28 ← no change

Name(last, fir

UCLA Computer Science Department

CS 180 **Algorithms & Complexity**

Midterm **Total Time: 1.5 hours** **October 31, 2017**

Each problem has 20 points.

All algorithm should be described in English, bullet-by-bullet

13 1 Consider a set of intervals I1, …, In. **a.** Design a linear time algorithm (assume intervals are sorted in any manner you wish) that finds a maximum subset of mutually non-overlapping intervals. **b.** Prove the correctness of your algorithm.

sort by ?

a)
- Add all intervals to a set of possible intervals P → O(n)
- Select the first-ending interval, (called "v") within P, add it to a set S of selected intervals, remove from P → O(n)
- Remove any intervals from P that overlap with v → O(n)
- Continue from bullet 2 until no more intervals exist in P. S will be filled with the max subset of non-overlapping intervals. → O(n)

→ All computations are some constant × n, therefore O(n).

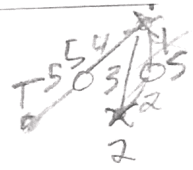
b) i: No overlapping intervals in S
- Assume intervals i and j overlap and are in S
- This means, say, i was added to S, then j was selected and added to S.
- Once i was added, all overlapping intervals (including j) were removed from P, so they couldn't be considered for S — a contradiction.

ii: S contains maximum subset
- Assume there exists a set Y where |Y| = |S| + 1 and Y contains non-overlapping intervals
  - ↳ this means there is an interval i that was unaccounted for in S, so it must overlap with a previous value in S, since S only contains values from P, which are guaranteed to not overlap with any previously selected intervals.
  - ↳ this means Y contains an interval i that indeed overlaps — a contradiction!

7

Name(

2 . a. Design an efficient algorithm better than O (n^2) to be used in sparse graphs for finding the shortest path between two vertices S and T in a positive weighted graph. **b.** Justify the correctness of your runtime analysis.



a) - Start at S, set all other vertex "distances" to ∞, v=S
- Build a minheap from the weights of all edges, starting from v
- Select the root (min) of the minheap, add that value to the node w incident to visited node v
- Heapify the minheap, run it again until T is found

-12.

b) - Building the minheap takes O(logn) time
- this happens (worst case) e total times for the number of edges.
- Therefore, O(elogn)

-1

3

Name(

3. Consider a sequence of positive and negative (including zero) integers. Find a consecutive subset of these numbers whose sum is maximized. Assume the weight of an empty subset is zero. **a.** Design a linear time algorithm. **b.** Prove the correctness of your algorithm.

Example: For the sequence -3 5 -12 the maximum sum is 5

a) Set max = 0 → O(1)

- Loop through values → start at i = first value → O(n)
  - ↳ If i is greater than max, set max = i → O(1)
    - ↳ Else, continue loop
  - ↳ Set temp = max + next value (j)
    - ↳ if temp > max, max = temp
    - ↳ If j > max, set max = j and i = j, continuing → O(1)
      - loop from bullet 2.
      - ↳ Else, i = j and continue
- Return max → O(1)

↳ O(n) total

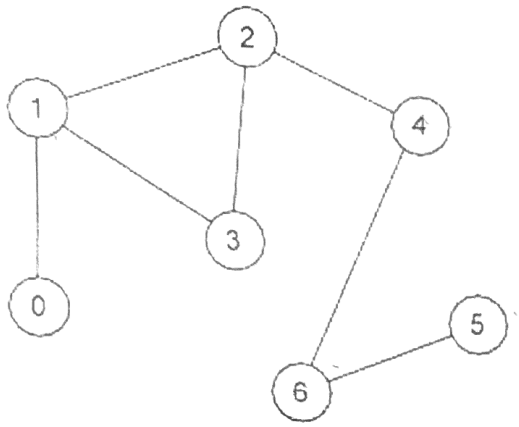
b) - Assume algorithm returned a sum < max

↳ Prove contradiction:

10

Name

4. Consider an unweighted graph G shown below. **a.** Starting from vertex **4,** show every step of DFS along with the corresponding stack next to it. **b**. What is the run time of DFS if the graph is not connected (no proof is necessary)?



b) Size of the component:
O(n)

a)

- Push 4 onto the stack, move to neighbor	"v"	Stack
	4	4
- Push 2 onto stack		2,4
- Push 1 onto stack		1,2,4
- Push 0 onto stack		0,1,2,4
- Explore 0, pop off stack		1,2,4
- Push 3 onto stack		3,1,2,4
- Explore 3, pop off stack		1,2,4
- Explore 1, pop off stack		2,4
- Explore 2, pop off stack		4
- Push 6		6,4
- Push 5		5,6,4
- Explore 5, pop		6,4
- Explore 6, pop		4
- Explore 4, pop		[]

5 Name(l

5. Consider a binary tree (it is not necessarily balanced). The tree is not rooted. Its diameter is the distance between two vertices that are furthest from each other (distance is measured by the number of edges in a simple path). Design a linear time algorithm that finds the diameter of a binary tree.

Essentially: find the longest path → BFS, but not rooted
↳ n-1 edges → max distance would be n-1
↳ start from node w/ lowest degree to maximize

Algorithm
- Select a node of degree 1, make it the root r
- Perform BFS from r, keeping track at L_0 $O(n+m)$
  of the number of levels in the resulting BFS tree → $O(n)$
- Return the bottom level of the BFS tree (i.e. if L_4 → return 4) → $O(1)$

↳ $O(n+m)$ total, which is linear