Analysis and Design of Algorithms '21
Midsem

March 3, 2021

1 General Instructions

Exam Ethics

1. Your camera needs to be turned on throughout the duration of the exam
2. The exam starts at 10:00 through GC assignment and ends at 12:00. You will be allowed 10 extra minutes for uploading. Any upload done after 12.10 according to GC will receive zero in the entire exam. So your attempt to upload needs to start at 12:00 sharp.
3. You can either write your solutions cleanly on paper/ type in a word document and upload a *single* pdf on GC.
4. Cellphones and tablets are strictly disallowed. You should leave your devices by your side and confirm this with your proctor.
5. In case you have any doubts, ask your proctor by unmuting yourself. Any important announcement will be made verbally by your proctor.
6. You can leave the exam room only once during the entire duration after securing permission from your proctor/TA. You cannot turn off your camera during this time and you must leave your electronic devices in front of the camera.

Problems

1. There would be five problems, each problem carries 10 points.
2. One problem on solving recurrence - you need to show proper calculations or a mathematically valid argument, and not just write the final answer. Vague statements like 'subproblem size reduces by 5 and so answer is $O(\log n)$' will carry zero credit. The argument carries most of the credit.

3. One problem which can be solved using some form of divide and conquer strategy. For this one, you need to state the algorithm precisely in the form of pseudocode/english description, justify why the algorithm is correct - we prefer formal arguments like contradiction, induction etc. A precise intuitive justification works as well, however, the credit judgement for that lies entirely with the evaluator and you will have less chances to argue if you are using vague statements. Finally, you need to justify the runtime / number of operations (depends on the problem) using recurrences or some other precise argument.

4. Two problems are to be solved by DP. Greedy solution for either of these receives zero. The format for writing a DP solution must be respected for both the DP problems. This means you have to write

   (a) Subproblem definitions

   (b) Recurrence to solve the subproblem from previous part

   (c) Final solution to original problem in terms of subproblems

   (d) Pseudocode for using memoized recursion/iterative procedure and DP table

   (e) Time complexity analysis

   for both. If you do not follow this structure, you will receive a zero like you did in the quiz. In addition, Proof of correctness of recurrence will be required for only one of the two problems. But this will carry significant weightage and you will require to write a proper proof using precise case analysis and contradiction etc. Vague claims will fetch zero credit.

5. One problem about greedy algorithms - you will be given a problem and a greedy algorithm which is wrong. You need to find a testcase/example where this algorithm is *not optimal*. You will need to write the following things-

   a. The description of the instance (you can draw figures)

   b. The output of the given greedy algorithm

   c. The actual optimal solution

   The greedy counter example problem will be graded in a binary fashion i.e. either you have a counter example that works (and gets full credit) or it doesn't work and you get zero.

6. Write precise statements in every problem. Writing a meandering essay will give you no credit.

1. **Find the tighest possible asymptotic behavior of $T(n)$ defined as**

$$T(n) = T(\lceil 3n/5 \rceil) + T(\lceil 4n/5 \rceil) + n^2, T(1) = 1$$

Solution. The correct answer is $\mathcal{O}(n^2 \log n)$. [+2 for just writing the answer and nothing else]

We consider the standard recursion tree for $T(n)$. It is straightforward to observe that the height of the tree in $\mathcal{O}(\log n)$. Now we prove the crucial claim that the total 'work done' at any level of recursion if n^2. We prove this for any level that is full - that is a level ℓ such that that all children of $\ell - 1$ are present. Note that this is an upper bound of the work done at any level.

[+2 for writing height is $\log n$ and +3 for writing that work done at each level is n^2.]

We prove the above by induction on the level ℓ, where $\ell = 0$ for the root and it increases as we descend. The hypothesis is

$$P(\ell) : \text{The total work done outside recursive calls at level } \ell \text{ is } n^2$$

This is clearly true for $\ell = 1$. Assume this true for all levels $0, 1, 2, \cdots \ell$ and consider level $\ell + 1$. Let there be k subproblem nodes at level ℓ. Now, choose a node i at level ell, $i \in \{1, 2 \cdots k\}$ - let the size of the subproblem at this node be n_i and hence the work done n_i^2. Now the children of this subproblem at $\ell + 1$ are of sizes $3n_i/5$ and $4n_i/5$ respectively. Hence, the total work done at these two children outside recursions is $\left(\frac{9}{25} + \frac{16}{25}\right) n_i^2$. Thus, total work done at level $\ell + 1$ is

$$\sum_{i=1}^{k} \left(\frac{9}{25} + \frac{16}{25}\right) n_i^2 = \sum_{i=1}^{k} n_i^2 = n^2$$

,

where the last equality follows from induction hypothesis. Hence we are done.

