CS180 midterm

Mingyang Zhang

TOTAL POINTS

100 / 100

QUESTION 1

1 problem 1 20 / 20

✓ **- 0 pts** Correct

QUESTION 2

2 problem 2 20 / 20

✓ **- 0 pts** Correct

QUESTION 3

3 problem 3 20 / 20

+ **3 pts** basic understanding of the question
✓ **+ 5 pts basic understanding of the question is correct**
✓ **+ 10 pts Correct algorithm**
+ **8 pts** Partially correct algorithm
+ **3 pts** Partially correct algorithm
✓ **+ 5 pts runtime analysis and justification**
+ **0 pts** wrong approach
+ **0 pts** no answer
+ **3 pts** Some clues were right but the overal approach was not correct
+ **2 pts** Prof graded

QUESTION 4

4 problem 4 20 / 20

✓ **+ 5 pts Complete proof of correctness**
✓ **+ 5 pts Complete complexity analysis**
✓ **+ 10 pts Correct algorithm**
+ **3 pts** Correct complexity with analysis error
+ **3 pts** Proof of correctness had minor errors
+ **8 pts** Good algorithm, minor errors
+ **5 pts** Incomplete algorithm
+ **0 pts** Algorithm uses non constant storage
+ **0 pts** Complexity analysis is wrong
+ **0 pts** Proof of correctness is wrong
+ **0 pts** Algorithm is wrong
+ **2 pts** Add 2 points
+ **15 pts** Prof graded
+ **20 pts** Prof graded

QUESTION 5

5 problem 5 20 / 20

✓ **- 0 pts** Correct

Name(last, first): Zhang Mingyang

UCLA Computer Science Department

CS 180 **Algorithms & Complexity** ID: 405170429

Midterm **Total Time: 1.5 hours** **November 6, 2019**

Each problem has 20 points .

All algorithm should be described in English, bullet-by-bullet (with justification) You cannot quote any time complexity proofs we have done in class: you need to prove it yourself.

Problem 1: Describe the topological sort algorithm in a DAG. Prove its correctness. Analyze its complexity.

Initially, we create a L with all node in the DAG with no incoming edges. And an empty output_list and we have the DAG G.

While L is not empty.
- Pop node n from L, put it into output_list
- for each node i that n points to
  - remove the edge (n, i), decrease its degree by 1
  - if i has no incoming edge.
    push it into the list L.
  - Endif
- Endfor
- End While

Return output_list / the topological ordering

Prove correctness by contradiction

Suppose in the output_list, we have node i, j s.t. j comes after i but j has a edge pointing to i.

Then, we know that i was pushed into output_list when no incoming edge is connected to it, while (j, i) exist - contradiction

Therefore, our algorithm gives a topological ordering in which only all nodes in the front point to nodes after them

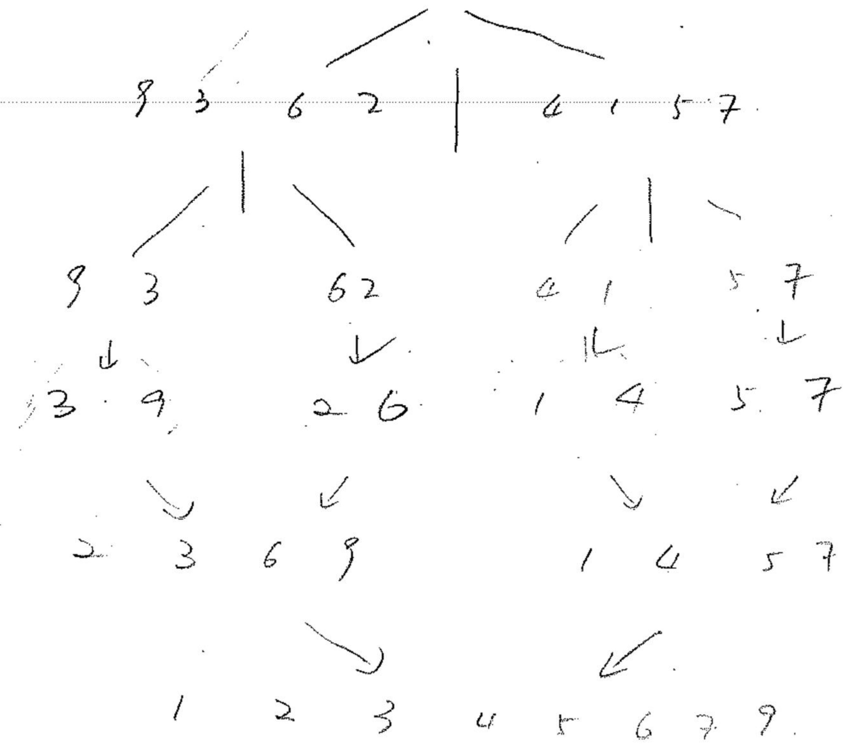
Time complexity: O(|V| + |E|) visit vertices & visit edge once

Since for each node, we have to search for all edges connected to it and we need to traverse all nodes. Therefore, it's O(|V|+|E|)

Name(last, first): Zhang Mingyang

Problem 2: Run Merge sort on the following set of numbers. Show every step. Analyze the time complexity of merge sort on a set of **n** numbers (show every step)

9 3 6 2 4 1 5 7



Sort A, B :

Set cur-pointer to the begining of A, B, create a output list L
 While none of cur_pointer reachs the end
  push the smaller element in L
 Endwhile
 push the rest of the non-empty A or B to L.
 Return L.

