EE 360C (Section 16685: Nikolova) Final Exam

KHATRI; HAMZA AHMED

TOTAL POINTS

67.5 / 100

QUESTION 1

1 Question 1a 10 / 10

- **- 0 Correct**
- - 1 See comments
- - 2 Incorrect conditions in OPT
- - 2 Partially incorrect OPT (highlighted)
- - 2 An issue with algorithm provided (see comments)
- - 3 Missing conditions in OPT on the weights
- - 3 Missing initializations (how to build the table)
- - 5 No (or incorrect) OPT recurrence or no (or incomplete) pseudocode
- - 10 No answer/ No credit

QUESTION 2

2 Question 1b 10 / 15

- **+ 4 Correct argument for correctness**
- + 2 Almost correct proof of correctennss
- + 1 Incorrect or very weak argument for correctness
- + 0 No reasonable argument for correctness
- + 7 Correct greedy algorithm
- **+ 4 Almost correct greedy algorithm**
- + 0 Incorrect greedy algorithm
- + 4 Correct runtime
- **+ 2 Incorrect runtime but correct analysis.**
- + 1 Almost correct runtime analysis
- + 0 No runtime or incorrect runtime analysis

> you just restated the problem almost. you need to provide an algorithm, not a constraint

QUESTION 3

3 Question 2a 9.5 / 10

- **+ 9 Correct Algorithm**
- + 7.5 Almost Correct Algorithm
- + 6 Incomplete Algorithm
- + 4 Incomplete Alg. /Major errors in Algorithm
- + 3 No solution/ some ideas
- **+ 1 Correct Running Time**
- + 0.5 Almost correct running time
- + 0 No running time
- **- 0.5 Point adjustment**

> Your algorithm is correct. It decides if there exists a 2-coloring or not. You should also describe how you are coloring the graph, i.e., alternate color between the layers of the BFS tree.

QUESTION 4

4 Question 2b 8.5 / 20

- - 0 Says run time for NP is O(V+E) instead of O(E)
- - 0 Says runtime for NP is O(n^2) instead of O(E)
- **- 0.5 Runtime missing/incorrect in NP**
- - 0.5 Runtime missing in construction
- - 12 Used a specific instance of 3-coloring or says any 3-colorable graph is also 4-colorable/ says any 4-colorable graph is 3-colorable. No construction attempted
- - 5 Proof for NP assumes that the problem can be solved in polynomial time
- - 5 Proof for NP is missing
- - 13 Provided a proof sketch for hardness
- - 12 Constructs 4-coloring from 3-coloring by adding another color
- **- 1 Algorithm for checking NP is incorrect**
- - 0 Correct proof for hardness
- - 0 Correct proof for NP
- - 1 Algorithm missing in NP
- - 7.5 Construction incomplete- says add in nodes and remove the 4th color
- - 0 Chooses to reduce to 2-coloring
- - 13 No reduction showing that 4-coloring >= 3-

coloring

- 12 Considers a graph whose maximum degree is 4. No construction provided
- 12 Reduction in the wrong direction. No construction attempted
- 2 Proves that 4-coloring is in P rather than NP
- **10 Attempts to reduce k-coloring to k-SAT. Construction incomplete and does not show equivalence**
- 7.5 Provides a construction of sorts that COULD be used to solve the problem, but many major inconsistencies in the proof
- 7.5 Attempted construction, but incorrect. Only shows that a 3-coloring can also be made a 4-coloring
- 5 Not valid to say 4-coloring is in NP because it is polynomially larger than 3-coloring
- 5 Incorrect understanding of what NP is
- 15 Proof of hardness is missing
- 10 Shows NP completeness of 3-coloring via reduction to 3-SAT and then says any 3-colorable graph is also 4-colorable. Does not show hardness of 4-coloring

QUESTION 5

5 Question 2c 0 / 5

+ 5 Says Unknown
+ **0 Says True**
+ 0 Says False
+ 2.5 Says False, but mentions that if it was true, it implies P=NP
+ 0 Says nothing
+ 0 Incorrect explanation

QUESTION 6

6 Question 3 17 / 20

- 0 Correct
- 10 Greedy: Tried to write greedy algorithm
- 2 Missing a rigorous analysis
- 5 No runtime provided
- **2 Net. Flow: Incorrect runtime (acceptable ones are O(n(n+m)), O((n+m)logn, or O(n^2), O(nlogn) if n is assumed to be >> m)**
- **1 Net. Flow: Did not attempt iff in argument**
- 0.5 Net. Flow: iff without mention of Integer Lemma
- 5 Did not attempt a correctness analysis
- 8 Unclear approach
- 2 Did not label middle capacities correctly
- 4 Did not label entry or exit capacities correctly
- 10 Net. Flow: No network description
- 20 No Credit
- 12 Dyn. Programming: Tried to use DP (substructure is not optimal)

> Your input is jobs and processes, so your running time needs to be WRT those!

