Week 6.1, Monday, Sept 23

Homework 3 Due Tonight: 11:59PM on Gradescope
Late Submissions: Close tomorrow night at 11:59PM on Gradescope
Practice Midterm 1: Solutions Released
Midterm Review Session: Tuesday (7:30-9:30PM) @ WALC 1018
Midterm 1: September 25 (evening)
Announcements

- No PSOs this week (due to Midterm)

- Midterm review on Tuesday night
  - WALC 1018 (7:30-9:30PM)

- Yes, we do have class on Wednesday
  - Classes canceled on October 28th and Dec 6th
  - Make up for two evening midterm exams

- We will release homework 3 solutions on Wednesday morning
  - At which point the submission server closes
  - No 2 day late submissions
Midterm 1: Logistics

- 90 minutes (8:00-9:30PM)
  - Tuesday/Thursday PSOs: SMTH 108  (Exam Capacity =115)
  - Friday PSO: MTHW 210  (Exam Capacity = 111)

- 1 Page of Notes (Single-Sided)

- Standard paper (or A4) is acceptable

- Bring number 2 pencil (for scanned exam)

- Closed book, no calculators, no smartphones, no smartwatches, no laptops etc…
Practice Midterm Solutions Released Soon
- Advice: Try to solve each problem yourself before checking answers

Topics:
- Induction
- Big-O
- Divide and Conquer
  - Sorting, Counting Inversions, Maximum Subarray, Skyline Problem, Karatsuba Multiplication
- Recurrences
  - Deriving a Recurrence
  - Unrolling
  - Recursion Trees
  - Master Theorem
- Greedy Algorithms

No Dynamic Programming Required (until Midterm 2)
Problem 1: Non Adjacent Selection (NAS)

S is an array of size n (positive integers in arbitrary order)
Select entries in S so that
i. the sum of the selected entries is a maximum
ii. no two selected entries are adjacent in array S

Examples
[14, 6, 33, 1, 2, 8]
[1, 4, 5, 4]
[15, 14, 10, 17, 10]
Approaches ...

Naive approach: Consider all possibilities of selecting entries

- If $S[i]$ is chosen, the two adjacent locations cannot be chosen
- There here is an exponential number possible solutions
Approaches ...

Naive approach: Consider all possibilities of selecting entries
  • If S[i] is chosen, the two adjacent locations cannot be chosen
  • There here is an exponential number possible solutions

Greedy: Create a sorted list of entries and choose entries from this order, skipping entries causing a violation in S
  • Correctness?
**Clicker Question**

**Greedy:** Create a sorted list of entries and choose entries from this order, skipping entries causing a violation in $S$

The Greedy algorithm fails to output the optimal solution on which of the following inputs?

A.  [14, 6, 33, 1, 2, 8]  
B.  [1, 4, 5, 4]  
C.  [7, 63, 64, 63, 2, 8]  
D.  B and C  
E.  All of the above
**Clicker Question**

**Greedy:** Create a sorted list of entries and choose entries from this order, skipping entries causing a violation in S

The Greedy algorithm fails to output the optimal solution on which of the following inputs?

A. \([14, 6, 33, 1, 2, 8]\)  (Greedy is Optimal)
B. \([1, 4, 5, 4]\)  vs  \([1, 4, 5, 4]\)
C. \([7, 63, 64, 63, 2, 8]\)  vs  \([7, 63, 64, 63, 2, 8]\)
D. B and C
E. All of the above
Approaches ...

Naive approach: Consider all possibilities of selecting entries
- If $S[i]$ is chosen, the two adjacent locations cannot be chosen
- There here is an exponential number possible solutions

Greedy: Create a sorted list of entries and choose entries from this order, skipping entries causing a violation in $S$
- Easy to find a counterexample

Use divide and conquer? How to combine?
- Recurse on arrays of size $n/2$ and then combine
- $[1, 4, 5, 3]$
- returns 4 and 5, respectively
Approaches ...

Naive approach: Consider all possibilities of selecting entries

- If S[i] is chosen, the two adjacent locations cannot be chosen
- There here is an exponential number possible solutions

Use divide and conquer? How to combine?

- Recurse on arrays of size n/2 and then combine
- Solve([1, 4, 5, 3])
  - Left: Solve([1, 4]) returns 4
  - Right: Solve([5, 3]) returns 5
- Combine? [1, 4, 5, 3]
- Can build D&C algorithm, but it is complicated…
Algorithmic Paradigms

**Greedy.** Build up a solution incrementally, myopically optimizing some local criterion.

**Divide-and-conquer.** Break up a problem into sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form solution to original problem.

**Dynamic programming.** Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems.
**Dynamic Programming History**

**Bellman.** [1950s] Pioneered the systematic study of dynamic programming.

**Etymology.**
- Dynamic programming = planning over time.
- Secretary of Defense was hostile to mathematical research.
- Bellman sought an impressive name to avoid confrontation.

