Chapter 15
Dynamic Programming

Algorithm Analysis

School of CSEE

Problem 1

1) Is the following statement true?
   "Dynamic programming is typically applied to optimization problems."

2) Compare dynamic programming vs divide-and-conquer.
   (a) How are they alike?
   (b) What is the difference?
   (c) Which requires more work?

Problem 2

1) What does “optimal substructure” mean?

2) What does “overlapping subproblems” mean?

Problem 3

The following is the naïve recursive algorithm for Fibonacci numbers.

```
long long rec_fib(int n){
    if (n < 2)
        return n;
    else
        return rec_fib(n-1) + rec_fib(n-2);
}
```

1) Express the time complexity T(n) of this algorithm as a recurrence equation.
2) What is the running time (lower/upper bound) of this algorithm?

Problem 3

```
long long rec_fib(int n){
    if (n < 2)
      return n;
    else
      return rec_fib(n-1) + rec_fib(n-2);
}
```

1) $T(n) = T(n-1) + T(n-2) + \Theta(1)$

Problem 3

Lower bound

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + \Theta(1) \\
&= T(n-2) + T(n-3) + \Theta(1) + T(n-2) + \Theta(1) \\
&= 2T(n-2) + T(n-3) + 2\,\Theta(1) \\
&\geq 2T(n-2) + 2\Theta(1) = 2^1T(n-2*1) + 2^1\Theta(1) \\
&= 2*\{T(n-3) + T(n-4) + \Theta(1)\} + 2^1\Theta(1) \\
&= 2T(n-3) + 2T(n-4) + 2\Theta(1) + 2\Theta(1) \\
&= 2\{T(n-4) + 2T(n-5) + \Theta(1)\} + 2T(n-4) + 2^2\Theta(1) \\
&= 2^2T(n-4) + 2T(n-5) + 2^1\Theta(1) + 2^2\Theta(1) \\
&\geq 2^2T(n-4) + (2^1+2^2)\Theta(1) = 2^2T(n-2*2) + (2^1+2^2)\Theta(1) \\
&\geq 2^3T(n-2*3) + (2^1+2^2+2^3)\Theta(1)
\end{aligned}
$$

……………..

Problem 3

$\geq 2^3T(n-2*3) + (2^1+2^2+2^3)\Theta(1)$

………………

$\geq 2^{(n-1)/2}T(n-2*(n-1)/2) + (2^1+2^2+\cdots+2^{(n-1)/2})\Theta(1)$

(n-2x=1. Thus, x=(n-1)/2)

$= 2^{(n-1)/2}T(1) + (2^{(n+1)/2}-2)\Theta(1)$

$= (2^{(n+1)/2}+2^{(n-1)/2} -2)\Theta(1)$

Since $T(n) \geq 2^{(n+1)/2}\Theta(1)$ when $n \geq 3$, $T(n)= \Omega(2^{n/2})$

Problem 3

Upper bound

$T(n) = T(n-1) + T(n-2) + \Theta(1)$

$\le 2T(n-1) + 2^0\Theta(1)$ (since $T(n-2) \le T(n-1)$)

$= 2\{T(n-2) + T(n-3) + \Theta(1)\} + \Theta(1)$

$\le 2^2T(n-2) + (2^1+2^0)\Theta(1)$ (since $T(n-3) \le T(n-2)$)

$= 2^2 \{T(n-3) + T(n-4) + \Theta(1)\}$

$\le 2^3T(n-3) + (2^2+2^1+2^0)\Theta(1)$

……………..

$\le 2^{n-1}T(n-(n-1)) + (2^{n-2}\ldots+2^2+2^1+2^0)\Theta(1)$

$= 2^{n-1}T(1) + (2^{n-1}-1)\Theta(1) = (2^n-1)\,\Theta(1)$

$\le 2^n\,\Theta(1)$

Since $T(n) \le 2^n\Theta(1)$, $T(n) = O(2^n)$

Problem 4

For Fibonacci numbers.

Find the running time using each of the following,

(a) Bottom-up DP algorithm

(b) Memoized DP algorithm

Fibonacci Number

```
Fib (n){
    int fib[n+1], i;
    fib[0] = 0;
    fib[1] = 1;
    for i=2 to n
        fib[i] = fib[i-1] + fib[i-2];
    return fib[n];
}
```

Running time = $\Theta(n)$

```
Fib (n){
    int fib, f1, f2;
    if (n < 2)
        fib = n;
    else
        if (list[n-1] == false)  f1 = Fib(n-1);
        else  f1 = list[n-1];
        if (list[n-2] == false)  f2 = Fib(n-2);
        else  f2 = list[n-2];
        fib = f1 + f2;
    list[n] = fib;
}
```

Running time = $\Theta(n)$

Problem 5

Write a program for recursive solution and dynamic program solution for Fibonacci number. Measure execution time for each.

- Use long long int type.
- Compare execution time for n=40, 43, 47, 49 etc

Check how many times fib(3) is called, when n=40

(First, check if your program is correct, when n= 5, 6..)