QUESTION 7

7 Question 4a 7.5 / 15

+ 15 Correct
+ 7.5 No sufficient explanation or proof given
+ **7.5 Major problems with algorithm**
- 2 Not sufficiently explained why the *overall* median will not be one of the elements that are "removed"/"not considered".
- 1 Not explained how to implement "removing" elements from databases. (Can't directly remove, but can simulate this by appropriate indexing.)
+ 0 No answer

> -This algorithm will not work because you are not preserving the median value (per hint).

QUESTION 8

8 Question 4b 5 / 5

+ **5 Full credit**
+ 0.5 Incorrect recurrence & incorrect application of Master Theorem
+ 2 Incorrect recurrence
+ 0 Incomplete/missing answer
+ 2 Your recurrence is correct, but you don't solve it.
+ 2 You don't solve recurrence
+ 2 Incorrect application of Master Theorem
+ 3.5 Forgot to include the constant work done in each recurrence (ie the comparisons).

EE360C: Algorithms
The University of Texas at Austin
Dr. Evdokia Nikolova and Dr. David Soloveichik

Final Exam
Dec 8, 2016

Name: Hamza Khatri **EID:** haK533

Section (circle): (Nikolova)/Soloveichik

Final Exam

Instructions. No calculators, laptops, or other devices are allowed. This exam is **closed book. No additional materials are allowed.** Write your answers on the test pages. If you need more scratch paper, use the back of the test pages, but indicate clearly where your answers are. Write down your process for solving questions and intermediate answers that may earn you partial credit. **You may invoke theorems used in class without proof, but you cannot invoke in-class exercises, homework problems or quizzes (in other words, you need to write such solutions from scratch).**

If you are unsure of the meaning of a specific test question, write down your assumptions and proceed to answer the question on that basis. **Questions about the meaning of an exam question will not be answered during the test.**

You have **180 minutes** to complete the exam. The maximum possible score is 100.

Master Theorem:
Let $T(n)$ be defined by the recurrence $T(n) = aT(n/b)+f(n)$ (where $a \geq 1$ and $b > 1$ are constants). Then $T(n)$ can be bounded asymptotically as follows:

(a) If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

(b) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$

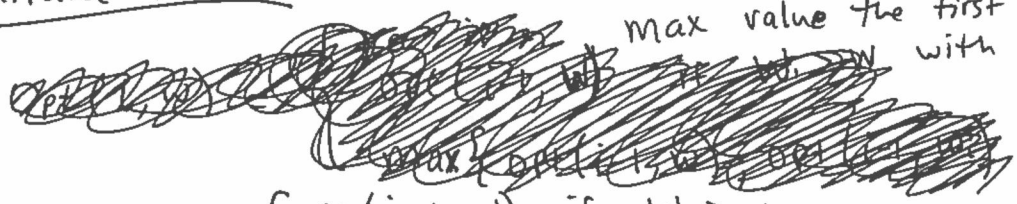
(c) If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Problem 1: The Knapsack Problem [25 points]

Recall the 0-1 Knapsack problem. You are given n items, where item i has weight w_i and value v_i, and a knapsack that can hold at most total weight W, where w_i, v_i and W are integers. The goal is to choose a subset of items that fit in the knapsack and produce the most value.

(a) **[10 points]** Provide a dynamic programming algorithm to solve the 0-1 Knapsack problem in $O(nW)$ running time. For full credit, be sure to give the recurrence definition (OPT), its explanation, and a brief description (or pseudocode) of the iterative (bottom up) algorithm based on the recurrence.

Recurrence Relation: Opt(i,w) is defined as the max value the first i items produce with max weight w.



$$Opt(i,w) = \begin{cases} opt(i-1, w) & \text{if } w_i > w \\ \max\{opt(i-1,w),\ opt(i-1, w-w_i) + v_i\} & \text{if } w_i \le w \\ 0 & \text{if } i = 0 \end{cases}$$

Explanation: There are 2 cases:

1. item fits in knapsack
   - in this case, we add the item to our knapsack if it increases the max value else keep the old max value. Update the weight accordingly.
2. item does not fit
   - we then keep the max value the same as it was without the item (because we don't add to the knapsack). Keep the previous weight b/c nothing was added to knapsack.

