Announcements

- Program resubmissions are ongoing; check the D2L announcement for details
- Final exam is on May 13th from 3:30pm-5:30pm. Please bring an ID.
- There will be a final exam review session…details coming soon!

Topic 11: Review

- We'll be reviewing a variety of selected topics
- Nothing "new," but… we'll cover the Maximum Contiguous Subsequence Sum Problem (as a way of reviewing step-counting and asymptocity)

Quicksort Partitioning Review

58 46 53 28 32 73 23 15 24 88 42 33 57	*Using the "Median of Three"*
Select 57 as the pivot, and swap with 58	*approach.*
57 46 53 28 32 73 23 15 24 88 42 33 58	*Start with L and R at the second and*
L *R*	*final indices*
57 46 53 28 32 73 23 15 24 88 42 33 58	*Move them until L's element is >*
L *R*	*pivot, and R's element is < pivot*
57 46 53 28 32 33 23 15 24 88 42 73 58	*Swap R and L's elements*
L R	*Identify the next swapping point.*
57 46 53 28 32 33 23 15 24 42 88 73 58	*Swap again*
R L	*Identify the next swapping point.*
42 46 53 28 32 33 23 15 24 57 88 73 58	*But R < L, so swap the pivot with R*

Find the Pattern Practice

- Find the closed-form expression for the recurrence relation R(n) = R(n - 1) + n, where R(0) = c.
- R(n - 1) = R(n – 2) + n - 1
- R(n - 2) = R(n – 3) + n – 2
- R(n – 3) = R(n – 4) + n - 3
- R(n) = R(n – 2) + n + (n – 1)
- R(n) = R(n – 3) + n + (n – 1) + (n – 2)
- R(n) = R(n – 4) + n + (n – 1) + (n – 2) + (n – 3)
- R(n) = R(n – a) + $\sum_{i=0}^{a} n - \mathrm{i}$
- n - a = 0, n = a
- R(n) = c + n * (n + 1) / 2

Inductive Proof Practice

- Prove that for the recurrence relation R(n) = 5 R(n - 1) + k, where R(1) = c, k is some constant, the equivalent closed-form expression is: R(n) = $c * 5^{n-1} + \frac{k(5^{n-1}-1)}{4}$
- Conjecture: The equivalent closed-form expression for the recurrence relation above is: R(n) = $c * 5^{n-1} + \frac{k(5^{n-1}-1)}{4}$
- **Base Case**: R(1) = c (given) and R(1) = $c * 5^{1-1} + 0 * \frac{k}{4}$.

Inductive Proof Practice

- Prove that for the recurrence relation R(n) = 5 R(n - 1) + k, where R(1) = c, k is some constant, the equivalent closed-form expression is: R(n) = $c * 5^{n-1} + \frac{k(5^{n-1}-1)}{4}$
- **Inductive Case:** Assume conjecture holds for some value of n. Must show it holds for n + 1. R(n+1) = 5 * R(n) + k.
- By IH, R(n+1) = $5 * \left(c * 5^{n-1} + \frac{k(5^{n-1}-1)}{4}\right) + k$
- = $\left(c * 5^{n} + \frac{k(5^{n}-5)}{4}\right) + k$
- = $\left(c * 5^{n} + \frac{k(5^{n}-5)+4k}{4}\right)$
- =$\left(c * 5^{n} + \frac{k(5^{n}-1)}{4}\right)$ QED.

Topics 9 and 10 “Quiz”

- What algorithm family would each of these algorithms fall under, if any: Quicksort, Kruskal’s, DFS, Bellman-Ford?
- If a potential solution to Problem A can be verified in polynomial time, is it guaranteed that Problem A can be solved in polynomial time? If so why, and if not, why not?

Topics 9 and 10 “Quiz”

- What algorithm family would each of these algorithms fall under, if any: Quicksort, Kruskal’s, DFS, Bellman-Ford?
- Quicksort is Divide and Conquer, Kruskal’s is greedy, DFS is none, Bellman-Ford is dynamic programming
- If a potential solution to Problem A can be verified in polynomial time, is it guaranteed that Problem A can be solved in polynomial time? If so why, and if not, why not?
- No, it is not guaranteed. The supposition is that A is in NP, but P may be a subset of NP rather than equal to it, so there may not be an algorithm that is can solve Problem A in polynomial time.