Merge A :
 if A has only one element
  - return A;
 -else
  - divide it to two equal parts in the middle A_1, A_2
 - r_1 := Merge (A_1);
 - r_2 := Merge (A_2)
 - r = sort (A_1, A_2) ;
 return r

Time complexity : we divide the task into 2 parts and each time, we sort 2 sorted list in linear time we divide it into 2
then we have $T(n) = 2T(\frac{n}{2}) + n = 2^i T(\frac{n}{2^i}) + in$
then $i = \log n \Rightarrow T(n) = n + n\log n = O(n \log n)$

Name(last, first): Zhang Mingyang

Problem 3: Suppose that you are given an algorithm as a blackbox. You cannot see how it is designed. The blackbox has the following properties: If you input any sequence of real numbers, and an integer **k**, the algorithm will answer YES or NO indicating whether there is a subset of the numbers whose sum is exactly **k**. Show how to use this blackbox to find the ~~subset~~ whose sum is **k**, if it exists.
You should use the blackbox O(**n**) ~~times~~ (where **n** is the size of the input sequence).

~~8~~ 8

1 5 3 4 2 8 7 9

1 2 7 8 9 9

1 2 4 7 10 12 14 2

```
blackbox( L )
  if L's size is 1
    return L[0] == k ? Yes ; No
  for each i ∈ L
    if (blackbox( L - {i} ) == k - i)
      return Yes
  End if
  End for
  Return "NO"
```

Time complexity: O(n)

In each iteration, we check only one and blackbox() is O(1). So in total O(n).

A H H B B B C C A A A
AO A A

Name(last, first): Zheng Mingyang

Problem 4: You have been commissioned to write a program for the next version of electronic voting software for UCLA. The input will be the number of candidates, **d**, and an array votes of size **v** holding the votes in the order they were cast where each vote is an integer from 1 to **d**. The goal is to determine if there is a candidate with a majority of the votes (more than half the votes) . You can use only a constant number of extra storage (note that **v** and **d** are not constants).Prove the correctness of your algorithm and analyze its time complexity.

Initially, we have a vote counter can hold [candidate #, n] and with initial value [d, 0]

for each vi (=di) in array vote
- if (di = d in the counter)
  count [di, n++] set
- Else
  - if counter's n > 0
    - count [d, n--] set
  - Else
    - count [di, 1] set
  - Endif
- End if
- End for

if n > 0
find the # of d in array vote. if greater than v/2 return majority
Return no majority

time complexity: since we traverse the array vote at most twice it's O(v). And we have only one counter it's constant storage

Prove by Induction.

when v=1, vote[0]=d true, we have (d, 1) in the end and d is the majority

Suppose it's correct for v = n.

when v = n+1

case 1: we have a majority d with count n in the end.

subcase 1: if the extra vote is for d, still return d (d is still majority) true

subcase 2: if the extra vote is not for d. if v/2 = # of d (d is not the majority), return no majority, true. if v/2 < # of d d is still majority we still return d true

case 2. we don't have a majority

subcase 1: now we have a majority di then # of d = (v-1)/2 so we have [d, 0] in the end of v-1 case, and now it becam [di, 1] return di true.

Subcase 2: now we still don't have a majority, then # of any d < v/2 still return no majority, true.

Name(last, first): Zheng Mingyang

Problem 5: Consider a sorted list of **n** integers and given integer **L**. We want to find two numbers in the list whose sum is equal to **L**. Design an efficient algorithm for solving this problem (note: an O(n^2) algorithm would be trivial by considering all possible pairs). Justify your answer and analyze its time complexity.

8 8 6

1 4 5 6 9 10 i j

Initially we have two pointer, first points to the beginning of the list. Second points to the end of the list.

while first < second

  if List[first] + List[second] > L

    second--;

  else if List[first] + List[second] < L.

    first++;

  else

    return first, second, we find the two number with sum = L.

  End if

End while

return no such numbers

Time complexity O(n) since we go through each element at most once.

Prove: in this algorithm, we increase our sum by the smallest possible value if our sum < L and verse vice.
So we will find L if it exists a sum in our array