Pseudocode:

```
optmatrix [n+1] [W+1] ← empty
initialize first row & column to 0
for (i=1; i ≤ n; i++){
    for (w=1; w ≤ W; w++){
      if (w(i) > w)
        optmatrix[i][w] = optmatrix[i-1][w];
      else
        optmatrix[i][w] = max(optmatrix[i-1][w], optmatrix[i-1][w-w(i)] + v(i));
}
```

Note: This assumes w(i) and v(i) to be hashmaps with constant access.

(b) [15 points] Suppose that the order of elements when sorted by increasing weight is the same as their order when sorted by decreasing value. Describe an efficient *greedy* polynomial-time algorithm for this special case of the problem. Rigorously argue its correctness and running time.

If this is true, then $W_1 \leq W_2 \leq W_3 \ldots \leq W_n$

and $V_1 \geq V_2 \geq V_3 \geq \ldots V_n$

Then the greedy approach will be to add the first K items such that $\sum_{i=1}^{K} w_i \leq W$ is maximized.

This guarantees maximum value since we will be adding the items that provide the most value but at the same time since the weights are the least we will also be adding the most items possible.

This is an $O(n)$ runtime.

Correctness: Let P be an optimal algorithm and G be our greedy algorithm. Then both algorithms will add the same items up to some item K. The $K+1^{th}$ item that G adds will have the least weight and highest value out of the remaining items, so $W_{K+1}(G) \leq W_{K+1}(P)$ and $V_{K+1}(G) \geq V_{K+1}(P)$. We can thus replace W_{K+1} and V_{K+1} of P with that of G, not changing the optimality of P. We can continue doing this for $K+2 \ldots j$ where j is the item that such that $W_{j+1} \geqslant W - \sum_{i=1}^{j} W_i$. Thus our greedy solution is the optimal solution.

RunTime: Since our greedy approach iterates through at max all the items, and assuming accessing V_i is $O(1)$, we know that at most the algorithm will run in $O(n)$ time.

Problem 2: k-Coloring [35 points]
Let us define k-COLORING of an undirected graph as coloring the vertices of the graph with at most k distinct colors such that no two adjacent vertices are of same color. The decision version of this problem is: given an undirected graph, decide (answer yes/no) whether or not a k-COLORING exists for it? It is known that 3-COLORING is NP-complete (you can assume this without proof).

(a) **[10 points]** Give a polynomial time algorithm to find a 2-COLORING of an undirected graph if one exists.

This is the same as finding a bipartite partition of the graph in question, for it is well known and proven that a graph has a 2-coloring iff it has a bipartite partition of V.

Algorithm:

We run BFS on the graph in question. and produce the BFS tree.
We note the levels of each node.
Then we iterate through all the edges and check if any edge is incident on 2 nodes of the same level.
If so, then no bipartite partition exists (and hence the graph is NOT 2-colorable).
If none such edges exist, then the graph is 2-colorable.

This algorithm runs in $O(|E|+|V|)$ time since at most we go through the edges 3 times.

Finding a

(b) [20 points] Prove that 4-COLORING is NP-Complete.

False. It is proven that any graph can be 4-colored (known as the 4-color theorem). Thus every graph has a 4-coloring, so the answer is always yes.

Note: HOWEVER, if we were asked to come up with the 4-coloring, that is NP-complete.

Finding a 4-coloring is NP because given a coloring, we could verify it it's a 4-coloring by iterating through the vertices and counting the # of colors, which is in polynomial time.

4-SAT.

Next we will show that finding a 4-coloring is as hard as

We can represent the colors as boolean literals that are either 0 or 1 and for each node, we create a clause such that the literal for the color of the node is set true iff none of the node's neighbors have the same color.

We OR the literals in the clause and AND the clauses. and all other colors are set true

have

Thus we created a combinational logic circuit that, when evaluated, returns true if a 4-coloring exists and false if it doesn't.

Thus, 4-coloring is polynomial time reducable to 4-SAT which implies 4-coloring is NP-complete.

(c) [5 points] Consider the following statement: "4-COLORING $\leq_p$ 2-COLORING". Is it true, is it false, or is its truth value unknown? Explain.

True. The statement is saying that 4-coloring is at least as hard as 2-coloring, which is true since 2-coloring is polynomial.

Problem 3: Processor Scheduling [20 points]

Suppose you have n unit length jobs to schedule on m processors. The ith job has a list L_i of the processors on which it can be run. Each job i can be assigned to at most one processor on its list L_i. Multiple jobs can be assigned to the same processor, but each processor j has a job limit P_j which is the maximum number of jobs that can be assigned to it.

