DYNAMIC PROGRAMMING FFT

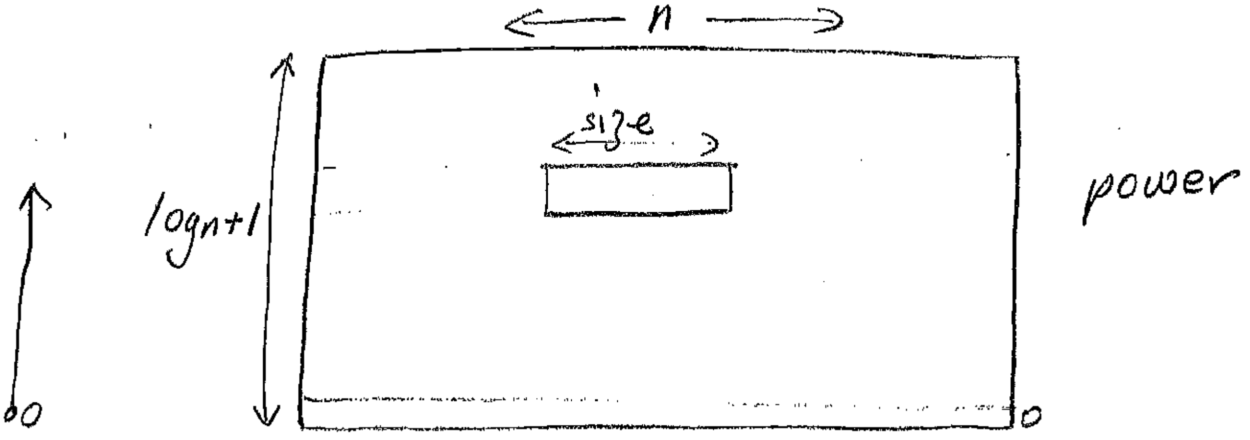
```
int RBS(int i, int k)
// returns the bit reversed index of
// input i, k the number of bits (log2 n)
if k==0 return i;
if i%2 == 1;
    return expt(2,k-1)+RBS(i/2,k-1);
else return RBS(i/2, k-1);


complex[] DPFFT(Complex[] poly, n)
 int logN= log2 n;
 complex[,] Sol = new Complex[logN+1, n];
 for (i=0; i<n; i++)
       Sol[0, RBS(i, log N)]= Poly[i]

  // Solution array initialized with base
  // cases
```

DYNAMIC PROGRAMMING FFT

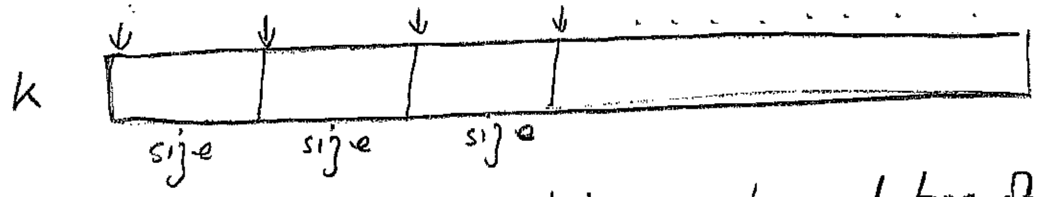
```
int power = n/2 ; // of Omega start at bottom
int size = 2; // smallest problem size
// scan from bottom to top
for (k=1; k ≤ log N; k++)
    // scan across by size for each subsolution
{   for (i=0; i<n; i=i+size)
        // fills in this solution
        for (j=0; j<size/2; j++)
          { odd = ω[j*power]* Sol[k-1, i+j+size/2];
            // + 1/2 solution
            S[k, i+j]= Sol[k-1, i+j]+ odd;
            // - 1/2 solution
            S[k, i+j+size/2]= Sol[k-1, i+j] -Odd;
          }
      // decrease power (more up)
      power = power/2;
      // increase size
      size = size*2
}
```

DYNAMIC PROGRAMMING FFT

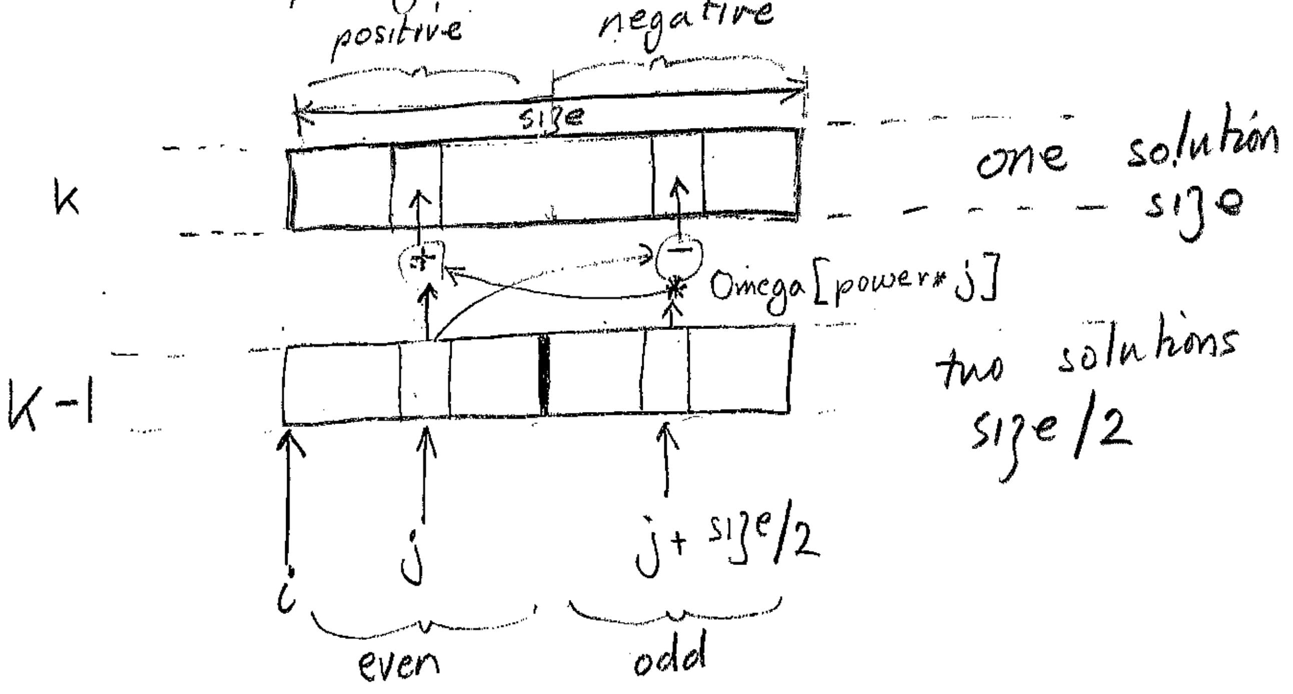
k loop starts a 1 and moves up



i loop scans over a row for each solution of size



j loop scans within each solution of size filling in the values from the lower solutions


positive
negative
size
k
one solution size
+
−
Omega[power*j]
K-1
two solutions size/2
i
j
j+size/2
even
odd