EECS2030 Fall 2017
Additional Notes
Solving Problems Recursively

Chen-Wei Wang

Given a problem of size n (e.g., an integer of value n, an array of n elements, *etc.*), adopt the following steps to solve the problem *recursively*:

Step 1: Understand the Problem We denote the original problem to be solved as P_n (i.e,. a problem P, where the subscript n denotes its size). For example:

Example 1. Compute the factorial of n.

Example 2. Compute the n^{th} number in the Fibonacci sequence.

Example 3. Compute if a string s of length n is a palindrome.

Example 4. Compute the reverse of a string s of length n.

Example 5. Compute the number of occurrences of a character c in a string s of length n.

Example 6. Compute if elements in index range $[from,\ to]$ of an array a are all positive.

Example 7. Compute if elements in index range $[from,\ to]$ of an array a are sorted in a non-descending order.

Example 8. Compute if elements in index range $[from,\ to]$ of a sorted array a contain a value k.

Step 2: Define the Base Cases We first define the solutions to the same problem whose sizes are small so that they can be solved immediately: P_0, P_1, P_2, *etc.* For example:

Example 1. Factorial 0 is just 1.

Example 2. The first and second Fibonacci numbers are both 1.

Example 3. An empty string and a string of length one are both palindromes.

Example 4. The reverse of an empty string or of a string of length one is simply the string itself.

Example 5. The number of occurrences of any character in an empty string is 0.

1. If index range $[from,\ to]$ is such that $from > to$, e.g., $[3, 2]$, then there is an empty collection of elements to be considered.

   Example 6. Since you cannot find a counter-example (i.e., a number which is not positive) from an empty collection, the result of determining all numbers being positive is simply *true*.

   Example 7. Since you cannot find a counter-example (i.e., a pair of adjacent numbers which are not sorted in a non-descending order) from an empty collection, the result of determining all numbers in an empty collection being sorted in a non-descending order is simply *true*.

   Example 8. Since an empty collection contains nothing, the result of determining if any value k exists in an empty collection is simply $false$.

2. If index range $[from,\ to]$ is such that $from == to$, e.g., $[3, 3]$, then there is a collection of exactly one element to be considered. We call such a collection a *singleton* collection. Say e is such an element that a singleton collection contains.

   Example 6. The result of determining all numbers being positive is simply $e > 0$.

   Example 7. Since you cannot find a counter-example (i.e., a pair of adjacent numbers which are not sorted in a non-descending order) from a collection of just one number, the result of determining all numbers in a singleton collection being sorted in a non-descending order is simply *true*.

   Example 8. The result of determining if any value k exists in a singleton collection is simply $k == e$.

Step 3: Assume that Solutions to Smaller Problems Exist We then assume that there exist solutions to **sub-problems** whose sizes are strictly smaller than the original problem: e.g., P_{n-1}, P_{n-2}, *etc.* For example:

Example 1. Assume the factorial of $n-1$ already exists (where $n > 0$). We denote this solution as P_{n-1} as its input size (i.e., value of number) is exactly one less than the original problem.

Example 2. Assume the $(n-1)^{th}$ and $(n-2)^{th}$ numbers in the Fibonacci sequence already exist (where $n > 2$). We denote these solutions as P_{n-1} and P_{n-2} as their input sizes (i.e., position in the Fibonacci sequence) are exactly, respectively, one and two less than the original problem.

Example 3. Assume we already know if a smaller substring of s (where $s.length() > 1$), with the first and last characters of s taken out, is a palindrome. We denote this solution as P_{n-2} as its input size (i.e., length of string) is two less than the original problem.

Example 4. Assume we already know the reverse of a smaller substring of s (where $s.length() > 1$), with the first character of s taken out. We denote this solution as P_{n-1} as its input size (i.e., length of string) is one less than the original problem.

Example 5. Assume we already know the the number of occurrences of a character c in a smaller substring of s (where $s.length() > 0$), with the first character of s taken out. We denote this solution as P_{n-1} as its input size (i.e., length of string) is one less than the original problem.

We assume we already know the solution for elements in a smaller index range $[from + 1,\ to]$ of an array a:

Example 6. We denote P_{n-1} as the solution for if the $n-1$ elements are all positive.

Example 7. We denote P_{n-1} as the solution for if the $n-1$ elements are sorted in a non-descending order.

Example 8. We denote P_{left} as the solution for if the left half (of roughly $\frac{n}{2}$ elements) of a sorted array contains a value k. Similarly, we denote P_{right} as the solution for if the right half (of roughly $\frac{n}{2}$ elements) of a sorted array contains a value k.

Step 4: Define the Recursive Cases We finally define the solution to the original problem P_n in terms of the solutions to other strictly smaller sub-problems: $P_n = f(\ P_{n-1},\ P_{n-2},\ \dots\ )$. That is, P_n is defined as a function f that combines solutions to strictly smaller problems P_{n-1}, P_{n-2}, *etc.* via some simple calculations. Informally speaking, we "massage" solutions to smaller problems into the solution to a bigger problem. For example:

Example 1. We define $P_n = n \times P_{n-1}$.

Example 2. We define $P_n = P_{n-1} + P_{n-2}$.

Example 3. We define $P_n = (c1 == c2$ **&&** $P_{n-2})$ (where $c1$ and $c2$ are, respectively, the first and the last characters of s). For example, *abcbc* is a palindrome because $a == c$ and *bcb* is a palindrome. However, *abccc* is not a palindrome because *bcc* is not a palindrome, even though $a == c$.

Example 4. We define $P_n = P_{n-1} + c1$ (where $c1$ is the first character of s, and the operator + means string concatenation). For example, the reverse of *abcd* is the reverse of *abc* (which is *dcb*) concatenated with a.

