CS 506, Online Foundations of CS	**Homework 7 Recurrences Solution**	Dr. David Nassimi

Study Module 8 (Classes of Recurrence Equations)
Divide-and-Conquer Recurrences (Master Theorem) and Linear Recurrences

1. The following class of recurrences often arise in analysis of divide-and-conquer algorithms. (Note: a, b, c, d and β are constants; and n is an integer power of b, $n = b^k$.)

$$T(n) = \begin{cases} a\ T(n/b) + c\ n^\beta, & n > 1 \\ d, & n = 1 \end{cases}$$

Let $h = \log_b a$. We used *repeated substitution* method to derive the solution:

$$T(n) = \begin{cases} A\ n^h + B\ n^\beta & = \Theta(n^\beta), & h < \beta \\ A\ n^h + B\ n^\beta & = \Theta(n^h), & h > \beta \\ A\ n^h + B\ n^h \log n & = \Theta(n^h \log n), & h = \beta \end{cases} \quad (1)$$

where A and B are some constants for each case. (A slightly different and more general version of this class of recurrences is presented in various textbooks and is commonly known as *Master Theorem*.)

Use the above formula to find the solution form for each of the following recurrences, where n is a power of 2. Express the solution form invloving constants A and B. (Don't bother to find the constants.) Then express in $\Theta(\ )$ form.

(a) $T(n) = 2T(n/2) + 1$

$a = 2, b = 2, \beta = 0$. $h = \log_b a = \log_2 2 = 1$. Since $h \neq \beta$,

$$T(n) = An^h + Bn^\beta = An + B = \Theta(n).$$

(b) $T(n) = 2T(n/2) + n$

$a = 2, b = 2, \beta = 1$. $h = \log_b a = \log_2 2 = 1$. Since $h = \beta$,

$$T(n) = An^h \log n + Bn^\beta = An \log n + Bn = \Theta(n \log n).$$

(c) $T(n) = 2T(n/2) + n^2$

$a = 2, b = 2, \beta = 2$. $h = \log_b a = \log_2 2 = 1$. Since $h \neq \beta$,

$$T(n) = An^h + Bn^\beta = An + Bn^2 = \Theta(n^2).$$

(d) $T(n) = T(n/2) + 1$

$a = 1, b = 2, \beta = 0$. $h = \log_2 1 = 0$. Since $h = \beta$,

$$T(n) = An^h \log n + Bn^\beta = A \log n + B = \Theta(\log n).$$

2. Consider the following recurrence, where n is a power of 2, $T(1) = 0$, and

$$T(n) = 4T(n/2) + n, \ n > 1.$$

(a) Find the exact solution by repeated substitution.

$$\begin{aligned} T(n) &= n + 4T(n/2) \\ &= n + 4(n/2 + 4T(n/4)) \\ &= n + 2n + 4^2T(n/2^2) \\ &= n + 2n + 2^2n + 4^3T(n/2^3) \\ &= n + 2n + 2^2n + \cdots + 2^{k-1}n + 4^kT(n/2^k), \quad \text{where } n = 2^k \\ &= n(1 + 2 + 4 + \cdots + 2^{k-1}) + 4^kT(1), \quad \text{where } T(1) = 0 \\ &= n(2^k - 1) = n(n-1) = n^2 - n. \end{aligned}$$

(b) Find the exact solution by using the above (Master Theorem) formula to obtain the solution form, and then find the constants A and B.

We have $a = 4, b = 2, \beta = 1$. Thus, $h = \log_2 4 = 2$. Since $h \neq \beta$,

$$T(n) = An^h + Bn^\beta = An^2 + Bn.$$

The constants A and B may be found by using two values of $n = 1, 2$.

$$\begin{aligned} T(1) &= 0 = A + B, \\ T(2) &= 2 + 4T(1) = 2 = 4A + 2B. \end{aligned}$$

Thus, $A = 1, B = -1$. That is, $T(n) = n^2 - n$.

3. Consider the following recurrence equation, where n is a power of 2.

$$T(n) = \begin{cases} 4T(n/2) + n + 6, & n \geq 2 \\ 0, & n = 1 \end{cases}$$

Prove by induction that the solution is

$$T(n) = An^2 + Bn + C$$

and find the constants A, B, C.

