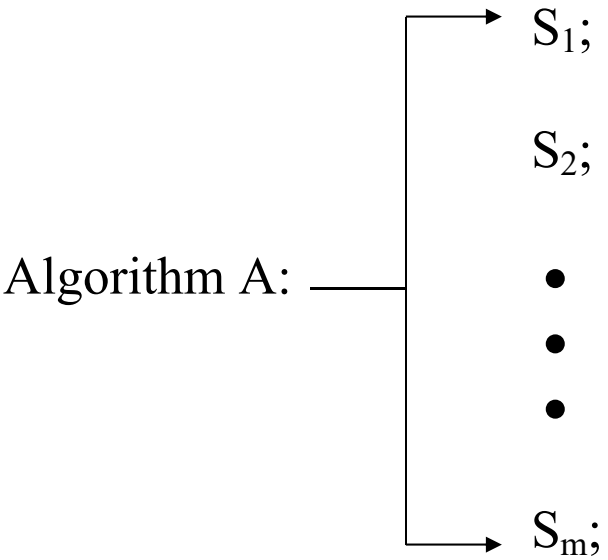
Topic 8: Recursive Functions & Their Applications

Read: Chpt.5.3-5.4, 8.1, Rosen

Recall that



$R(n) = \Sigma_{1 \le i \le m} \text{cost}(S_i)$.

Q: What if a statement S_i is a recursive call and, how do we compute R(n)?

A: We will use recursive function to characterize the performance function.

Recursive Function:

A recursive function is a function that is being defined by itself (recursive calls).

Format:

f(x) = g(f(y)), y < x, g is a function.

Structure of a Recursive Function:

- **Base case(s)**: The function is defined directly when the variable size is small.
- **Recursive case**: The function is defined in terms of itself but with a smaller variable size.

Implementation:

Function		**Algorithm**
Recursive function	→	recursive algorithm
Base case	→	terminating condition
Recursive case	→	recursive call

Example:

$$f(8) = 1,$$
$$f(n) = 2f(n-1) + 1, n > 8.$$

Computing $f(12)$ using (Top-Down) Forward Evaluation:

$$f(9) = 2f(9-1) + 1 = 2f(8) + 1 = 3,$$
$$f(10) = 2f(10-1) + 1 = 2f(9) + 1 = 7,$$
$$f(11) = 2f(11-1) + 1 = 2f(10) + 1 = 15,$$
$$f(12) = 2f(12-1) + 1 = 2f(11) + 1 = 31.$$

n	8	9	10	11	12	…	n
f(n)	1	3	7	15	31		?

Q: What is $f(n)$?
Can you represent $f(n)$ in closed-form?

Guessing: $f(n) = 2^{n-7} - 1, n \geq 8.$

Q: How can we verify it?
Use induction.

Consider the corresponding proposition P(n):
If $f(8) = 1, f(n) = 2f(n-1) + 1,$ then $\forall n \geq 8, f(n) = 2^{n-7} - 1.$
Proof: We use induction on n, n ≥ 8.
Basis step: When n = 8, $f(8) = 2^{8-7} - 1 = 1.$
Hence, P(8) is true.
Inductive step: Assume that P(k) is true for all k,
k ≥ 8. Hence, by assumption, $f(k) = 2^{k-7} - 1.$
We now need to prove that P(k+1) is true; or
$f(k+1) = 2^{k-6} - 1.$
By definition,

$$\begin{aligned} & f(k+1) \\ &= 2f(k) + 1, \\ &= 2 * (2^{k-7} - 1) + 1, \quad \text{(Inductive hypothesis)} \\ &= 2^{k-6} - 1. \end{aligned}$$

Hence, if P(k) is true, then P(k+1) is also true.
By induction, P(n) is true for all n ≥ 8.

Q: How can we compute the closed-form expression that $f(n) = 2^{n-7} - 1, n \geq 8$, without guessing?

Use the Method of Repeated Substitutions.