[+3 for formally proving this by induction. No other proofs like 'true for first two levels hence true for all levels' style of proof fetched points for this part]

2. **Back to Sidhapur.** We are back to Sidhapur, this time to do a population survey of the n residents. What we came to know is there are two types of people here - the Sidha folks - they always speak the truth and the Tedha folks - you do not know whether they are speaking the truth or not. Now, you want to identify all the Sidha folks. Everyone in the town knows everyone else and knows whether they are Sidha or Tedha. But, the folks do not speak much and hence the only question you are allowed to ask somebody, say X is - "Namaste X ! Do you know if Y is Sidha or Tedha" ? But beware, you cannot ask "Hey X, Are you Sidha or Tedha?" That's blatantly rude !!

Fortunately, you have insider information that more than $n/2$ of the folks are Sidha (that's why the town is called Sidhapur and not Tedhapur). Design a strategy to identify all the Sidha folks using only $\mathcal{O}(n \log n)$ questions. You need to give pseudocode/formal description of the strategy, justify the correctness (ideally using formal techniques, but clean intuitive explanation is also fine) and argue the number of questions your strategy asks.

Solution.

We are assuming that n is a power of 2. The following algorithms finds one Sidha folk assuming that Sidha there are more than $n/2$ Sidha folk. This is enough since we can check every other person with this one Sidha folk and conclude the identity of all Sidha folks.

Algorithm 1 Algorithm for finding one Sidha Folk

```
procedure FindSidha($Folks[1:n], n$)
    if $n == 2$ then
        Ask ($Folks[1], Folks[2]$) about each other
        if Both answers ‘Sidha’ then
            Return $Folk[1]$
        else
            Return ‘No Sidha’
        end if
        $s_1$ = FindSidha ($Folks[1:n/2], n/2$)
        $s_2$ = FindSidha ($Folks[n/2+1:n], n/2$)
        Check $s_1$ with everyone except $s_1$ herself
        Check $s_2$ with everyone except $s_2$ himself
        if majority says ‘Sidha’ for either $s_1$ or $s_2$ then
            Return the one who has majority ‘Sidha’ votes
        else
            Return ‘No Sidha’
        end if
```

. Correctness

Claim 1 *For an array of size n, the procedure* FindSidha *finds one Sidha folk correctly if there are strictly more than $n/2$ Sidha folks in the array*

Proof: We prove this by induction (Of course, what else?!!!).

$P(n)$: procedure FindSidha returns one Sidha folk correctly if there are more than $n/2$ Sidha folks in the array

First we prove the base case for $n = 2$. The only case where we need to check the claim is when both the folks are Sidha. Clearly, the procedure returns a Sidha folks correctly by the first ‘If’ condition.

Now consider $P(k)$ true for all $k = 2, 3, \cdots n-1$ and take an array $Folk$ of size n. Consider the two subarrays $Folk[1 : \lfloor n/2 \rfloor]$ and $Folk[\lfloor n/2 \rfloor + 1 : n]$. One crucial observation is that since Sidha is a strict majority in the whole array, they have to be a strict majority in at least one of the two subarrays - let it be the first half (the other half case is symmetric). Now applying induction hypothesis on the first half, we know that the returned answer s_1 is correctly identified as ‘Sidha’. Further, when we check s_1 with everyone except s_1, we know that we are asking at least $n/2$ Sidha folks and at most $n/2 - 1$ Tedha folks. Hence, the majority will answer truthfully that s_1 is Sidha. Hence, we are done. □

Runtime. The recurrence is same as mergesort since there are two recursive calls roughly of size $n/2$ each and the number of questions asked outside the recursive calls in upper bounded by $2n$. Hence the total number of questions is $\mathcal{O}(n \log n)$.