Proof: For the base, $n = 1$,

$$T(1) = 0 = A + B + C$$

To prove the solution for any $n \geq 2$, suppose the solution is correct for $n/2$. That is,

$$T(n/2) = A(n/2)^2 + B(n/2) + C \quad \text{hypothesis}$$

Then,

$$\begin{array}{rcll} T(n) & = & 4T(n/2) + n + 6 & \text{from the recurrence} \\ & = & 4[An^2/4 + Bn/2 + C] + n + 6 & \text{from the hypothsis} \\ & = & An^2 + (2B+1)n + (4C+6) & \\ & = & An^2 + Bn + C & \text{To reach the conclusion} \end{array}$$

To satisfy the latter equality (to reach the conclusion), we need to equate term-by-term.

$$2B + 1 = B$$

$$4C + 6 = C$$

So we have three equations for A, B, C:

$$A + B + C = 0; \quad 2B + 1 = B; \quad 4C + 6 = C$$

We solve for the constants and get

$$B = -1, C = -2, A = 3$$

Therefore we have proved that

$$T(n) = 3n^2 - n - 2.$$

4. Find the solution of each of the following linear recurrences.

   (a)

$$F_n = \begin{cases} 3, & n = 0 \\ 29, & n = 1 \\ 10F_{n-1} - 21F_{n-2}, & n \geq 2 \end{cases}$$

$$F_n - 10F_{n-1} + 21F_{n-2} = 0$$
$$F_n = r^n$$
$$r^n - 10r^{n-1} + 21r^{n-2} = 0$$
$$r^{n-2}(r^2 - 10r + 21) = 0$$
$$(r-7)(r-3) = 0$$
$$r_1 = 7; \quad r_2 = 3$$
$$F_n = A7^n + B3^n$$

Use base cases (initial values) to find A, B.

$$F_0 = 3 = A + B$$
$$F_1 = 29 = 7A + 3B$$

We find $A = 5, B = -2$. Therefore,

$$F_n = 5 * 7^n - 2 * 3^n.$$

(b)

$$F_n = \begin{cases} 3, & n = 0 \\ 50, & n = 1 \\ 10F_{n-1} - 25F_{n-2}, & n \geq 2 \end{cases}$$

$$F_n - 10F_{n-1} + 25F_{n-2} = 0$$
$$F_n = r^n$$
$$r^n - 10r^{n-1} + 25r^{n-2} = 0$$
$$r^2 - 10r + 25 = 0$$
$$(r-5)^2 = 0$$
$$r_1 = r_2 = 5$$

Since we have reapted roots, the solution is:

$$F_n = A5^n + Bn5^n$$

Use base cases (initial values) to find A, B.

$$F_0 = 3 = A$$
$$F_1 = 50 = 5A + 5B$$

So, $A = 3, B = 7$. Therefore,

$$F_n = 3 * 5^n + 7 * n5^n.$$

5. A linear recurrence of order 3 has the following initial values.

$$F_0 = 4; \quad F_1 = 19; \quad F_2 = 69$$

The recurrence equation has the following characteristic equation in factored form.

$$(r-2)^2(r-5) = 0.$$

(a) Find the complete solution.

Roots are $r_1 = r_2 = 2, \quad r_3 = 5$. Therefore, the complete solution is

$$F_n = A2^n + Bn2^n + C5^n$$

Use the base cases to find the constants A, B, C.

$$\begin{aligned} F_0 &= 4 = A + C \\ F_1 &= 19 = 2A + 2B + 5C \\ F_2 &= 69 = 4A + 8B + 25C \end{aligned}$$

So, $A = 3, B = 4, C = 1$. Therefore,

$$F_n = 3 * 2^n + 4 * n2^n + 5^n.$$

(b) Work backward from the characteristic equation to find the recurrence equation.

$$\begin{aligned} &(r-2)^2(r-5) = 0 \\ &(r^2 - 4r + 4)(r-5) = 0 \\ &r^3 - 9r^2 + 24r - 20 = 0 \\ &r^n - 9r^{n-1} + 24r^{n-2} - 20r^n n - 3 = 0 \\ &F_n - 9F_{n-1} + 24F_{n-2} - 20F_{n-3} = 0 \\ &F_n = 9F_{n-1} - 24F_{n-2} + 20F_{n-3}. \end{aligned}$$

Additional Exercises (Not to be handed-in)

6. Consider the following recurrence equation.

$$T(n) = \begin{cases} 2T(n/2) + \log n, & n \geq 2 \\ 0, & n = 1 \end{cases}$$

(a) Try to prove by induction that the solution is

$$T(n) = An + B \log n$$

You will see the induction will fail because you get a constant term on one side but not the other side of an equality. That suggests to refine our guess by adding a constant.