The Maximum Contiguous Subsequence Sum Problem

- The Problem: Given integers $a_1, a_2, \ldots, a_n$, find the maximum value of $\sum_{k=i}^{j} a_k$, where $0 \leq i, j \leq n$. If all values in the sequence are negative, the sum is 0 (subsequences may be empty).
- Given this definition, what is the MCSS of the list below?
- [-6, 4, 2, -3, 4, -4, 5, -3, 2]
- MCSS: 8
- The sequence? [4, 2, -3, 4, -4, 5]

The MCSS Problem

- Algorithm #1: Exhaustive Search (credit: Ulf Grenander)
- Idea: Compute sums of all possible subranges i…j.

```
 1  public static int maxSubsequenceSumV1 (int[] list, int n)
 2  {
 3      int thisSum, maxSum = 0;
 4
 5      for (int i=0; i<n; i++) {
 6          for (int j=i; j<n; j++) {
 7              thisSum = 0;
 8              for (int k=i; k<=j; k++) {
 9                  thisSum += list[k];
10              }
11              if (thisSum > maxSum) maxSum = thisSum;
12          }
13      }
14      return maxSum;
15  }
```

The MCSS Problem

- Analysis of Algorithm #1: Abbreviated step-counting!
- Each loop can execute a maximum of *n* times. More precisely:

$$\begin{aligned}
\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j} 1 &= \sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1) \\
&= \sum_{i=0}^{n-1}\frac{(n-i+1)(n-i)}{2} \\
&= \frac{n^3+3n^2+2n}{6} \\
&\in O(n^3)
\end{aligned}$$

The MCSS Problem

- Algorithm #2: Smarter Exhaustive Search (credit: Ulf Grenander)
- Idea: Consider each lower endpoint (i) just once.

```
 1  public static int maxSubsequenceSumV2 (int[] list, int n)
 2  {
 3      int thisSum, maxSum = 0;
 4
 5      for (int i=0; i<n; i++) {
 6          thisSum = 0;
 7          for (int j=i; j<n; j++) {
 8              thisSum += list[j];
 9              if (thisSum > maxSum) maxSum = thisSum;
10          }
11      }
12      return maxSum;
13  }
```

The MCSS Problem

- Analysis of Algorithm #2: Let's show the details this time.
- Our nested-sum step-counting expression is: $\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} 1$

$$\sum_{j=i}^{n-1} 1 = [(n-1)-i]+1 = n-i$$

$$\sum_{i=0}^{n-1}(n-i) = n+(n-1)+\cdots+(n-(n-1))$$

$$= \sum_{k=1}^{n} k$$

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

$$\in O(n^2)$$

The MCSS Problem

- Algorithm #3: Divide and Conquer (credit: Michael Shamos)
- First: What are the three steps of Divide and Conquer algorithms?
- Let's go back to our starting list: [-6, 4, 2, -3, 4, -4, 5, -3, 2]
- Consider splitting the list in half: [-6, 4, 2, -3, 4] and [-4, 5, -3, 2]
- There are three cases: the MCSS is entirely in the left subproblem, entirely in the right subproblem, or it's split between them (or "straddled" between them)
- For "straddle" case, have to find largest sum that ends at the right-most index in the left half, and starts at the left-most index in the right half

The MCSS Problem

- Algorithm #3: Divide and Conquer (credit: Michael Shamos)
- [-6, 4, 2, -3, 4, -4, 5, -3, 2]
- Straddle case: suppose we're combining our last two split subarrays, [-6, 4, 2, -3, 4] and [-4, 5, -3, 2]
- For the left subarray, we start at 4 and try to add to the sum by moving down the array, keeping track of our current sum, and the best sum (with it's associated index) we've found.
- **4** -> -3 -> 2 -> **4** -> 6
- For the right subarray, we start at -4 and try to add to the sum by moving up the array similarly.
- -4 -> **5** -> -3 -> 2

The MCSS Problem

- Analysis of Algorithm #3: No code this time!

$$T(1) = 1$$

$$T(n) = 2T(n/2) + n$$ the straddle case is linear

Let's practice finding the pattern!

$$= 2^k T(n/2^k) + kn \quad \text{where } k = \log_2 n$$

$$= nT(1) + n \log_2 n$$

$$\in O(n \log_2 n)$$

The MCSS Problem

- Algorithm #4: Work Smarter, Not Harder! (credit: Jay Kadane)
- Idea: Extend the fixed-endpoint straddling idea to discard subsequences that have a non-positive sum
- [-6, 4, 2, -3, 4, -4, 5, -3, 2]
- Start with best_sum = -inf, current_sum = 0, start looping through the array.
- current_sum = max(list[i], current_sum+list[i])
- best_sum = max(best_sum, current_sum)

The MCSS Problem

- [-6, 4, 2, -3, 4, -4, 5, -3, 2]
- current_sum = max(list[i], current_sum+list[i])
- best_sum = max(best_sum, current_sum)
- current_sum = 0, best_sum = -inf, list[i] = -6
- current_sum = -6, best_sum = -6, list[i] = 4
- current_sum = 4, best_sum = 4, list[i] = 2
- current_sum = 6, best_sum = 6, list[i] = -3
- current_sum = 3, best_sum = 6, list[i] = 4
- current_sum = 7, best_sum = 7, and so on…

