Ruirui Li
I forgot Section #

UCLA
Computer Science Department

CS180– Midterm
Algorithms & Complexity

10/30/2018

Name: [redacted]
UID:

This exam contains 7 pages (including this cover page) and 6 questions.

- Writing has to be legible.
- Express algorithms in bullet form, step by step.

Distribution of Marks

Question	Points	Score
1	20	20
2	20	14
3	20	7
4	10	10
5	20	20
6	10	6
Total:	100	77

20

Name(last, first): Midterm 10/30/2018

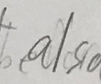
1. (20 points) Consider a set of intervals $I_1, I_2, \cdots, I_n$:
   (a) Design a linear time algorithm (assume that intervals are sorted in any manner you wish) that assigns the intervals to the minimum number of processors.
   (b) Prove the correctness of your algorithm.

✓

a.) • Assume that the intervals are sorted chronologically by start and end time (if I_1 has start S_1 and end F_1, I_2 has start S_2 and end F_2, I_n has start S_n and end F_n, then the sorted list is of the form similar to $S_1, F_1, S_2, S_3, F_2, F_3, \dots F_n$)

```
• Create two variables MAX and CURRENT and set both to 0
• For every element in the sorted list of start and end times of the intervals:
    • If the current element is a start time
        • increment CURRENT by one
    • Else if the current element is an end time
        • decrement CURRENT by one
    • If CURRENT is greater than MAX
        • Set MAX equal to current
• Return MAX, which contains the minimum number of processors
```

✓

b.) The minimum number of processors is equal to the maximum number of intervals that overlap at any point, if all intervals will be processed in parallel. Therefore we need to find this maximum overlap. Two intervals overlap if one interval starts after a second interval starts but also starts before the second interval ends (). Therefore, by incrementing and decrementing current each time we see a start or end time, we keep track of the number of intervals that overlap at any time. Storing the maximum value we see and updating it if needed gives us the minimum number of processors. Therefore the algorithm is correct.

✓

~~15~~ 14/20

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.

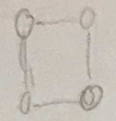
(b) Analyze the run time of your algorithm.

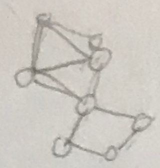
a.) We define a source node as having no incoming edges ✓
- Find a source node S by choosing an arbitrary node and moving backwards over incoming edges until we reach a node with no incoming edges. ✓
- Create an array A to hold our ordered vertices
- Run a topological sort over the DAG starting at S and store each sorted node in A
- Starting from the beginning of A, output every vertex in order

Missing 6/10

b.) Let n represent the number of nodes/vertices and let e represent the number of edges. Finding a source node is ~~O(n)~~ O(n+e) because in the worst case every node will be visited. Running the topological sort and storing the results takes O(e+n) time, because in an efficient topological sort, every node is visited and every edge is traversed once, and storing results in an array takes constant time. Iterating through the array at the end takes O(n) ✓ time because every node is stored in the array. Therefore the total runtime of the algorithm is $\boxed{O(e+n)}$.

7/10





7

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.) Connected graphs (graphs where every node can be reached from every other node via edges) in which the sum of the degrees of all nodes in the graph is even have the property X.

-7

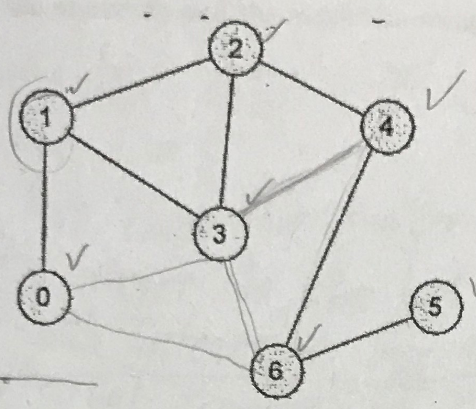
b.) • Search for a node S with odd degree, if one exists. If not, pick an arbitrary node
- Traverse an edge of the starting node, marking it as traversed and marking all the other edges from the node as seen
- For each node visited:
  - If there are no edges that have not been marked traversed, exit loop.
  - If there is an edge that has not been marked as seen:
    - Traverse that edge, marking it as or traversed
  - Else
    - Traverse an edge that has been marked as seen, marking it as traversed instead
- Mark all untraversed edges from the node that was just left as seen if they have not been marked already

-6

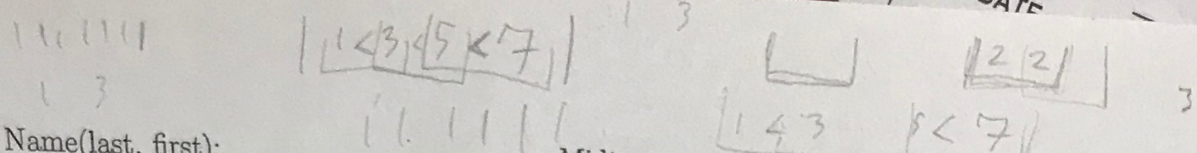
10

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.



1.) Push 1 onto FIFO [1]
2.) Pop 1 from FIFO, mark as visited [ ]
3.) Push 0, 3, and 2 onto FIFO [0|3|2]
4.) Pop 0 from FIFO, mark as visited [3|2]
5.) Pop 3 from FIFO, mark as visited [2]
6.) Pop 2 from FIFO, mark as visited [ ]
7.) Push 4 onto FIFO [4]
8.) Pop 4 from FIFO, mark as visited [ ]
9.) Push 6 onto FIFO. [6]
10.) Pop 6 from FIFO, mark as visited [ ]
11.) Push 5 onto FIFO [5]
12.) Pop 5 from FIFO, mark as visited [ ]
13.) No unvisited vertices, so the algorithm terminates



Name(last, first): Midterm 10/30/2018

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.

```
• Create two variables, MIN = ∞ and MAX = −∞
• Create a variable i = 1
• While i+1 is less than the number of integers in the list:
  ~~• if the ith integer is the last integer in the list:~~
    ~~• if the ith integer is less than MIN:~~
      ~~• set MIN equal to the ith integer~~
    ~~• else if the ith integer is greater than MAX:~~
      ~~• set MAX equal to the ith integer~~
  ~~• else~~
  • if the ith integer is less then the (i+1)th integer in the list
    • if the ith integer is less than MIN
      • MIN equals the ith integer
    • if the (i+1)th integer is greater than MAX
      • MAX equals the (i+1)th integer
  • else
    • if the (i+1)th integer is less than MIN
      • MIN equals the (i+1)th integer
    • if the ith integer is greater than MAX
      • MAX equals the ith integer
  • i = i+2
outside → • If there are an odd number of integers in the list
while loop
  • if the last integer is less than MIN:
    • MIN equals the last integer
  • else if the last integer is greater than MAX:
    • MAX equals the last integer
(RETURN MIN and MAX at the end)
```

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.

- Pick an arbitrary node as the starting point for BFS.
- Conduct BFS from the starting node, marking each node with its level in the corresponding BFS tree (e.g. if the starting node ~~root~~ is L_0, then its neighbors are in L_1, whose unvisited neighbors are in L_2, continuing until the deepest level L_n)
- At each vertex visited:
  - If the vertex is marked with an even level, color it with the color A
  - If the vertex is marked with an odd level, color it with the other color B

-4 Time complexity