(b) Now, prove by induction that the solution is as follows.

$$T(n) = An + B \log n + C.$$

The above procedure illustrates how we may use master theorem to guide us in guessing the solution when the recurrence equation is not exactly in the form required by our version of master theorem. In this case, the term at the end, which is called "forcing function", is $\log n$ and not the form n^β.

How did we come up with the above guess in the first place? Observe that

$$h = \log_b a = \log_2 2 = 1$$

So we first guessed the solution as

$$T(n) = An + B * \text{ (forcing function) } = An + B \log n.$$

But the induction proof failed because we were missing a constant to make the final equality. So we refined our guess by adding the constant C.

7. **Algebra Facts:**

   (a) Prove that for every integer $n \geq 2$,

   $$\lfloor \log \lfloor n/2 \rfloor \rfloor = \lfloor \log n \rfloor - 1$$

   Hint: Let $2^k \leq n < 2^{k+1}$. Then:

   $$k \leq \log n < k + 1,$$
   $$\lfloor \log n \rfloor = k.$$

   And,

   $$2^{k-1} \leq n/2 < 2^k$$
   $$2^{k-1} \leq \lfloor n/2 \rfloor < 2^k,$$
   $$k - 1 \leq \log \lfloor n/2 \rfloor < k,$$
   $$\lfloor \log \lfloor n/2 \rfloor \rfloor = k - 1.$$

   (b) Prove that for every integer $n \geq 2$,

   $$\lceil \log \lceil n/2 \rceil \rceil = \lceil \log n \rceil - 1.$$

   Hint: Let $2^k < n \leq 2^{k+1}$. Prove in a similar way.

8. Consider the following recursive version of binary-search algorithm.
   The original call to this is BS$(A, 0, n-1, X)$.

```
int BS (dtype A[ ], int i, int j, dtype X)
{ //Search A[i..j] for key X
1    if (i > j) return -1; //Not found if n = 0
2    m = ⌊(i + j)/2⌋;
3    if (X == A[m]) return m;
4    else if (X < A[m])
5       return BS(A, i, m − 1, X);
6    else
7       return BS(A, m + 1, j, X);
}
```

   Let $f(n)$ be the worst-case number of key comparisons to search an array of n elements. Let us count the key comparisons in lines 3 and 4 as one comparison. (We explained the justification in class.) Then,

$$f(n) = \begin{cases} 1, & n = 1 \\ f(\lfloor n/2 \rfloor) + 1, & n > 1. \end{cases}$$

(a) Use the recurrence equation to compute and tabulate $f(n)$ for $n = 1$ to 15.

n	1	2	3	4	5	6	7
$f(n)$	1	2	2	3	3	3	3

(b) Prove by induction that the solution is

$$f(n) = \lfloor \log_2 n \rfloor + 1.$$

Hint: Use the algebra fact proved above.

Proof: For $n = 1$, $f(1) = 1$ from the recurrence. And, the claimed solution is: $f(1) = \log 1 + 1 = 1$. Thus, the solution is correct for $n = 1$.
Now, we will prove the solution is correct for any $n \geq 2$, supposing that $f(m) = \lfloor \log_2 m \rfloor + 1, \ \forall m < n$.

$$\begin{aligned} f(n) &= f(\lfloor n/2 \rfloor) + 1 \\ &= \lfloor \log \lfloor n/2 \rfloor \rfloor + 1 + 1 \quad \text{by hypothesis} \\ &= (k-1) + 1 + 1 \\ &= k + 1 \\ &= \lfloor \log n \rfloor + 1. \end{aligned}$$

9. Given a sorted array A of n elements, and a search key X. Modify the binay-search algorithm to find the left-most occurence of X in the array, in case there are several elements equal to X. For example, for the following array, a search key $X = 5$ should return index 3.

i	0	1	2	3	4	5	6	7	8	9	10	11
$A[i]$	1	2	4	5	5	5	5	5	5	5	10	20

The algorithm must be asymptotically efficient, and have worst-case time $O(\log n)$, even if the entire array consists of values equal to X. (An algorithm that performs partly sequential search and thus has worst-case time $O(n)$ is completely unacceptable.)