The Method of Repeated Substitutions:

$$
\begin{aligned}
& f(n) \\
&= 2f(n-1)+1 \\
&= 2[2f(n-2)+1]+1 \\
&= 2^2 f(n-2)+2^1+2^0 \\
&= 2^2[2f(n-3)+1]+2^1+2^0 \\
&= 2^3 f(n-3)+2^2+2^1+2^0 \\
&= \ldots \\
&= 2^{n-8} f(n-(n-8))+2^{n-9}+2^{n-10}+\ldots+2^1+2^0 \\
&= 2^{n-8} f(8)+\sum_{i=0}^{n-9} 2^i \\
&= 2^{n-8}+2^{n-8}-1 \\
&= 2^{n-7}-1.
\end{aligned}
$$

Remarks:

1. Must verify final solution (using induction).
2. The Method of Repeated Substitutions is a (bottom-up) backward evaluation method.
3. It is usually much more difficult to use forward evaluation to discover and compute a closed-form expression for f.

Analyzing Recursive Algorithms:

1. Factorial Function:

Defn: Given a non-negative integer n ≥ 0.

0! = 1,

n! = n∗(n-1)∗(n-2)…3∗2∗1, n > 0,

= n(n-1)!

```
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return (n*factorial(n-1));
} //end factorial
```

Q: How good is this recursive algorithm?

Let $T(n)$ be the amount of computing resources required to execute factorial(n) when the input is n.

Observe that $T(n)$ can now be defined recursively as follow:

$T(0) = c,$

$T(n) = T(n-1) + d,$ where c and d are constants.

Q: How do we compute $T(n)$?

Using the method of repeated substitutions, we have

$$
\begin{aligned}
& T(n) \\
&= T(n-1) + d \\
&= T(n-2) + 2d \\
&= T(n-3) + 3d \\
&= \ldots \\
&= T(n-k) + kd \\
&= \ldots \\
&= T(0) + nd \\
&= c + nd.
\end{aligned}
$$

HW: Prove that $T(n) = c + nd$ using induction.

2. Fibonacci Sequence Numbers:

Dfn: Given a non-negative integer $n \geq 0$.

$f_0 = 0$,

$f_1 = 1$,

$f_n = f_{n-1} + f_{n-2}$, $n > 1$.

Observe that

n	0	1	2	3	4	5	6	7	…	n
f(n)	0	1	1	2	3	5	8	13		**?**

Q: How do we compute $f(n)$?

Recursive Algorithm for Computing Fibonacci Sequence Numbers:

```
int fib(int n)
   {
     if (n == 0)
          return 0;
     else
     {  if (n == 1)
              return 1;
        else
              return fib(n-1) + fib(n-2);
     }
   }
```

Let $T(n)$ be the complexity in computing fib(n).

$T(n) = c,$

$T(1) = c,$

$T(n) = T(n-1) + T(n-2) + d, \forall n > 1,$ c & d are constants.

Q: How do we compute $T(n)$?

Using the method of repeated substitutions, we have

$$
\begin{aligned}
& T(n) \\
&= T(n-1) + (n-2) + d \\
&\le 2T(n-1) + d \\
&\le 2[2T(n-2) + d] + d \\
&= 2^2 T(n-2) + 2^1 d + 2^0 d \\
&\le 2^2 [2T(n-3) + d] + 2^1 d + 2^0 d \\
&= 2^3 T(n-3) + 2^2 d + 2^1 d + 2^0 d \\
&\ldots \\
&\le 2^{n-1} T(1) + 2^{n-2} d + 2^{n-3} d + \ldots + 2^1 d + 2^0 d \\
&= c 2^{n-1} + d \sum_{i=0}^{n-2} 2^i \\
&= \frac{c}{2} 2^n + d 2^n \\
&= O(2^n).
\end{aligned}
$$

3. Ackermann's Function:

Define the **Ackermann's function** $A : N \times N \to N$ by

$A(0, n) = n + 1,$

$A(m, 0) = A(m-1, 1)$, if $m > 0$,

$A(m, n) = A(m-1, A(m, n-1))$, if $m, n > 0$.

Example:

$A(0, 0) = 1,$

$A(0, 1) = 2,$

$A(0, 2) = 3,$

$A(0, 3) = 4,$

…

$A(1, 0) = A(0, 1) = 2,$

$A(1, 1) = A(0, A(1, 0)) = A(0, 2) = 3,$

$A(1, 2) = A(0, A(1, 1)) = A(0, 3) = 4,$

$A(1, 3) = 5,$

…

$A(2, 0) = A(1, 1) = 3,$

$A(2, 1) = A(1, A(2, 0)) = A(1, 3) = 5,$

$A(2, 2) = A(1, A(2, 1)) = A(1, 5) = 7,$

$A(2, 3) = 9,$

…

$A(3,0) = A(2, 1) = 5,$
$A(3, 1) = A(2, A(3,0)) = A(2, 5) = 13,$
$A(3, 2) = A(2, A(3, 1)) = A(2, 13) = 29,$
$A(3, 3) = A(2, A(3, 2)) = A(2, 29) = 61,$
…

$A(4,0) = A(3,1) = 13,$
$A(4,1) = A(3,A(4,0)) = A(3,13),$
$A(4,2) = ?$

Remark: A(m,n) is an extremely fast growing function and, hence, its inverse function $\alpha(m,n)$, which often appears in data structures analyses and counting, grows extremely slow. For all practical purpose, $\alpha(m,n)$ can be treated as a constant.

