Lecture 2: Structure of Computation

Review

- Problem, Algorithm, Efficiency, Model of Computation, Data Structure
- How to Solve an Algorithms Problem
  - Reduce to a problem you already know (use data structure or algorithm)
  - Design your own (recursive) algorithm

Class	Graph	Visited
Brute Force	Star	All
Decrease & Conquer	Chain	All
Divide & Conquer	Tree	All
Dynamic Programming	DAG	All
Greedy / Incremental	DAG	Some

Contains

- **Problem**: Given an array A of n integers, does it contain integer v?
- **Example**: Seven element array.

Brute Force $O(n)$

```python
def contains(A, v):
    for i in range(len(A)):
        if A[i] == v: return True
    return False
```

Decrease & Conquer $O(n)$

```python
def contains(A, v, i = 0):
    if i == len(A):   return False
    if A[i] == v:     return True
    return contains(A, v, i + 1)
```

Divide & Conquer $O(n)$

```python
def contains(A, v, i = 0, j = None):
    if j is None:  j = len(A)
    if j - i == 0: return False
    c = (i + j) // 2
    if A[c] == v:  return True
    left  = contains(A, v, i, c)
    right = contains(A, v, c + 1, j)
    return left or right
```

Greedy, if sorted A, $O(\log n)$

```python
def contains(A, v, i = 0, j = None):
    if j is None:  j = len(A)
    if j - i == 0: return False
    c = (i + j) // 2
    if A[c] == v:  return True
    if A[c] > v:
        return contains(A, v, i, c)
    return contains(A, v, c + 1, j)
```

Max Subarray Sum

- **Problem:** Given an array A of n integers, what is the largest sum of any nonempty subarray? (in this class, **subarray** always means a contiguous sequence of elements)
- **Example:** `A = [-9, 1, -5, 4, 3, -6, 7, 8, -2]`, largest sum is 16.
- Brute Force:
  - # subarrays: $\binom{n}{2} + \binom{n}{1} = O(n^2)$
  - Can compute subarray sum of k elements in $O(k)$ time
  - n subarrays have 1 element, $n-1$ have 2, ... , 1 has n elements
  - Work is $c\sum_{k=1}^{n}(n-k+1)k = cn(n+1)(n+2)/6 = O(n^3)$
  - Graph: single node, or quadratic branching star, each with linear work

```python
def MSS(A):
    m = A[0]
    for j in range(1, len(A) + 1):
        for i in range(0, j):
            m = max(m, SS(A, i, j))
    return m
```

```python
def SS(A, i, j):
    s = 0
    for k in range(i, j):
        s += A[k]
    return s
.
```

- Divide & Conquer
  - Max subarray is either:
    1. fully in left half,
    2. fully in right half,
    3. or contains elements from both halves.
  - Third case $=$ (`MSS_EA`, ending at middle) $+$ (`MSS_SA`, starting at middle)
  - Combine step takes linear time
  - Graph is binary tree with linear work at each vertex
  - $T(n) = 2\,T(n/2) + O(n) \implies T(n) = O(n \log n)$
  - (Draw tree, Master Theorem will also be discussed in recitation)

```python
def MSS(A, i = 0, j = None):
    if j is None:  j = len(A)
    if j - i == 1: return A[i]
    c = (i + j) // 2
    return max(MSS(A, i, c), MSS(A, c, j),
        MSS_EA(A, i, c) + MSS_SA(A, c, j))
```

```python
def MSS_SA(A, i, j):
    s = m = A[i]
    for k in range(1, j - i):
        s += A[i + k]
        m = max(s, m)
    return m
```

```python
def MSS_EA(A, i, j):
    s = m = A[j - 1]
    for k in range(1, j - i):
        s += A[j - 1 - k]
        m = max(s, m)
    return m
```

- Dynamic Programming
  - `MSS_EA(A, 0, k)` finds largest subarray in A ending at k
  - `MSS` must end somewhere, so check `MSS_EA(A, 0, k)` for all k. (Brute Force)

```python
def MSS(A):
    m = A[0]
    for k in range(len(A)):
        s = MSS_EA(A, 0, k + 1)
        m = max(m, s)
    return m
```

  - But takes $c\sum_{k=1}^{n} k = cn(n+1)/2 = O(n^2)$ time.
  - Computing a lot of subarray sums; can we reuse any work?
  - Let's rewrite `MSS_EA` recursively

```python
def MSS_EA(A, i, j):
    if j - i == 1:  return A[i]
    return A[j - 1] + max(0, MSS_EA(A, i, j - 1))
```

  - Graph of function calls is a tree with $O(n^2)$ nodes
  - Same function called many times!
  - Redraw call graph as a DAG of overlapping problems.
  - Only $O(n)$ nodes!
  - Dynamic programming: remember work done before, or compute from bottom up

```python
def MSS(A):
    m = mss_ea = A[0]
    for i in range(1, len(A)):
        mss_ea = A[i] + max(0, mss_ea)
        m = max(m, mss_ea)
    return m
```