Example 5. We define $P_n = 1 + P_{n-1}$ if the first character of s matches c, and in case they do not match, we define $P_n = 0 + P_{n-1}$. For example, the number of occurrences of character a in string *ababa* is 1 ($\because$ a matches the first character in the string) plus the number of occurrences of a in *baba* (which is 2). But, the number of occurrences of character b in string *ababa* is 0 ($\because$ b does not the first character a in the string) plus the number of occurrences of b in *baba* (which is 2).

Example 6. We define $P_n = a[from] > 0$ **&&** P_{n-1}. For example, numbers in $\{1, 2, 3, 4, 5\}$ are all positive because $1 > 0$ and numbers in $\{2, 3, 4, 5\}$ are all positive. But, numbers in $\{-1, 2, 3, 4, 5\}$ are not all positive because $-1 > 0$ is *false*, even though and numbers in $\{2, 3, 4, 5\}$ are all positive. Also, numbers in $\{1, 2, -3, 4, 5\}$ are not all positive because numbers in $\{2, -3, 4, 5\}$ are not all positive, even though $1 > 0$ is *true*.

Example 7. We define $P_n = a[from] \leq a[from + 1]$ **&&** P_{n-1}. For example, say *from* is 0, then numbers in $\{1, 2, 2, 3, 4\}$ are sorted because $1 \leq 2$ and numbers in $\{2, 2, 3, 4\}$ are sorted. But, numbers in $\{1, -1, 2, 3, 4\}$ are not sorted because $1 \leq -1$ is $false$, even though numbers in $\{-1, 2, 3, 4\}$ are sorted. Also, numbers in $\{1, 2, 2, -1, 4\}$ are not sorted because numbers in $\{2, 2, -1, 4\}$ are not sorted, even though $2 \leq 2$ is *true*.

Example 8. We exploit the fact that array a is sorted: for each element in a, all elements to its left are smaller, whereas all elements to its right are larger. We calculate a middle index $m = \frac{from+to}{2}$ (where we have an integer division in Java, and this is mathematically equivalent to the calculation of its floor $\lfloor \frac{from+to}{2} \rfloor$), and compare $a[m]$ against the value k being searched. We define $P_n = true$ if $a[m] == k$ (i.e., it is found). If k is not found immediately but $k < a[m]$, then we know that if k exists, it must be to the left of $a[m]$: $P_n = P_{left}$. Symmetrically, if k is not found immediately but $k > a[m]$, then we know that if k exists, it must be to the right of $a[m]$: $P_n = P_{right}$.

Problem (P_n)	**Base Case(s)** ($P_0,\ P_1,\ P_2$)	**Recursive Solution(s) to Sub-Problem(s)** ($P_{n-1},\ P_{n-2}$)	**Solution**
$factorial(n)$	$P_0 = factorial(0) = 1$	$P_{n-1} = factorial(n-1)$	$n \times P_{n-1}$
$fib(n)$	$P_1 = fib(1) = 1$ $P_2 = fib(2) = 1$	$P_{n-1} = fib(n-1)$ $P_{n-2} = fib(n-2)$	$P_{n-1} + P_{n-2}$
$isP(s)$	$P_0 = isP(\texttt{""}) = true$ $P_1 = isP(\texttt{"a"}) = true$	$P_{n-2} = isP(s.substring(1, s.length() - 1))$	$s.charAt(0) == charAt(s.length() - 1)$ && P_{n-2}
$rev(s)$	$P_0 = rev(\texttt{""}) = \texttt{""}$ $P_1 = rev(\texttt{"a"}) = \texttt{"a"}$	$P_{n-1} = rev(s.substring(1, s.length()))$	$P_{n-1} + s.substring(0)$
$occ(s, c)$	$P_0 = occ(\texttt{""}, c) = 0$	$P_{n-1} = occ(s.substring(1, s.length()), c)$	$1 + P_{n-1}$ **if** $s.charAt(0)$ == c $0 + P_{n-1}$ **if** $s.charAt(0)$!= c
$allPosH(a,\ from,\ to)$	$P_0 = allPosH(a,\ from,\ to)$ $= true$ **if** $from > to$ $P_1 = allPosH(a,\ from,\ to)$ $= a[from] > 0$ **if** $from == to$	$P_{n-1} = allPosH(a,\ from + 1,\ to)$	$a[0] > 0$ && P_{n-1}
$isSortedH(a,\ from,\ to)$ $isSortedH(a,\ from,\ to)$	$P_0 = isSortedH(a,\ from,\ to)$ $= true$ **if** $from > to$ $P_1 = isSortedH(a,\ from,\ to)$ $= true$ **if** $from == to$	$P_{n-1} = isSortedH(a,\ from + 1,\ to)$	$a[from] \leq a[from + 1]$ && P_{n-1}
$binSearchH(a,\ from,\ to,\ k)$	$P_0 = binSearchH(a,\ from,\ to,\ k)$ $= false$ **if** $from > to$ $P_1 = binSearchH(a,\ from,\ to,\ k)$ $= a[from] == k$ **if** $from\ ==\ to$	$P_{left} = binSearchH(a,\ 0,\ \lfloor\frac{from\ +\ to}{2}\rfloor - 1,\ k)$ $P_{right} = binSearchH(a,\ \lfloor\frac{from\ +\ to}{2}\rfloor + 1,\ to,\ k)$	P_{left} **if** $k < a[\lfloor\frac{from\ +\ to}{2}\rfloor]$ P_{right} **if** $k > a[\lfloor\frac{from\ +\ to}{2}\rfloor]$ $true$ **if** $k == a[\lfloor\frac{from\ +\ to}{2}\rfloor]$