The MCSS Problem

- Analysis of Algorithm #4: This is obviously linear, but let's look at the sum anyway

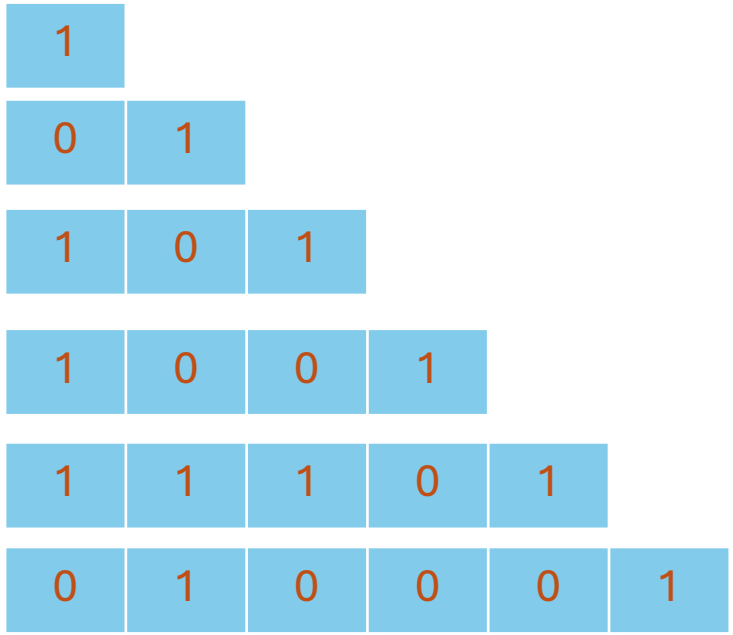
$$\sum_{j=0}^{n-1} 1 = [(n-1)-0]+1 = n \in O(n)$$

Array Storage Review

- Suppose we have a 3D array. What would the equation for calculating an element's address be if we're operating under RMO? What about CMO?
- RMO_3d_address[page_index][row_index][column_index] = base address + esize * (NUMROWS * NUMCOLS * page_index + NUMCOLS * row_index + column_index)
- CMO_3d_address[page_index][column_index][row_index] = base address + esize * (NUMROWS * NUMCOLS * page_index + NUMROWS * column_index + row_index)

Array Storage Review

- Suppose we have a lower left triangular array. What would the equation for calculating an element's address be?
- Lower_left_tri_address[row_index][column_index] = base address + esize * ((row_index + 1) * row_index / 2 + column_index)



Asymptocity Review

- Definition: *Size-x (?())*
- *Let f(n) and g(n) be functions that map positive integers to positive reals. We say that f(n) is ?(g(n)) if ____ real constant c > 0, there exists an integer constant* $n_0 \geq 1$ *such that* $f(n)\_\_c * g(n)$ *for every integer* $n \geq n_0$.