"it's impossible to use dynamic in a pejorative sense"
"something not even a Congressman could object to"

Let’s try something else

S[1], S[2], S[3], S[4], ..., S[n-2], S[n-1], S[n]

When is the nth element selected?

- Depends on what optimum solutions look like on elements 1 to n-1
  - If optimal solution does not include S[n-1] then we can add S[n] to the solution
  - What if the optimal solution does include S[n-1]?
Let’s try something else

S[1], S[2], S[3], S[4], … , S[n-2], S[n-1], S[n]

When is the $n^{th}$ element selected?
• Depends on what optimum solutions look like on elements 1 to n-1

Let $OPT(k)$ be the optimum solution in subarray $S[1:k]$
Assume we know $OPT(n-2)$ and $OPT(n-1)$
Then, $OPT(n) = \max\{OPT(n-1), OPT(n-2) + S[n]\}$
Let’s try something else

S[1], S[2], S[3], S[4], … , S[n-2], S[n-1], S[n]

Let OPT(k) be the optimum solution in subarray S[1:k]
Assume we know OPT(n-2) and OPT(n-1)
Then, \( OPT(n) = \max\{OPT(n-1), OPT(n-2) + S[n]\} \)

**Case 1:** Optimal does not use \( S[n] \)

\( \Rightarrow \) Use optimal solution for subarray \( S[1:n-1] \) (\( OPT(n-1) \))
Let’s try something else

S[1], S[2], S[3], S[4], … , S[n-2], S[n-1], S[n]

Let OPT(k) be the optimum solution in subarray S[1:k]
Assume we know OPT(n-2) and OPT(n-1)
Then, \( \text{OPT}(n) = \max\{\text{OPT}(n-1), \text{OPT}(n-2) + S[n]\} \)

**Case 2:** Optimal solutions includes S[n]

\( \rightarrow \) Cannot use S[n-1]

\( \rightarrow \) add S[n] to optimal solution on subarray S[1:n-2]
The DP Recurrence Relationship

\[ \text{OPT}(n) = \max \{ \text{OPT}(n-1), \text{OPT}(n-2) + S[n] \} \]

\[ \text{OPT}[1] = S[1] \]
\[ \text{OPT}[2] = \max \{ \text{OPT}(1), S[2] \} \]
\[ \text{OPT}[k] = \max \{ \text{OPT}(k-1), \text{OPT}(k-2) + S[k] \}, \quad 3 \leq k \leq n \]

**Case 1:** Optimal solution to sub-problem \( S[1:k] \) does not include \( S[k] \)

\[ \Rightarrow \text{use optimal solution to } S[1:k-1] \]

**Case 2:** Optimal solution to sub-problem \( S[1:k] \) includes \( S[k] \)

\[ \Rightarrow \text{add } S[k] \text{ to optimal solution to } S[1:k-2] \]
Now we have an efficient algorithm

\[
\text{OPT}(n) = \max \{ \text{OPT}(n-1), \text{OPT}(n-2) + S[n] \}
\]

\[
\text{OPT}[1] = S[1]
\]
\[
\text{OPT}[2] = \max \{ \text{OPT}(1), S[2] \}
\]
\[
\text{OPT}[k] = \max \{ \text{OPT}(k-1), \text{OPT}(k-2) + S[k] \}, \quad 3 \leq k \leq n
\]

Compute entries of array OPT in \(O(n)\) time in one left to right scan (at position \(k\), look at \(k-1\) and \(k-2\))

\[
S = [14, 6, 8, 9, 7, 2]
\]
Now we have an efficient algorithm

OPT[1] = S[1]
OPT[2] = max{OPT(1), S[2]}
OPT[k] = max{OPT(k-1), OPT(k-2) + S[k]}, 3 \leq k \leq n

How do we determine the elements selected?

- Once OPT[n] is known, use the entries in array OPT to construct the answer
Start at n scanning left and determine elements in set T

T = {} ; k = n

while k ≥ 1

    if OPT[k-1] ≥ OPT[k-2] + S[k] then
        k = k-1  // S[k] is not selected
    else
        add index k to set T;
        k = k-2

Return T
Start at \( n \) scanning left and determine elements in set \( T \)

\[
T = \{ \} ; \ k = n \\
\textbf{while} \ k \geq 3 \\
\quad \textbf{if} \ \text{OPT}[k-1] \geq \text{OPT}[k-2] + S[k] \\
\quad \quad \textbf{then} \ k = k-1 \quad \text{//} \ S[k] \text{ is not selected} \\
\quad \textbf{else} \ \text{add index } k \text{ to set } T; \ k = k-2 \\
\textbf{if} \ (T \text{ contains 3 or } S[1] > S[2]) \ \textbf{then} \ \text{add index 1 to set } T \\
\quad \text{Else add index 2 to set } T \\
\text{Return } T
\]

Generating the elements in the solution costs \( O(n) \) time

Note: Revisit the \( O(n) \) time iterative solution to maximum subarray problem (it is DP)