HW: Construct a recursive algorithm to compute A(m,n).

Another Example using Repeated Substitutions:

$T(2) = 1,$

$T(n) = 2T(\frac{n}{2}) + n + 1, n = 2^k, k > 1.$

$T(n)$

$= 2T(\frac{n}{2}) + n + 1$

$= 2[2T(\frac{n}{2^2}) + \frac{n}{2} + 1] + n + 1$

$= 2^2 T(\frac{n}{2^2}) + 2 * \frac{n}{2} + 2 + n + 1$

$= 2^2[2T(\frac{n}{2^3}) + \frac{n}{2^2} + 1] + 2 * \frac{n}{2} + 2 + n + 1$

$= 2^3 T(\frac{n}{2^3}) + 2^2 * \frac{n}{2^2} + 2^2 + 2 * \frac{n}{2} + 2 + n + 1$

$= 2^3 T(\frac{n}{2^3}) + 3n + 2^2 + 2^1 + 2^0$

$= \ldots$

$= 2^{k-1} T(\frac{n}{2^{k-1}}) + (k-1)n + \sum_{i=0}^{k-2} 2^i$

$= 2^{k-1}(1) + (k-1)n + (2^{k-1} - 1)$

$= 2 * 2^{k-1} + (k-1)n - 1$

$= n + n(\lg n - 1) - 1$

$= n \lg n - 1.$ (HW: Prove it.)

Given a sequence of real-valued functions $\{a_i\}_{i \in I}$.

Recurrence Relation (RR):

$f(a_n, a_{n-1}, \ldots, a_0) = 0.$ (*)

Solving a Recurrence Relation

Find a sequence $\{g_i\}_{i \in I}$ satisfying (*).

Modeling & Applications:

1. **Compound Interest Computation:**

   Given: *Initial deposit*: p
   Interest Rate: t%, compounded annually
   Term: n years

   Q: Compute total amount (principal + interest) after n years.

   Let P_n be the total amount after n years.

   Observe that

   $P_0 = p,$
   $P_1 = P_0 + P_0 * t/100,$
   $P_2 = P_1 + P_1 * t/100,$
   ...
   $P_n = P_{n-1} + P_{n-1} * t/100$
   $= P_{n-1} * (1 + t/100)$
   $= P_{n-2} * (1 + t/100) * (1 + t/100)$
   $= P_{n-2} * (1 + t/100)^2$
   $= P_{n-3} * (1 + t/100) * (1 + t/100)^2$

$= P_{n-3}*(1 + t/100)^3$
$= \ldots$
$= P_0*(1 + t/100)^n$

Hence, the total amount when matured after n years can always be computed using the function

$P_0 = p$,
$P_n = P_0*(1 + t/100)^n$.

2. **The Rabbit Problem:**

Given a pair of new-born rabbits, we assume that, ***after*** two months, this pair of rabbits will give birth to another pair of rabbits each month, and this pair of new rabbits will then go through the same cycle again. For simplicity, we assume that the rabbits will never die and this process will go on indefinitely.

Q: How many pairs of rabbits do we have at the beginning of the n^{th} month?

Observe that

Month	0	1	2	3	4	5	6	…	n
Pair Rabbits	0	1	1	2	3	5	8		?

Let a_n be the number of pair of rabbits we have at the beginning of the n^{th} month. Hence,

$a_0 = 0$,

$a_1 = 1$,

$a_2 = 1$, (before breeding starts)

$a_n = a_{n-1} + a_{n-2}$, $n > 2$.

(a_{n-2} = # of pairs of new-born rabbits at the beginning of the n^{th} month.)

Observe that a_n is simply the n^{th} term Fibonacci sequence number.

3. The Tower of Hanoi Problem:

Given 3 pegs and 64 different sized disks sorted on peg 1 such that the smallest (largest) disk is on top (bottom).