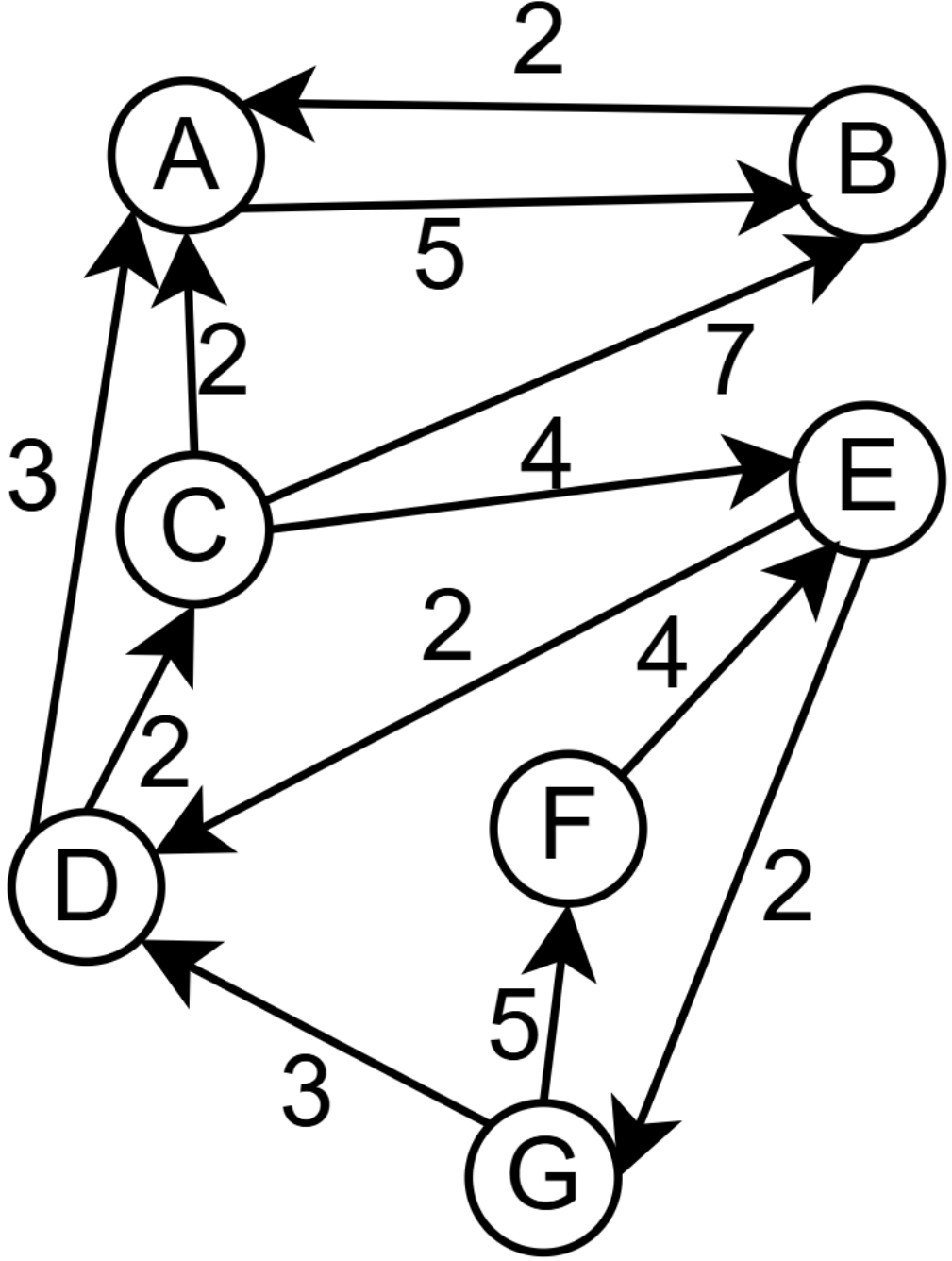
- We have big-O, little-o, Big-Omega, little-Omega, and big-Theta. Replace the blanks in the generic definition above for each.
- Little-O: “for any” and $<$
- Big-O: “there exists” and $\leq$
- Big-Omega: “there exists” and $\geq$
- Little-Omega: “for any” and $>$
- Big Theta… doesn’t fit the generic definition above.

Asymptoticity Review

- **Symmetry of Θ**: $f(n) \in \Theta(g(n))$ iff $g(n) \in \Theta(f(n))$
- **Transpose Symmetry**: $f(n) \in O(g(n))$ iff $g(n) \in \Omega(f(n))$ **and** $f(n) \in o(g(n))$ iff $g(n) \in \omega(f(n))$
- **Reflexivity**: for X=O, Ω, and Θ, f(n) is big-X(f(n))
- Why doesn't reflexivity work for little-o or little-ω?
- **Transitivity**: for X=O, o, Ω, ω, and Θ, if $f(n) \in X(g(n))$ and $g(n) \in X(h(n))$, then $f(n) \in X(h(n))$
- For example, $5n + 4 \in o(n(\log(n))$ and $n\log(n) \in o(n^2)$, so $5n + 4 \in o(n^2)$

Dijkstra's Review

Known	Fringe
$F_{0,-}$	$E_{4,F}$
$E_{4,F}$	$D_{6,E}$ $G_{6,E}$
$D_{6,E}$	$G_{6,E}$ $C_{8,D}$ $A_{9,D}$
$G_{6,E}$	$C_{8,D}$ $A_{9,D}$
$C_{8,D}$	$A_{9,D}$ $B_{15,C}$
$A_{9,D}$	$B_{14,A}$
$B_{14,A}$	---



Quadratic Probing

- Quadratic probing general function: $qp(key, i) = (h(key) + c * i + d * i^2) \% m$
- Let's do a specific example. Let $h(key) = key \% 7$, c = 3, d = 2
- $qp(key, i) = (key \% 7 + 3 * i + 2 * i^2) \% 7$
- Let's insert 7, 14, 21

0	1	2	3	4	5	6
7					14	21

qp(7, 0) = 0
qp(14, 0) = 0 (collision!)
qp(14, 1) = 5
qp(21, 0) = 0 (collision!)
qp(21, 1) = 5 (collision!)

qp(21, 2) = 0 (collision!)
qp(21, 3) = 6