CS180 midterm

Utsav Munendra

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): MUNENDRA, UTSAV

U C L A Computer Science Department

CS 180 **Algorithms & Complexity** ID: 805127226

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.

[Input : G graph]

- Maintain an array I assocating each node with its in-degree
- Initialize every element of I to zero.
- For each edge $e = (u \rightarrow v)$ in G:
  - Add 1 to $I[v]$

- Let S be set of nodes with zero in-degree
- For all nodes n in G with $I[n] = 0$:
  - Add n to S

- While S is not empty:
  - Pop node n from S arbitrarily
  - Print n as next in the topological sort.
  - For all nodes v such that $n \rightarrow v$
    - Decrement $I[v]$
    - If $I[v] = 0$
      - Add v to S
  - Delete node n and its edges
  - Repeat

Proof of correctness

1. Every DAG has some node with zero indegree

- Proof by contradiction
  - Suppose a DAG has no node with zero indegree.
  - Then we can follow the node back to another node always
  - . After (n = no. of nodes) of these backtracks, we will end up with a node we already visited (say v)
  - This means there is a path from v to v.
  - ∴ DAG has a cycle. Contradiction

2. Removing nodes will not produce a cycle in DAG since no new edges are being added.

3. Zero indegree nodes have to come first in a topological sort as nothing needs to preceed them

4. Algorithm will enter while loop due to (1), will never cause DAG to not be DAG by modifying it (2) and will print the zero-indegree nodes first, then print the topological sort of the remaining graph.

∴ Algorithm is correct by design

n - no. of nodes
e - no. of edges

Running Time

- Initialize I has O(e)
- Initialize S is O(n)
- while loops for O(n+e)

∴ Run-time is O(n + e)

Name(last, first): MUNENDRA, UTSAV

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

⇒ [9 3 6 2] ; [4 1 5 7]

⇒ [9 3] ; [6 2] ; [4 1] ; [5 7]

Dividing,
[ ] instance of function recursively running

⇒ [9] ; [3] ; [6] ; [2] ; [4] ; [1] ; [5] ; [7]

Merging

⇒ [3 9] ; [2 6] ; [1 4] ; [5 7]

⇒ [2 3 6 9] ; [1 4 5 7]

⇒ [1 2 3 4 5 6 7 9]

If T(n) is the time taken to merge n numbers,

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

$$= 2\left[2T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right] + cn = 2^2 T\left(\frac{n}{2^2}\right) + 2cn$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right] + 2cn = 2^3 T\left(\frac{n}{2^3}\right) + 3cn$$

in general

$$= 2^i T\left(\frac{n}{2^i}\right) + icn$$

When $i = \log_2 n$

$$T(n) = 2^i\, T\left(\frac{n}{2^i}\right) + icn$$

$$T(n) = n\,T\left(\frac{n}{n}\right) + cn\log n$$

$$T(n) = nT(1) + cn\log n$$

$$T(n) = n + cn\log n$$

$$T(n) = O(n\log n)$$

Name(last, first): MUNENDRA, UTSAV

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).

-1 1 2 6 9   K = 7

yes yes yes yes yes

K = 9

yes No yes

K ≠ 9

eg.

9

-1

100 100 5 5 100 $\boxed{19}$ 100 100

Input

- sequence as an array A; |A| = n
- Set of indices I
- Algorithm B (set of indices, sum)
- sum wanted K

Algorithm

- check if subset exists by B({0,...,n-1}, k)
- if returned No, exit with { }
- Let S be solution indices
- For i = 0 .... m-1
  - Let I = { i, i+1, i+2, ..., m-1 }
  - Let Ans = B(I, K)
  - If Ans = No
    - Add i to solution S
    - K = K - A[i]
  - If K = 0
    - Return solution
- Return solution

Proof

If there are multiple such subsets, the algorithm tries to find the last solution subset

Consider the point in algorithm where we have looked past the initial valid subsets and only the last one remains

Removing a single element from the solution would cause the blackbox to return false.