3. **Counter Example.** For the following problem, you need to find an instance/example/test-case where the proposed greedy strategy is not optimal. Your solution needs to have three things and three things only.

   a. The examples (you can draw pictures and label properly or give a written description) **Example.** Three exams with $(p_1 = 5, d_1 = 5), (p_2 = 4, d_2 = 8), (p_3 = 4, d_2 = 8)$ [+4 if example properly written]

   b. The output of the greedy strategy The greedy algorithm will only output exam 1 and discard the rest. [+3 if greedy solution written]

   c. The optimal solution for this example The optimal solution is exams 2 and 3. [+3 if optimal solution written]

   Its exam time and you are working on n take-home assignments. All exams are released on the same day (say day 0) and exam h has deadline $d_h >= 1$. Further, you have an estimate $p_h >= 1$ of how many consecutive days you need to work on exam h. Now your task is to submit as many exams as possible *within their respective deadlines.* Of course you can work on exactly one exam on a particular day and as said earlier, once you start with an exam, you need to submit it before starting on the next one.

   Consider the following strategy. You order the exams in non-decreasing order of deadlines and work on them in this ordering. Show that this strategy is not optimal.

4. **No k in a row.** You are given n balls arranged in a row. Each ball i has a value $v_i >= 0$. Give an algorithm to pick a maximum value subset S of the balls so that *no k consecutive balls* are in S, where $k >= 2$ is an integer.

   For example, say the ball values are

   $$2 \quad 2 \quad 3 \quad 2 \quad 2$$

   and $k = 3$. Then, the maximum value subset has the first, second, fourth and the fifth ball, and their total value is $2 + 2 + 2 + 2 = 8$. Your algorithm should run in time polynomial in n and k.

   For this problem, you also need to write a proof of correctness of the recurrence.

   Subproblems. For all $0 \le i \le n$, let $\mathsf{OPT}(i)$ denote the optimal selection of balls from the set of balls $\{1, 2, \cdots i\}$ such that there are no k consecutive balls.

   Recurrence

   $$\mathsf{OPT}(i) = \max \left\{ \begin{array}{l} \max_{1 \le j \le k-1} \mathsf{OPT}(i - j - 1) + \sum_{\ell=i-j+1}^{i} v_\ell \\ \mathsf{OPT}(i - 1) \end{array} \right\}, \forall k \ge i \le n$$

   $$\mathsf{OPT}(i) = \sum_{\ell=1}^{i} v_\ell, \forall i \le k - 1 \text{ (Base cases)}$$

Final Solution is given by $\mathsf{OPT}[n]$.

Proof of Correctness. We abuse notations and use $\mathsf{OPT}(i)$ to denote both the solution itself and the value of the solution.

The recurrence is easy to check for the base cases when $i \leq k-1$.

Now consider any $k \leq i \leq n$. There could be a few cases for how $\mathsf{OPT}(i)$ looks like.

(a) (ball i is not selected) - This case is exactly the same as we did in lecture for the $k = 2$ case and hence I skip the proof

(b) (ball i is selected) - This case has a number of sub-cases as follows. The ball i can be the last of j consecutive balls , where $1 \leq j \leq k-1$. Each subcase corresponds to a particular value of j in the above range. We prove the correctness of the recurrence for one of these cases. The first ball among the last j included balls is $i-j+1$ - that is the ball $i-j$ is *not included* in this solution. Now consider the set of balls which is $\mathsf{OPT}(i) \setminus \{i-j+1, i-j+2, \cdots i\}$. This solution is clearly feasible for the subproblem with balls $\{1, 2, \cdots i-j-1\}$. We claim that it is actually the optimal solution $\mathsf{OPT}(i-j-1)$. Assume not and say $\mathsf{OPT}'(i-j-1)$ is a different optimal solution for the subproblem, strictly better than $\mathsf{OPT}(i-j-1)$. Now we take this solution and include balls $\{i-j+1, i-j+2, \cdots i\}$. Since $j \leq k-1$, there are at most k consecutive balls in this set of included balls - hence it is feasible for the subproblem with balls $\{1, 2, \cdots i\}$. Further, the value of this solution is $\mathsf{OPT}'(i-j-1) + \sum_{\ell=i-j+1}^{i} v_\ell > \mathsf{OPT}(i)$ - which contradicts the optimality of $\mathsf{OPT}(i)$.