Q: How fast can we move all the disks from peg 1 to peg 2 if

(1) one can only move one disk at a time, and

(2) no larger disk can be placed on top of a smaller one?

Let H_n be the minimum # of moves required to move n disk from peg i to peg j, $i \neq j$, $1 \leq i, j \leq 3$. Observe that, in order to move n disks from peg 1 to peg 2, we must first move the smaller n–1 disks

from peg 1 to peg 3 (H_{n-1} moves) and then move the largest disk from peg 1 to peg 2 (1 move). Once the largest disk is moved to peg 2, we must now move the n−1 disks from peg 3 to peg 2 (another H_{n-1} moves). Hence,

$$H_1 = 1,$$
$$H_n = H_{n-1} + 1 + H_{n-1}$$
$$= 2H_{n-1} + 1, \; n > 1.$$

$$H_n = 2H_{n-1} + 1$$
$$= 2(2H_{n-2} + 1) + 1$$
$$= 2^2H_{n-2} + 2 + 1$$
$$= 2^2(2H_{n-3} + 1) + 2 + 1$$
$$= 2^3H_{n-3} + 2^2 + 2^1 + 2^0$$
$$= \ldots$$
$$= 2^{n-1}H_1 + 2^{n-2} + 2^{n-3} + \ldots + 2^1 + 2^0$$
$$= 2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2^1 + 2^0$$
$$= 2^n - 1.$$

In the original Tower of Hanoi problem, n = 64. Hence,

$$H_n = 2^{64} - 1$$
$$= 18{,}446{,}744{,}073{,}709{,}551{,}615$$

Remark: Even if you can move one disk per second, it will still take you more than 500 billion years to move the 64 disks!

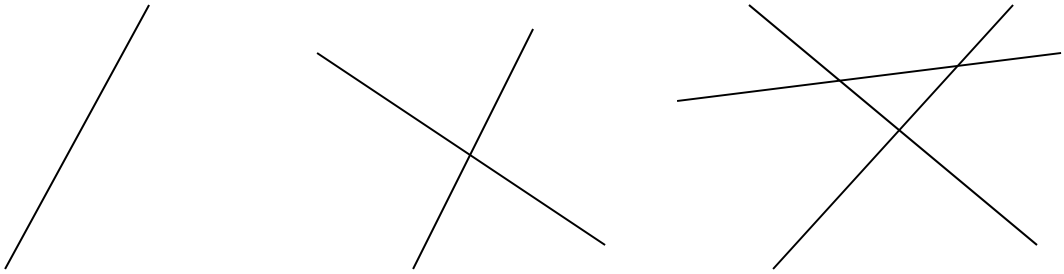
4. Plane Division Problem:

Q: How many regions can a plane be divided by n straight lines if

(1) every pair of lines must intersect, and

(2) no more than 2 lines can intersect at a common point.

Examples:



# lines:	1	2	3	4	5	6	7	8	…
# regions:	2	4	7	11	16	22	29	37	…

Observations:

1. The 2nd line interests the first line to yield 2 more regions.
2. The 3rd line interests the first 2 lines to yield 3 more regions.
3. The 4th line interests the first 3 lines to yield 4 more regions.

Q: Do you see the pattern?

In general, the n^{th} line interests the first n–1 lines to yield n more regions!

Let R_n be the number of regions divided by n lines.

Observe that

$R_1 = 2$,

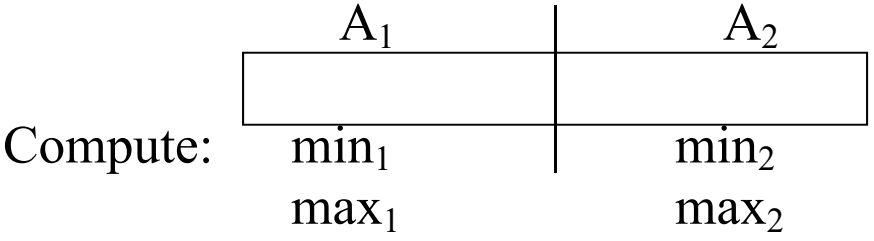
$R_n = R_{n-1} + n$, $n > 1$.

