Lecture 16: Divide and Conquer Algorithms and Recurrences

Please use the notes on Canvas not 10.6 Grimaldi.

What is the fastest algorithm for sorting an array of n numbers ?
What is the fastest algorithm to multiply two polynomials of degree n ?

Sorting Algorithms

Suppose we want to sort an A of n integers e.g.

$$A = \boxed{9 \mid 3 \mid 11 \mid 2 \mid 6 \mid 13 \mid 5}$$

To compare sorting algorithms, by tradition, we count the number of comparisons they do. Bubblesort does exactly $n(n-1)/2$ comparisons. Mergesort does at most $n\log_2 n - n + 1$ comparisons. Below is a table for various values of n comparing the number of comparisons of these two algorithms.

	n	4	16	64	1024	10^6
Bubblesort	$n(n-1)/2$	6	120	2016	523776	approx 5×10^{11}
Mergesort	$n\log_2 n - n + 1$	5	49	321	9217	approx 20×10^6

For $n = 10^6$ Mergesort does a factor of over $25,000$ fewer comparisons!
Demo Mergesort

```c
void Merge( int A[], int n1, int B[], int n2, int C[] ) {
// Merge the sorted arrays A of length n1 and B of length n2 into C
   int i,j,k;
   i = j = k = 0;
   while( i<n1 && j<n2 )
      if( A[i]<B[j] ) { C[k] = A[i]; i++; k++; }
      else { C[k] = B[j]; j++; k++; }
   while( i<n1 ) { C[k] = A[i]; i++; k++; }
   while( j<n2 ) { C[k] = B[j]; j++; k++; }
   return;
}
```

Figure: C code for merging two sorted arrays A and B into the array C

The Mergesort Algorithm

```
void Mergesort( int A[], int n, int C[] ) {
// sort A[0],A[1],...,A[n] into ascending order
// C is an array of length n for working storage
   int n1,n2,*B;
   if( n<=1 ) return;
   n1 = n/2;
   n2 = n-n1;
   B = A + n1;
   Mergesort(A,n1,C); // sort the first half of A
   Mergesort(B,n2,C); // sort the second half of A
   Merge(A,n1,B,n2,C); // merge A and B into C
   for( i=0; i<n; i++ ) A[i] = C[i]; // copy C into A
   return;
}
```

Solving $C(n) \leq 2C(n/2) + n - 1$ with $C(1) = 0$.

Divide and Conquer Algorithms

Suppose we are given a problem of size n.

S1: Divide the problem into $a \geq 2$ subproblems of approximately the same size, say size b. Algorithm Mergesort divided A into $a = 2$ subproblems of size $n_1 = n/2$ and $n_2 = n - n_1$.

S2: Solve the subproblems recursively using the same "divide-and-conquer" approach.

S3: Combine the results from the subproblems to obtain the final solution. Algorithm Mergesort merges two sorted arrays of size n_1 and n_2 into one sorted array of size n.

Example. Adding an array of numbers.

```
double Add( double A[], int n ) {
// Add A[0]+A[1]+...+A[n-1]
   double s1,s2,*B;  int n1,n2;
   if( n==1 ) return A[0];
   n1 = n/2; n2 = n-n1;
   s1 = Add(A,n1); // s1 = A[0]+A[1]+...+A[n1-1]
   B = A + n1; // B is a subarray of A starting at n1
   s2 = Add(B,n2); // s2 = A[n1]+A[n2+1]+...+A[n-1]
   return s1+s2;
}
```

Solving recurrences using Maple's rsolve command.

A second order recurrence

```
> re := a(n) = 5*a(n-1) - 6*a(n-2);
```

$$re := a(n) = 5\,a(n-1) - 6\,a(n-2)$$

```
> rsolve( {re,a(0)=1,a(1)=4}, a(n) );
```

$$2\,3^n - 2^n$$

The mergesort recurrence

```
> re := c(n) = 2*c(n/2) + n-1;
```

$$re := c(n) = 2\,c(n/2) + n - 1$$

```
> expand( rsolve( {re, c(1)=0}, c(n) ) );
```

$$-n + \frac{\ln(n)\,n}{\ln(2)} + 1$$