Pseudocode.

Algorithm 2 DP for No k in a row

```
procedure FindOptBalls($val[1:n], n, k$)
    $\mathsf{OPT}[]$ : Array of size $n+1$
    for $0 \leq i \leq k-1$ do
        $\mathsf{OPT}[i] \leftarrow \sum_{\ell=1}^{i} val[i]$
    end for
    for $i = k$ to $n$ do
        $\mathsf{OPT}[i] = -\infty$
        for $j = 1$ to $k-1$ do
            $\mathsf{OPT}[i] = \max\{\mathsf{OPT}[i-j-1] + \sum_{\ell=i-j+1}^{i} v_\ell, \mathsf{OPT}[i]\}$
        end for
        $\mathsf{OPT}[i] = \max\{\mathsf{OPT}[i], \mathsf{OPT}[i-1]\}$
    end for
    Return $\mathsf{OPT}[n]$
end procedure=0
```

Runtime. There are n entries to be filled up. The first k entries require $\mathcal{O}(k^2)$ operations. For any $i > k$, there are k^2 many operations in the worst case - k possible values of j (inner loop) and for each choice of j, we need $\mathcal{O}(k)$ time to compute the sum of j values. Hence, the overall runtime is $\mathcal{O}(nk^2)$.

5. **Jatin turns DJ again.** Fed up with the accidental question paper leak, Jatin decides to turn DJ once again. Now, he has n club nights he can possibly perform at, for simplicity the ith club night is on night i. If he performs on night i, he earns money v_i and this costs him energy w_i. Now, every day he eats food (regardless of whether he has a performance that night) which gives him exactly 1 unit of energy. Jatin starts with an initial energy of 0, so just before the first night, his energy is exactly 1 (from the food that day).

   Jatin needs to make sure that his energy never goes below zero else he won't be able to teach ADA ever again. In other words, if Jatin performs at a subset S of nights then S should satisfy

$$i - \sum_{j \in S : j \leq i} w_j \geq 0 \text{ for all } 1 \leq i \leq n$$

   Give a polynomial-time (in n) algorithm to find the subset of club nights Jatin can perform at to make the most money, while ensuring that his energy always stays non-negative. Assume that w_i, v_i are integers.

   Solution.

   Subproblems. Define $\mathsf{saveJatin}(i, j)$ to be the optimal solution to the subproblem with the nights under consideration being $i, i+1, \cdots n$ and the energy before the performance on night i equal to j, $\forall 1 \leq i \leq n+1, j = 0, 1, 2, \cdots n$. (Note that since Jatin is getting 1 unit of energy every night, the total energy cannot exceed n). [+2 for subproblems]

   Recurrence.

$$\mathsf{saveJatin}(i, j) = \max \left\{ \begin{array}{l} \mathsf{saveJatin}(i+1, j+1) \\ \mathsf{saveJatin}(i+1, j - w_i + 1) + v_i \end{array} \right\}, \forall 1 \leq i \leq n, w_i \leq j \leq n$$

$$\mathsf{saveJatin}(i, j) = \mathsf{saveJatin}(i+1, j+1), \forall 1 \leq i \leq n, j < w_i$$

$$\mathsf{saveJatin}(n+1, j) = 0 \forall\ j$$

   Final Solution. is given by $\mathsf{OPT}[1][1]$

   Runtime. We are filling n^2 table entries and each of them takes constant time. Hence the complexity is $\mathcal{O}(n^2)$.

Algorithm 3 DP for Save Jatin

```
procedure FindOptBalls($v[1:n], w[1:n]$)
    saveJatin$[1:n+1][0:n]$ : Array of size $n+1 \times n+1$
    for $j = 0$ to $n$ do
        saveJatin$[n+1][j] \leftarrow 0$
    end for
    for $i = n$ down to 1 do
        for $j = 1$ to $n$ do
            if $j < w_i$ then
                saveJatin$[i][j] \leftarrow$ saveJatin$[i+1][j+1]$
            else
                saveJatin$[i][j] \leftarrow \max\{$saveJatin$[i+1][j-w_i+1] + v_i,$ saveJatin$[i+1][j+1]\}$
            end if
        end for
    end for
    Return saveJatin$[1][1]$
end procedure
```