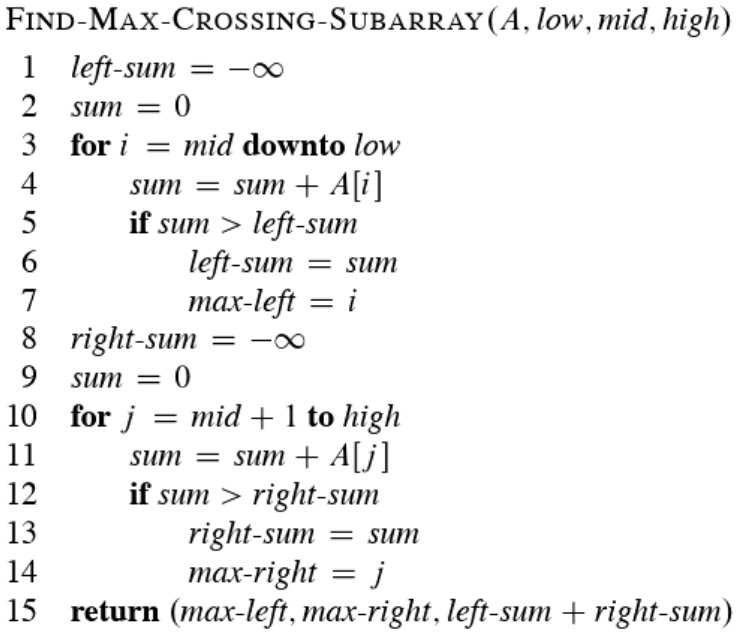
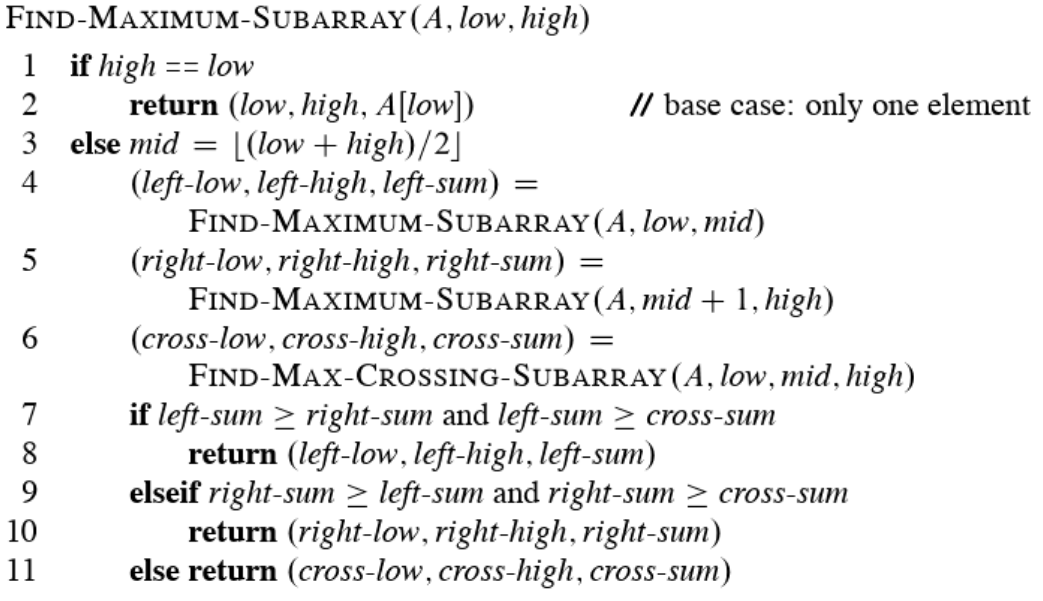
**CS 325 – Module 2 Notes – Divide and Conquer and Recursion**

* **Divide and conquer** solves a problem recursively, applying 3 steps at each level of recursion
  + **Divide** the problem into a number of subproblems that are smaller instances of the same problem
  + **Conquer** the subproblems by solving them recursively. If the subproblem’s sizes are small enough, however, just solve the subproblems in a straightforward manner
  + **Combine** the solutions to the subproblems into the solution for the original problem
* **Recursive case** is when the subproblems are large enough to solve recursively
* Once they are small enough to no longer need recursion, we say the recursion “bottoms out” and we have gotten down to the **base case.**
* Subproblems are not constrained to being a constant fraction of the original problem size. For example, a recursive version of linear search would create just one subproblem. Each recursive call would take constant time plus the time for the recursive call it makes, yielding the recurrence **T(n) = T(n – 1) + θ(1)**
* We will look at 3 different methods for solving recurrences
  + **substitution method** – we guess a bound and then use the mathematical induction to prove our guess correct
  + **recursion tree-method** – converts the recurrence into a tree whose nodes represent the costs incurred at various levels of the recursion. We use techniques for bounding summations to solve the recurrence.
  + **master method –** provides bounds for recurrences of the form ***T(n) = aT(n/b) + f(n)***, where ***a ≥* 1, *b > 1*, and *f(n­)* is a given function.** 
    - a recurrence in the form of the one above characterizes a divide-and-conquer algorithm that creates ***a*** subproblems, each of which is ***1/b*** the size of the original problem, and in which the divide and combine steps together take ***f(n)*** time.
    - to use the **master method**, you will need to memorize three cases, but once you do that, you will easily be able to determine asymptotic bounds for many simple recurrences. Can be used for **maximum-subarray** problems
* when we state and solve recurrences, we often omit floors, ceilings, and boundary conditions
  + they are generally determined whether or not they matter later on
* **maximum contiguous subarray sum** – finding a nonempty contiguous subarray of a given array (*A*) whose values have the largest sum
  + there may be more than one in some instances and this is **generally used only when the array contains some negative numbers**. If the entire array consisted of all positive numbers, then the max-subarray would present no challenge since the entire array would give the greatest sum
* **Using divide-and-conquer to solve a max-subarray problem…**
  + lets say we want to find a max-subarray of the subarray ***A[low..high]***. Divide-and-conquer suggests that we divide the subarray into two subarrays of as equal size as possible. That is, we find the midpoint (***mid***) and consider the subarrays ***A[low..mid]*** and ***A[mid + 1..high]***
  + any contiguous subarray ***A[i..j]*** of ***A[low..high]*** must lie in exactly one of the following places…
    - entirely in the subarray ***A[low..mid]***, so that ***low ≤ i ≤ j ≤ mid***
    - entirely in the subarray ***A[mid + 1..high]***, so that ***mid ≤ i ≤ j ≤ high***
    - crossing the midpoint, so that ***low ≤ i ≤ j ≤ mid < j ≤ high***
  + below is the method for **find-max-crossing-subarray** and **find-max-subarray**





* **See max-subarray program breakdown on pgs. 71-73**
* **Recurrences Solutions**
  + **T(n) = T(n – 1) + n - θ(n2**)
    - Recursive algorithm that loops though the entire input to eliminate one item
  + **T(n) = T(n/2) + c - θ(log n)**
    - recursive algorithm that halves the input in one step
  + **T(n) = T(n/2) + n - θ(n)**
    - Recursive algorithm that halves the input but must examine every item in the input
  + **T(n) = 2T(n/2) + 1 - θ(n)**
    - Recursive algorithm that splits the input into 2 halves and does a constant amount of other work
* [Recursive Example 1: Making Postage - OSU MediaSpace (oregonstate.edu)](https://media.oregonstate.edu/media/t/0_lu4bv9ha)