Our algorithm records this and marks that index to be in the solution and then keeps sampling the remaining list with updated sum

Running time

Loop iterates n times and calls B each time

$\therefore$ Calls to B are $O(n)$

Name(last, first): MUNENDRA, UTSAV

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.

[V is 1-indexed.]

- let the candidates be $1 \ldots d$
- Let $V[i]$ represent the i^{th} vote and $V[i] \in \{1 \cdots d\}$
- let $w = V[1]$ be the running winner
- let $c = 1$ be the running winners margin of win
- For each vote $V[2] \cdots V[v]$ represented by $V[i]$
  - If $V[i] = w$
    - Increment c
  - If $V[i] \neq w$
    - If $c = 0$
      - $w = V[i]$
      - $c = 1$
    - else
      - Decrement c
- Go through $V[i]$ and count the number of votes of w is $> v/2$
- If yes
  - Return w won
- If no
  - Return no majority

Proof of correctness

- Suppose X is the person with majority votes
  - ∴ X has more than $\frac{v}{2}$ votes
  - At each iteration, we cancel votes of two different candidates until we are left with one candidate
  - let $\overline{X}$ be all the opponents of X. Since X has $> \frac{v}{2}$ votes, those votes will cancel $\overline{X}$'s $< \frac{v}{2}$ votes and still X's votes will be left.
  - We would check if X really had $> \frac{v}{2}$ and we are sure no body else had $> \frac{v}{2}$ votes.

  ∴ Algorithm is correct

- Suppose no person has majority
  - w at the end is an arbitrary candidate
  - If check w if he has majority, and return 'no majority' when we fail to find one

  ∴ Algorithm is correct

Running time

loop iterates $O(v)$

Cross-checking takes $O(v)$

∴ Run-time is $O(v)$

Space complexity

We only use w, c apart from input

∴ Space is $O(1)$

Name(last, first): MUNENDRA, VTSAV

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.

eg.

1 2 4 9 10 12 20

29 ?

21 ?

28 ?

Input :

- A, an array of sorted integers
- $|A| = n$
- L, the sum we want

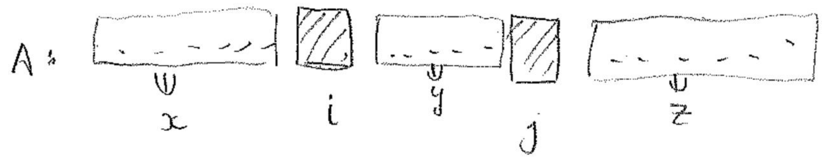
Algorithm

- If $m = 0, 1$ Return that no such numbers exist
- let $s = 0$, $e = m - 1$
- While $s \neq e$
  - let sum = $A[s] + A[e]$
  - If sum = L
    - Return $A[s]$ and $A[e]$ as the numbers
  - If sum < L
    - Increment s
  - else sum > L
    - Decrement s

Repeat

- Return so such numbers exists on the list

Proof of correctness



Assume such a $\underset{= L}{\underline{\text{sum}}}$ exists and it is $A[i] + A[j]$

then since the list is sorted

$A[z] > A[j]$	$A[x] < A[i]$
$A[i] + A[z] > A[j] + A[i]$	$A[x] + A[j] \quad A[j] + A[i]$
$A[i] + A[z] > L$ —(1)	$A[x] + A[j] < L$ —(2)

we keep decrementing the end pointer and we keep incrementing the start pointer

Once any one of those point to the correct pointer, (say $s = i$ but e is in z), then by (1), the sum is always going to be larger than L and we decrement the end pointer

if sum becomes smaller before becoming equal we have to increment s as i would have been further down.

Thus by design, the algorithm will never cross i if we are in Z because it would be decrementing e and the algorithm will never cross j if we are in x because it would be incrementing s

Running time : $O(n)$ as we go over each element once in loop