Describe an efficient algorithm which finds a maximal assignment of jobs—i.e., as many jobs are scheduled as possible. (The running time will be taken into account in the grading.) Rigorously analyze correctness and running time. *Note that it is up to you to determine the type of algorithm to use! Make sure not to get stuck on one approach—there might be a much easier one!*

We will model this as a network-flow problem.
The jobs are nodes on the left and the processors are nodes on the right. For each Processor K in the i^{th} job's L_i, we add an edge from job i to processor K (directed edge). Thus we now have a directed graph with edges going from jobs to processors based on their assignments.
Now, construct a source node and add edges with unit capacity to each job. Add a sink with edges from the processor to the sink with capacity equal to their job limit. Make the rest of the edges unit capacity (i.e. all edges of job assignments are of unit capacity).
Run Ford-Fulkerson to determine the max flow/min cut. Examine the min cut — all edges crossing the cut now give you the maximal assignment of jobs!

Correctness: Since we have unit capacity on the jobs, we can only augment in unit flows — once we augment one path we can no longer use that vertex. Thus, the jobs are assigned to at most 1 processor. Furthermore, the Ford Fulkerson algorithm guarantees we get the min-cut, and the min-cut maximizes the flow in the graph which maximizes the # of jobs assigned.

Run Time: The Edmond-Karps version of Ford-Fulkerson runs in $O(VE^2)$ time which allows us polynomial runtime.

Problem 4: Divide and Conquer [20 points]

You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values—so there are $2n$ values total—and you may assume that no two values are the same. You'd like to determine the median of this set of $2n$ values, which we will define here to be the n^{th} smallest value.

However, the only way you can access these values is through *queries* to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the k^{th} smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

(a) [15 points] Give a divide-and-conquer algorithm that finds the median value using at most $O(\log n)$ queries. Prove its correctness. (You may assume n is a power of 2.)

Hint: Observe that if you remove m elements less than the median and m elements larger than the median, the median remains unchanged. Thus if in each recursion of your algorithm, you determine that m elements are smaller than the overall median, and m elements are larger than the overall median, you can "eliminate them from consideration" while maintaining the same overall median.

Let Q_1 and Q_2 represent the queries to database 1 and 2 respectively.

initialize $K = n$;

Med (n, K):

if $(Q_1(K) < Q_2(n-K))$

return Med $(n, \frac{K}{2})$;

else

return $Q(K)$;

This guarantees that the median is unchanged since we check n and $n-K$ and remove half the items each time.

This runs in $O(\log n)$ at most.

(b) [5 points] Write the recurrence relation ($T(n)$) for the running time of your algorithm. Verify the running time of your algorithm by applying the Master Method to your recurrence.

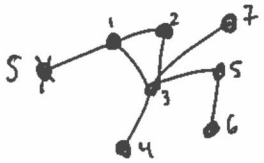
$T(n) = T(\frac{n}{2}) + 1$

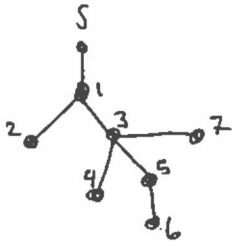
By the master theorem,

$f(n) = 1 = \Theta(n^{\log_2 1}) = \Theta(1)$ ✓

Thus, $T(n) = \Theta(n^{\log_2 1} \log n) = \Theta(\log n)$ ✓

Scratch Page





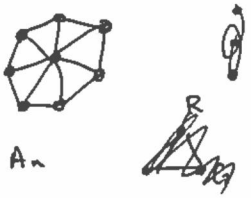
1 ..... n

1 ..... n

Q(K) gives K^{th} smallest #

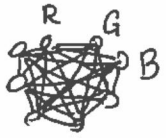
4-SAT

↳ $(X_1 \vee \bar{X}_2 \vee X_3 \vee \bar{X}_4) \wedge \cdots$



$A_1 \cdots A_n$

$B_1 \cdots B_n$



n jobs, unit length

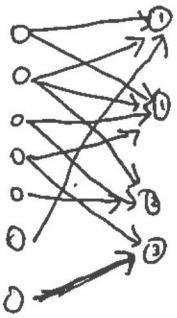
m procs, job limit P_j

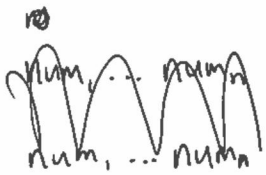
job i has L_i of processors it can go to

multiple jobs to 1 proc
but each job to at most 1 proc

n jobs

m procs





Scratch Page

SORTED

med (n, n)

Check $Q_1(n)$ and $Q_2(n)$

then check

IF($Q_1(n) > Q_2(n)$)

med ($\frac{n}{2}$, n)

else

med (n, $\frac{n}{2}$)