CS180 midterm

Feiqian Zhu

TOTAL POINTS

78 / 100

QUESTION 1

1 problem 1 **13 / 20**

✓ **- 7 pts** No proof of correctness

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

QUESTION 4

4 problem 4 **5 / 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

QUESTION 5

5 problem 5 **20 / 20**

✓ **- 0 pts** Correct

gradescope

Name(last, first): Zhu, Feiqian

U C L A Computer Science Department

CS 180 **Algorithms & Complexity** ID: 905108312

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.

Maintain two sets:

N: ~~number of~~ nodes of graph with no incoming edges

S: number of incoming edges of each node in ~~th~~ the graph.

Go over DAG ~~and find all one nodes without~~ and ~~incoming edge~~ edges, initialize two sets.

while N is not empty

Put ~~that~~ one node, in the output list. (~~without incoming edges~~) in N

delete that node ~~for~~ from N, subtract the number of incoming edges of nodes who has an incoming edge from that node by 1.

If ~~th~~ this causes a node to have no incoming edges

put that node to N.

endif

endwhile.

The initialization costs $O(m+n)$, since every node has been visited together with all edges incident with it. (m is the number of edges, n is the number of nodes.)

One iteration of while loop is $O(1)$.

The while loop is $O(n)$

∴ Total time complexity is $O(m+n)$.

Name(last, first): Zhu, Feiqian

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

1. set A: 9 3 6 2
   set B: 4 1 5 7

2. set A: 9 3
   set B: 6 2

3. set A: 9
   set B: 3

4. merge: set A: 3 9

~~5. merge: set B~~

5. set A: 6
   set B: 2

6. merge: set B: 2 6

7. merge: set A: 2 3 6 9

8. set A: 4 1
   set B: 5 7

9. set A: 4
   set B: 1

10. merge: set A: 1 4

11. set A: 5
    set B: 7

12. merge: set B: 5 7

13. merge: set B: 1 4 5 7

14. merge: 1 2 3 4 5 6 7 9.

$T(n) = 2T(\frac{n}{2}) + cn$

$= 2(2T(\frac{n}{4}) + \frac{cn}{2}) + cn$

$= 4T(\frac{n}{4}) + 2cn$

~~$= 2(2(2T(\frac{n}{8}) +$~~

$\vdots$

$= \log n \, T(1) + n \log n$

$= n \log n$

Name(last, first): Zhu, Feiqian

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).

Put the whole sequence in the blackbox
if ~~YES~~ NO
  return there's no such subset
if Yes
  delete the first number in the current sequence
  ~~while the~~ put ~~it~~ the sequence in blackbox
  ~~do~~ while the return value is YES ~~put the sequence into the blackbox without its first number~~
    ~~then put the sequence~~
    delete its first number
    put the sequence in blackbox
  end while
  Restore the last number deleted from the sequence
  delete the last number in the current sequence
  put ~~it in~~ the sequence in blackbox
  while the return value is YES
    delete the last number
    put the sequence in blackbox
  endwhile
  Restore the last number deleted from the sequence
end if
Return the current sequence.

wrong

see the back for ~~an~~ another answer

~~Since the algorithm deletes numbers.~~
If a number is deleted and never restored, then it means that the sequence without that number still has a subset whose sum is k, which means that number doesn't belong to the target subset. At the end of this algorithm, we find the start and end point of the subset, which denotes the target subset.

Time complexity: Each while loop is O(n). Therefore the total time complexity is O(n).

create a ~~a~~ variable temp
for each element in this ~~array~~ sequence
put it into temp
delete it from the sequence
~~if~~ put the resulting sequence into blackbox
if the return value is YES
continue
else
restore the value in temp
continue
endif

endfor
return the current sequence

time complexity: in worst case, for loop will go over every element in the sequence. During each iteration, it requires one operation. Therefore, this algorithm runs for $O(n)$

correctness: a number is deleted and not restored when the sequence has a subset ~~of~~ with a sum of k. This means this ~~nu~~ number does not belong to that subset. If it is restored, it means without that number, there's no subset summing to k. Therefore, this number belongs to the target subset. ~~Since we go over every number in the sequence, the resulting subset won't have any number not belonging to~~

Name(last, first): Zhu, Feiyuan

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.

we create the following variables :

~~int current majority: an integer~~ majority_1 , majority_2 : two integers.

cmp : an array of size two

~~while the input array is not empty~~

~~if there are at least two elements in the array~~

~~put the first two of them in cmp~~

~~delete them from the input array.~~

time complexity: $O(n^2)$

since every iteration is $O(n)$, it can iterate for n times.

~~Iterate~~

if the input array has only one element

return that element

else if the input array has two elements

return the element if they are the same

else

recursively call the algorithm to the first half of the array

store the return value in majority_1.

call the algorithm to the remaining half of the array

store the return value in majority_2.

~~If neither majority_1~~ ~~and~~ nor majority has a value ~~or they have different~~ values

~~return nothing.~~

~~else if one of them has a return value~~

~~return that value~~

~~else if ther are the same~~

~~return that value.~~

~~end if~~

~~endif~~ else return nothing.

if majority_1 has a value

compare it with every element in the remaining part of the array

return majority_1 if it is the majority

else

if majority_2 has a value

compare it with every element in the other part of the array

return majority_2 if it is the majority.

The algorithm checks for every pair of

Name(last, first): Zhu, Peiyuan

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.

while L is not empty
    Calculate the sum of the first and last element in list
    if it is greater than L
        remove the last element from list
    else if it is smaller than L
        remove the first element from list
    else
        return the first and last element.
    endif
endwhile

Time complexity: each iteration is O(1). since during each iteration we remove one element, the total number of iterations ~~are~~ is at most n. therefor the total ~~runtime~~ time complexity is O(n).