(a) Write a recursive version of the modified binary search algorithm.

(b) Illustrate your algorithm for the above example array.

(c) Let $f(n)$ be the worst-case number of key comparisons to search an array of n elements. Write a recurrence for $f(n)$.

(d) Use the recurrence to compute and tabulate $f(n)$ for $n = 1$ to 16.

(e) Guess the exact solution for $f(n)$ and prove the correctness by induction.

Solution: There are several ways of writing this algorithm. Below is probably the best algorithm.

```
int BS (dtype A[ ], int i, int j, dtype X) {
//Search A[i..j] for key X
if (i == j) //case n = 1 terminates recursion
   { if (X == A[i]) return i;
   else return -1; };
m = ⌊(i + j)/2⌋
if (X ≤ A[m])
   return BS(A, i, m, X)
else
   return BS(A, m + 1, j, X)
}
```

Recurrence:

$$f(n) = \begin{cases} 1, & n = 1 \\ f(\lceil n/2 \rceil) + 1, & n > 1. \end{cases}$$

Computed Values:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$f(n)$	1	2	3	3	4	4	4	4	5	5	5	5	5	5	5	5

From the computed values in the above table, it is easy to see the solution is:

$$f(n) = \lceil \log n \rceil + 1.$$

Proof by induction: Let

$$2^{k-1} < n \leq 2^k.$$

Then,

$$2^{k-2} < \lceil n/2 \rceil \leq 2^{k-1}.$$

Induction base, $n = 1$, $f(1) = 1 = \lceil \log 1 \rceil + 1$.
Induction step:

$$\begin{aligned} f(n) &= f(\lceil n/2 \rceil) + 1 \quad \text{(from the recurrence)} \\ &= \lceil \log \lceil n/2 \rceil \rceil + 1 + 1 \quad \text{(from the hypothesis)} \\ &= (k-1) + 2 \\ &= k + 1 = \lceil \log n \rceil + 1. \end{aligned}$$

□

10. Given two sorted arrays A and B of size s and t, respectively. We want to merge them into array C of size $n = s + t$.

    (a) Write a non-recursive program to do the merge.

```
merge (dtype A[ ], int s, dtype B[ ], int t, dtype C[ ])
{
int i, j, k;
i = 0; j = 0; k = 0;
while (i ≤ s − 1 and j ≤ t − 1)
   { if (A[i] ≤ B[j])
      C[k++] = A[i++];
   else
```

```
        C[k++] = B[j++];
    }
while (i ≤ s − 1)
        C[k++] = A[i++];
while (j ≤ t − 1)
        C[k++] = B[j++];
}
```

(b) Write a recursive version of the program to do the merge.

Hint: You can formulate the recursion in such a way that on a recursive call, the arrays to be merged always start at index 0. That would eliminate the need for additional parameters. To accomplish this, the algorithm starts by comparing the largest element of A with the largest element of B.

```
merge (dtype A[ ], int s, dtype B[ ], int t, dtype C[ ])
{
if (s == 0 and t == 0) return;
if (s > 0 and t > 0) {
    if (A[s − 1] > B[t − 1])
        { C[s + t − 1] = A[s − 1];   s−− };
    else
        { C[s + t − 1] = B[t − 1];   t−− };
    merge(A, s, B, t, C);
    };
else {
    while (s > 0)
        { C[s − 1] = A[s − 1];    s−− };
    while (t > 0)
        { C[t − 1] = B[t − 1];    t−− };
    }
}
```

(c) Let $f(n)$ be the worst-case number of key comparisons to merge two sorted arrays with combined length of n. Write a recurrence for $f(n)$. Then find the solution by repeated substitution.

$$f(n) = \begin{cases} 0, & n = 1 \\ 1 + f(n-1), & n > 1 \end{cases}$$

Solution:

$$f(n) = 1 + f(n-1) = 1 + 1 + f(n-2) = 3 + f(n-3) = \cdots = n - 1 + f(1) = n - 1.$$