CS1102S Data Structures and Algorithms

Assignment 01: Algorithm Analysis – Solution

1. Exercise 2.1 on page 50: Order the following functions by growth rate: N, $\sqrt{N}$, $N^{1.5}$, N^2, $N\log N$, $N\log\log N$, $N\log^2 N$, $N\log(N^2)$, $2/N$, 2^N, $2^{N/2}$, 37, $N^2\log N$, N^3. Indicate which functions grow at the same rate and show why this is the case.

   Answer:

$$\begin{aligned} 2/N &< 37 < \sqrt{N} < N < N\log\log N < N\log N \leq N\log(N^2) \\ &< N\log^2 N < N^{1.5} < N^2 < N^2\log N < N^3 < 2^{N/2} < 2^N \end{aligned} \tag{1}$$

   The only two functions that grow at the same rate are $N\log N$ and $N\log(N^2)$:

$$N\log(N^2) = 2N\log N = \Theta(N\log N) \tag{2}$$

   For all other functions, the ordering is strict. In particular the following functions do *not* grow at the same rate:

$$2^{N/2} \neq \Theta(2^n) \text{ , as: } \lim_{N\to\infty}\frac{2^{N/2}}{2^N} = \lim_{N\to\infty}\frac{2^{N/2}}{2^{N/2} * 2^{N/2}} = \lim_{N\to\infty}\frac{1}{2^{N/2}} = 0 \tag{3}$$

$$N\log^2 N = N[\log N]^2 \neq N\log\log N \tag{4}$$

$$\begin{aligned} N^{1.5} &\neq \Theta(N\log^2 N) \text{ , as } \lim_{N\to\infty}\frac{N^{1.5}}{N\log^2 N} = \lim_{N\to\infty}\frac{N^{0.5}}{\log^2 N} \\ &= \lim_{N\to\infty}\frac{0.5N^{-0.5}}{2\log N\frac{1}{N}} = \lim_{N\to\infty}\frac{0.25N^{0.5}}{\log N} = \infty \end{aligned} \tag{5}$$

2. Excercises 2.22–2.24, pages 53-54:

   (a) Show that X^{62} can be computed with only eight multiplications.

Answer:

$$
\begin{aligned}
X^{62} &= X^{20} \times X^{42} \\
X^{42} &= X^{20} \times X^{20} \times X^{2} \\
X^{20} &= X^{10} \times X^{10} \\
X^{10} &= X^{5} \times X^{5} \\
X^{5} &= X^{2} \times X^{2} \times X \\
X^{2} &= X \times X
\end{aligned} \tag{6}
$$

(b) Write the fast exponentiation routine without recursion in Java. Submit your solution on paper. You don't need to actually implement the algorithm (optional).

Answer:

```java
public static int pow(int base, int exp) {
  int acc = 1;
  int e = exp;
  int b = base;

  if (exp == 0) {
    return 1;
  }

  while (e != 1) {
    if ( e % 2 == 1) {
      acc *= b;
    }
    b *= b;
    e /= 2;
  }
  return acc * b;
}
```

To understand the algorithm, think of the binary representation of exp:

$$
\begin{aligned}
base^{exp} &= base^{\sum_i a_i 2^i} \\
&= \Pi_i base^{a_i 2^i}
\end{aligned} \tag{7}
$$

The index i ranges from 0 to $\lceil \log_2(N+1) \rceil$. In every step the next component $base^{a_i 2^i}$ (from the right) is added to the accumulator. The loop invariant is $acc = base^{exp \% 2^i}$. When the loop ends, the accumulator equals $base^{exp}$ which is the desired result.

For example, in the case of base=3 and exp=5 we have:

$$\begin{aligned} 3^5 &= 3^{1*2^2+0*2^1+1*2^0} \\ &= 3^{1*2^2} * 3^{0*2^1} * 3^{1*2^0} \end{aligned} \tag{8}$$

(c) Give a precise count on the number of multiplications used by the fast exponentiation routine. (Hint: Consider the binary representation of N.)

Answer: The fast exponentiation algorithm iterates over all bits in the binary representation of exp. In every iteration, the value of x is squared (one multiplication). If the current bit is 1, the value of x is multiplied with the result (another multiplication). When the number of bits is 1, n will be 1 or 0; in this case, no multiplication is carried out. Thus, the total number of multiplications is: $\{\#$ bits in N$\} + \{\#$ '1' in N$\} - 2$.