Hence,

$$\begin{aligned} R_n &= R_{n-1} + n \\ &= R_{n-2} + (n-1) + n \\ &= R_{n-3} + (n-2) + (n-1) + n \\ &= \dots \\ &= R_1 + 2 + 3 + \dots + (n-2) + (n-1) + n \\ &= [1 + 2 \dots + (n-1) + n] + 1 \\ &= [n(n+1)/2] + 1. \end{aligned}$$

5. MinMax-Finding Problem:

Input: An array A[1..n] of n distinct integers, $n = 2^k \geq 2$.
Output: The minimum and maximum integers in A.



Observe that $\min A = \min\{\min_1, \min_2\}$ and $\max A = \max\{\max_1, \max_2\}$.

Let T(n) be the number of comparisons in finding the min and max integers in A with size n.

$T(2) = 1,$
$T(n) = 2T(n/2) + 2,\ n = 2^k,\ k > 1 \ldots\ldots\ldots\ldots\ldots\ldots (*)$

Using the Method of Repeated Substitutions,

$$\begin{aligned}
T(n) &= 2T(n/2) + 2 \\
&= 2[2T(n/2^2) + 2] + 2 \\
&= 2^2T(n/2^2) + 2^2 + 2 \\
&= 2^2[2T(n/2^3) + 2] + 2^2 + 2 \\
&= 2^3T(n/2^3) + 2^3 + 2^2 + 2^1 \\
&= \quad \ldots \\
&= 2^{k-1}T(n/2^{k-1}) + 2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2^1 \\
&= 2^{k-1}T(2) + (2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2^1) \\
&= 2^{k-1} + (2^k - 1) - 1 \\
&= n/2 + n - 2 \\
&= 3n/2 - 2.
\end{aligned}$$

Q: How do we know that we have the right pattern? How do we verify that $T(n) = 3n/2 - 2$ is indeed the solution to the given RR (*)?

Two Basic Approaches:

1. Induction
2. Backward substitutions

1. Verification using Induction:

Let P(n) be the proposition function

$\forall$ n, $n = 2^k$, $k \geq 1$, $T(n) = 3n/2 - 2$ is the solution to (*).

By substituting $n = 2^k$, we have

$T(2^1) = 1$,

$T(2^k) = 2T(2^{k-1}) + 2, k > 1.$ ..... (**)

Solution:

$T(2^k) = 3(2^{k-1}) - 2.$

Let Q(k) be the proposition function

$\forall$ $k \geq 1$, $T(2^k) = 3(2^{k-1}) - 2$ is the solution to (**).

Observe that P $\leftrightarrow$ Q.

Hence, we can prove the validity of P by proving Q using induction.

Let's prove Q by using induction on k:

Basis step: For k =1,

$T(2^1) = 3(2^{1-1}) - 2 = 1.$

Hence, Q(1) is true for k = 1.

Inductive step: Need to prove that $\forall k, Q(k) \rightarrow Q(k+1)$.

Assume that $T(2^k) = 3(2^{k-1}) - 2$ is the solution to $T(2^k) = 2T(2^{k-1}) + 2$, we will prove that

$T(2^{k+1}) = 3(2^k) - 2$ is the solution to $T(2^{k+1}) = 2T(2^k) + 2$.

Observe that

$T(2^{k+1}) = 2T(2^k) + 2$ (Def of $T(2^{k+1})$)

$= 2[3(2^{k-1}) - 2] + 2$ (Inductive hypothesis)

$= [3(2^k) - 4] + 2$

$= 3(2^k) - 2$

Hence, if Q(k) is true, then Q(k+1) is also true.

By induction, Q, and hence P, is true.

2. Verification using Backward Substitutions:

To verify that $\forall$ n, $n = 2^k$, $k \geq 1$, $T(n) = 3n/2 - 2$ is the solution to

$T(2) = 1$,

$T(n) = 2T(n/2) + 2$, $n = 2^k$, $k > 1$,

we must prove that T(n) satisfies both the given initial condition(s) and the general RR.

Verifying the initial condition:

$$T(2) = 3(2)/2 - 2$$
$$= 1$$

Hence, the solution holds for the initial condition.

Verifying the general recurrence:

$$T(n) = 2T(n/2) + 2$$
$$= 2[3(n/2)/2 - 2] + 2$$
$$= [3(n/2) - 4] + 2$$
$$= 3n/2 - 2$$

Hence, the solution holds for the RR.

Practice HW:

Chpt.5.3, 3, 5, 7, 9, 11, 13, 15

Chpt.5.4, 9, 15

Chpt.8.1, 1,11

11/1/17