CS 260 Week 1

The Order of Growth, Calculating the Running Time of a Program, Analysis of Merge Sort

The Order of Growth

Definitions of $\boldsymbol{O}$ and $\boldsymbol{\Omega}$ (textbook page 18)

$f(n) = O\big(g(n)\big)$: there are constants $c > 0$ and a non-negative integer n_0 such that for all $n \geq n_0$ we have $f(n) \leq cg(n)$

$f(n) = \Omega\big(g(n)\big)$: there is a constant $c > 0$ such that $f(n) \geq cg(n)$ for infinitely many n

The Order of Growth

Auxiliary Facts

Fact 1. If $0 < \alpha < \beta$, then

$$\frac{n^\alpha}{n^\beta} \to 0, \text{as } n \to \infty.$$

Fact 2. For $\alpha > 0$ we have

$$\frac{\log n}{n^\alpha} \to 0, \text{as } n \to \infty.$$

Fact 3. If $\frac{f(n)}{g(n)} \to 0, \text{as } n \to \infty$, then $f(n) = O\big(g(n)\big)$.

The Order of Growth

Example Problems (selected aspects of textbook problems 1.10, 1.16)

1. Examine all O and Ω relations between f_1, f_2, f_3, where

$$f_1(n) = n^2,\ f_2(n) = n^2 + 1000n,\ f_3(n) = \begin{cases} n \text{ if } n \text{ is odd} \\ n^3 \text{if } n \text{ is even} \end{cases}.$$

2. Order the following functions according to their growth rate: (a) $n^{1/2}$, (b) $\log_2 n$, (c) $n\log_2 n$, (d) n, (e) n^2.

Calculating the Running Time of a Program: Bubble Sort, Textbook Example 1.9

```
procedure bubble ( var A: array [1..n] of integer );
{ bubble sorts array A into increasing order }
var
          i, j , temp: integer;
          begin
(1)                 for i:= 1 to n-1 do
(2)                           for j:= n downto i+1 do
(3)                                     if A[j-1] > A[j] then begin
                                                { swap A[j-1] and A[j] }
(4)                                               temp:= A[j-1];
(5)                                               A[j-1]:= A[j];
(6)                                               A[j]:= temp
                                        end
          end; { bubble }
```

Calculating the Running Time of a Program: $n!$, Textbook Example 1.10

```
function fact ( n: integer ) : integer;
    begin
        if n <= 1 then
            return (1)
        else
            return (n * fact(n-1))
    end; { fact }
```

Calculating the Running Time of a Program: 2^{n-1}, Textbook Problem 1.12 d

```
function F( n: integer) : integer;
    begin
        if n <= 1 then
            return (1)
        else
            return (F(n-1) + F(n-1))
    end; { F }
```

Analysis of Merge Sort: Textbook Section 9.2

```
function mergesort ( L: List; n: integer ) : List;
    { L is a list of length n. A sorted version of L
      is returned. We assume that n is a power of 2. }
    var
            $L_1$, $L_2$: List;
    begin
            if n=1 then
                    return (L)
            else begin
                    break L into two halves, $L_1$ and $L_2$, each of length n/2;
                    return ( merge (mergesort($L_1$,n/2), mergesort($L_2$,n/2)))
            end
    end; { mergesort }
```