1. (20 points) Consider a set of intervals $I_1, I_2, \cdots, I_n$:
   (a) Design a linear time algorithm (assume that intervals are sorted in any manner you wish) that assigns the intervals to the minimum number of processors.
   (b) Prove the correctness of your algorithm.

a) Assume: The intervals are sorted by finishing time with the earliest finishing time first. ✗

There are n processors available.

Set I intervals w/ elements $I_1, I_2, I_3, \ldots I_n$

Algorithm:

- Pick the interval with the earliest finish time and assign it to processor n_1
- Go through the rest of the intervals, and if interval I_j's start time conflicts with the finish time of the interval before it I_i assign it to another processor.
  - Once an interval's finish time ends, free up the processor to be used for another interval.

b) • Assume greedy algorithm outputs A and the optimal solution is O
  - Implies that O used less processors than A
- Base Case: n=1 ; Both A and O used 1 processor
- Inductive:
  - For the 1st n-1 elements, A and O used the same number of processors.
  - This would mean that for O, on the last element n, one of the processors became free since an interval would have finished.
    - meaning that A would still have n-1 interval still running the nth element starts.
  - However, this contradicts our greedy algorithm of picking the earliest finish time.
    - The elements of A should finish before or at the same time as the corresponding element of O.

cont on back.

- Thus, due to the contradiction, A is the same as the optimal solution. (It will have the min. # of processors)

- Because of this the algorithm is correct.

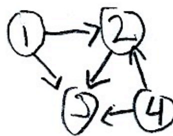
Time complexity

- After the sorting which takes $O(n\log n)$, the algorithm runs through the list once.
  - thus runs in $\boxed{O(n)}$ time

2. (20 points) (a) Design an efficient algorithm that outputs the vertices of a DAG (Directed Acylic Graph), such that if there is an edge (x, y) then x is output before y.

   (b) Analyze the run time of your algorithm.

n vertices, m edges



a) • Run topological sort on the DAG
- Once done output the vertices in order of the topological sort
  - This allows vertex x to be before y if there is edge (x, y)

need explanation!

3/10

b) Time Complexity:

- Topological sort runs in O(m+n)
- Running through the sorted graph takes O(n) time ✓ outputting the node at each iteration is constant time

Thus it runs in O(m+2n) or O(m+n) time

No breakdown of steps

3/10

3. (20 points) An undirected graph is said to have property X if you can start from a vertex, traverse all edges of the graph exactly once, without removing your pen from the paper.

   (a) Classify the graphs that have property X?. ?

   (b) Design an efficient algorithm for generating a traversal of a graph that has property X.

a) Graphs with property X are classified as Euler graphs since they have an Eulerian cycle

b) Run a modified DFS: There are n vertices and m edges

- Check degrees of each node
  - if there are more than 2 nodes of odd degrees, output "Graph does not have property X" ✓
  - if there are 2 nodes of odd degree, pick one as the starting node s
  - if there are no nodes of odd degree, pick one as the starting node s
- Run DFS from node s.
  - when adding nodes to the DFS stack, check if the edge leaving the current node has been traversed not if the adjacent node has been visited.
    - If the edge has not been traversed, add the adjacent node to the stack

−6

- After DFS has run:
  - check if there were any edges that were not traversed
    - if so, Output "Graph does not have property X"
  - Otherwise: Output "Graph has property X"

continued on back.

Time Complexity Analysis

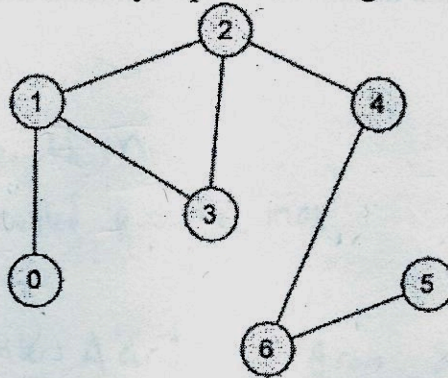
- With the correct data structures, finding the degree of each node will be $O(n)$ time
- Checking the conditions for each node degree will take $O(n)$
  - there are n vertices and the if statements are constant time
- DFS is used which takes $O(m+n)$ time
  - the change of checking if an edge was traversed rather than a vertex was visited is still constant time.

∴ Thus the time complexity is $\boxed{O(n+m)}$

4. (10 points) Consider an unweighted graph G shown below:

(a) Starting from vertex 1, show every step of BFS along with the corresponding FIFO next to it.

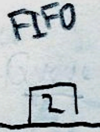
-2 wrong start node



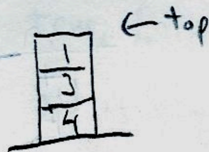
the top is the 1st in

FIFO

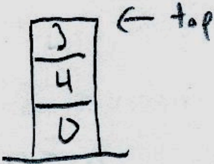
Step 1: Output



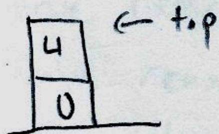
Step2:



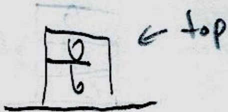
Step3:



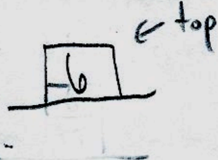
Step4:



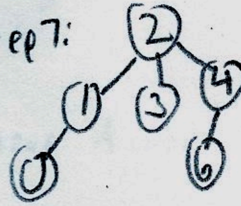
Step 5:

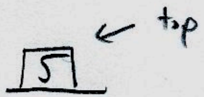


Step 6:

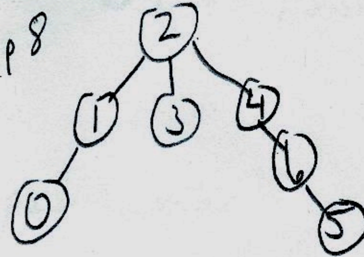


Step7:





Step 8



5. (20 points) Consider an unsorted list of integers. You can find the minimum number in the list with $n-1$ comparisons. Similarly, you can find the maximum with $n-1$ comparisons. So you can find both the minimum and the maximum with about $2n-3$ comparisons. Design an algorithm that finds both the minimum and the maximum using about $\frac{3n}{2}$ comparisons.

- Given list S of length n
- Create another list called possible_max
- Arbitrarily pick 2 numbers A and B from S
  - check if A > B
    - if true, leave B in S and move A into possible_max
    - if false, leave A in S and move B into possible_max
  - repeat until S only has 1 element
    - this is the minimum
- Arbitrarily pick 2 numbers C and D from possible_max
  - check if C > D
    - if true, remove D from possible_max and leave C in it
    - if false, remove C from possible_max and leave D
  - Repeat until possible_max has 1 element (the max value)
- Output S as the minimum value and output possible_max as the maximum value.

Proof: • The first comparison of A > B, will only leave the smaller values in S until only the smallest value remains.
- this takes n-1 comparisons

The second comparison is similar but will only leave the largest values in S
- this takes n-2 comparisons

. The total # of comparisons is 2n-3.

cont on back →

6. (10 points) Give an algorithm to color a graph with 2 colors (assuming it is 2-colorable). A proof of correctness is not necessary.

There are 2 colors: R and B for easier reference

- Run a modified version of BFS
- Assign start node s the color R when pushing it to the queue.
- when adding vertices v adjacent to current vertex u to the queue assign v a different color than u (ex: if u is R, v is B)
  - however, is trying to add a node v to the queue with the same color as the current node u, output that it is not 2-colorable
- Continue BFS as usual
- Output graph with the color assignments for the nodes

-4 Time Complexity