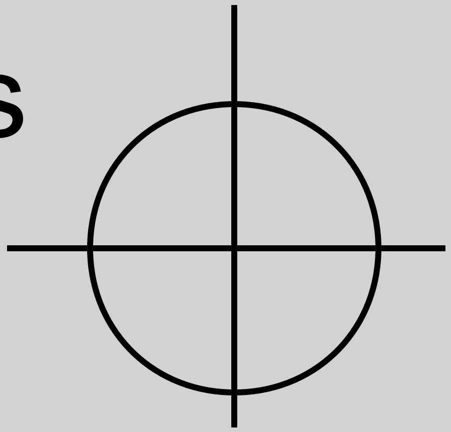
$p_6 = (c – d)(e + g)$

$p_7 = (b – d)(f + h)$

FFT, Convolution and Polynomial Multiplication

- Preview
  - FFT - O(n log n) algorithm
    - Evaluate a polynomial of degree n at n points in O(n log n) time
  - Computation of Convolution and Polynomial Multiplication (in O(n log n)) time

Complex Analysis



- Polar coordinates: re$^{!\ \ i}$
- e$^{!\ \ i}$ = cos !   + i sin !
- a is a nth root of unity if a^{n} = 1
- Square roots of unity: +1, -1
- Fourth roots of unity: +1, -1, i, -i
- Eighth roots of unity: +1, -1, i, -i, !   + i!   , !   - i!   , -!   + i!   , -!   - i!   where !   = sqrt(2)

Polynomial Multiplication

$$\begin{tikzcd}
\begin{array}{l} \text{n-1 degree polynomials} \\ A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}, \\ B(x) = b_0 + b_1x + b_2x^2 + \ldots + b_{n-1}x^{n-1} \end{array} \arrow[d, "{p_1, p_2, . . ., p_{2n}}"'] & \begin{array}{l} C(x) = A(x)B(x) \\ C(x)=c_0+c_1x + c_2x^2 + \ldots + c_{2n-2}x^{2n-2} \end{array} \\
\begin{array}{l} A(p_1), A(p_2), . . ., A(p_{2n}) \\ B(p_1), B(p_2), . . ., B(p_{2n}) \end{array} \arrow[r, "{C(p_i) = A(p_i)B(p_i)}"'] & C(p_1), C(p_2), . . ., C(p_{2n}) \arrow[u]
\end{tikzcd}$$

FFT Algorithm

```
// Evaluate the 2n-1th degree polynomial A at
// $!_{0,2n}, !_{1,2n}, !_{2,2n}, . . ., !_{2n-1,2n}$
FFT(A, 2n)
    Recursively compute FFT($A_{even}$, n)
    Recursively compute FFT($A_{odd}$, n)

    for j = 0 to 2n-1
        $A(!_{j,2n}) = A_{even}(!^2_{j,2n}) + !_{j,2n}A_{odd}(!^2_{j,2n})$
```

INVESTMENT PROBLEM

You’re consulting for a small computation-intensive investment company, and they have the following problem they wish to solve. They’re doing simulation in which they look at n consecutive days of a given stock, at some point in the past. Let’s number the days 1,2,3..n; for each day ‘i’ they have a price p(i) per share for the stock on that day. Suppose during this time period, they wanted to buy 1000 shares on some day and sell all these shares on some later day. They want to know : When should they have bought and when should they have sold in order to have made as much money as possible. For example suppose n=3, p(1)=9,p(2)=1, p(3)=5. Then answer is “buy on 2 sell on 3” . The brute force algorithm is to try all possible combinations of buying and selling. But this will be O(n*n). They wish to have a better solution.

The median of an ordered set is an element such that the number of elements less than the median is within one of the number that are greater, assuming no ties. Write the best possible algorithm that can compute the median of 'n' numbers. Simulate it on any data of your choice. State its' best, average, worst Case Time complexity.

Suppose you are consulting for a bank concerned with fraud detection. They have n bank cards that they've confiscated suspecting them of being used in fraud. The bank has a high-tech “equivalence tester” that takes two cards and after some computations determines if they are equivalent, since one cannot read anything from card directly.

Their question is the following: among the collection of n cards, is there a set of more than n/2 of them that are all equivalent to each other? Assume the only operation you can do, with the cards is to pick up two of them and plug into equivalence tester. Show how to do with O(nlogn) invocations of the tester.

You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values – so there are 2n values total- and you may assume that no two values are the same. You’d like to determine the median of this set of 2n values. However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the kth smallest value that it contains. Give an algorithm that computes the median using at most O(log n)