Name: 周宇琛

Student ID: 2021533042

School: SIST

Year of Entrance: 2021

ShanghaiTech University Midterm Examination Cover Sheet

Academic Year : 2024 Term: Spring

Course-offering School: SIST

Instructor: Rui Fan

Course Name: Algorithm Design and Analysis / 算法设计与分析

Course Number: CS 240

Exam Instructions for Students:

1. All examination rules must be strictly observed throughout the entire test, and any form of cheating is prohibited.

2. Other than allowable materials, students taking closed-book tests must place their books, notes, tablets and any other electronic devices in places designated by the examiners.

3. Students taking open-book tests may use allowable materials authorized by the examiners. They must complete the exam independently without discussion with each other or exchange of materials.

For Marker's Use:

Section	1	2	3	4	5	6					Total
Marks											
Recheck											

Marker's Signature:

Date:

Rechecker's Signature:

Date:

Instructions for Examiners:

1. The format of the exam papers and answer sheets shall be determined by the school and examiners according to actual needs. All pages should be marked by the page numbers in order (except the cover page). All text should be legible, visually comfortable and easy to bind on the left side. A4 double-sided printing is recommended for the convenience of archiving (There are all-in-one printers in the university).

2.The examiners should make sure that exam questions are correct and appropriate, If errors are found in exam questions during the exam, the examiners should be responsible to respond on site, which will be taking into account in the teaching evaluation.

Instructions for Students

In all problems in which you are asked to design algorithms, you should ***clearly describe*** *how your algorithm works, provide code or pseudocode when asked to, and argue why your algorithm is correct.*

Do not *write your answers on the exam paper. Instead, write them on* separate pieces of paper. ***Write*** *your name and student ID at the top of* ***each piece*** *of paper.*

All answers must be written neatly and legibly in English. If there are ***brief parts*** *of your answer which you cannot express clearly in English, you may write them in Chinese.*

Problem 1

Solve the following recurrences.

(a) $T(n) = 2T(n/8) + \sqrt[3]{n}$ $n^{\frac{1}{3}}$ $a=2, b=8, \log_b a = \log_{2^3} 2 = \frac{1}{3}$

$O(n^{\frac{1}{3}} \log n)$

(b) $T(n) = T(n/3) + T(n/4) + 5n$.

$Cn = C\cdot(\frac{n}{3}) + C\cdot(\frac{n}{4}) + 5n \Rightarrow \frac{5}{12}C = 5 \quad C=12. \quad T(n) = 12n = O(n)$

Answer true or false to the following questions, and explain your answers.

(c) There exist problems in NP which are not NP-complete. X

(d) Every problem in NP can be solved in exponential time. √

(e) Suppose someone found an $O(n^{2024})$ time algorithm to solve the 3-CNF-SAT problem. Then every NP problem can be solved in $O(n^{2024})$ time. X

(5 parts, 5 points / part, 25 points total)

Problem 2

Suppose you are given n integers in the range $[0, k]$, for some constant k. Describe how to process the numbers into a data structure so that any query of the following form can be answered in $O(1)$ time: given two integers x and y, how many input values are in the range $[x, y]$?

prefix sum.

(15 points)

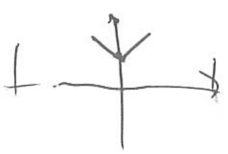
a[y] − a[x−1]

Problem 3

Given a sequence of numbers $a_1, \ldots, a_n$, a *local maximum* is a value $a_i, 1 < i < n$, such that $a_{i-1} \leq a_i \geq a_{i+1}$. In addition, we say a_1 is a local maximum if $a_1 \geq a_2$, and a_n is a local maximum if $a_{n-1} \leq a_n$. Note that there may multiple local maxima for a given input sequence. Give an efficient algorithm based on divide and conquer to find *any* local maximum. Argue that your algorithm is correct and analyze its time complexity.

$O(\log n)$

(15 points)



EXAM CONTINUES ON NEXT PAGE

precomputed

$dp[0]=0$ ~~$dp[1]=a_1$~~ $\quad dp[i] = \max\limits_{1 \le j \le i} \left\{ dp[j-1] + \prod_{k=j}^{i} a_k \right\}$

Problem 4

Given a sequence of numbers $a_1, \ldots, a_n$, the *maximum sum-product* is the largest number which can be formed by multiplying some pairs of consecutive numbers, such that each number can take part in at most one multiplication, and then adding together the products. For example, given the list 1,2,3,1,2,3,4, the maximum sum-product is $22 = 1 + 2 \times 3 + 1 + 2 + 3 \times 4$. Give an algorithm for computing the maximum sum-product, and analyze its time complexity.

$O(n^2)$

(15 points)

$O(nm)$: $\underset{\text{i-dice}}{dp[i]}\underset{\Sigma=j}{[j]} = \sum_{k=1}^{6} dp[i-1][j-k]$

Problem 5

Suppose you have n dice, where each dice has the numbers 1, ..., 6 on its six sides. Give an algorithm to compute the number of ways for the sum of the n dice to add up to a value m. For example, for $n = 2$ and $m = 3$, there are two ways, (1,2) and (2,1). Can you make your algorithm run in $O(n \min(n, m))$ time?

$m \in [n, 6n]$

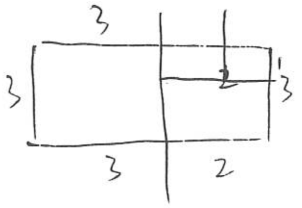
(15 points)

Problem 6

Suppose you are given an $n \times m$ rectangle, and want to cut it into squares using the minimum number of cuts. In each cut you can select a rectangle and cut it into two rectangles in a way that all side lengths remain integers. For example, given a 3×5 rectangle, you can first cut it into a 3×3 square and a 3×2 rectangle. Then you can cut the 3×2 rectangle into a 2×2 square and a 1×2 rectangle. Finally you can cut the 1×2 rectangle into two 1×1 squares. In total this uses 3 cuts. Give an algorithm to solve this problem and analyze its time complexity.

greedy. $O(\min(n, m))$

(15 points)



END OF EXAM

周宇琛 2021533042

Problem 1

(a) $T(n)=2T(\frac{n}{8})+\sqrt[3]{n} \longrightarrow T(n)=aT(\frac{n}{b})+f(n)$

where $a=2, b=8, f(n)=n^{\frac{1}{3}}$

According to the Master Theorem,

since $\log_b a=\log_8 2=\log_{2^3} 2=\frac{1}{3}$, i.e. $f(n)=\Theta(n^{\log_b a})$

so $T(n)=\Theta(n^{\log_b a}\cdot\log n)=\cancel{\Theta(n^{\frac{1}{3}}\log n)}=O(n^{\frac{1}{3}}\log n)$

So above all, $\cancel{T(n)=\Theta(n^{\frac{1}{3}}\log n)}$ $T(n)=O(n^{\frac{1}{3}}\log n)$

(b) $T(n)=T(\frac{n}{3})+T(\frac{n}{4})+5n$.

Guess: $T(n)=\Theta(n)$ for suffciently large n.

So there exist a constant $C>0$, s.t. $T(n)=Cn$.

so $\begin{cases} T(n)=T(\frac{n}{3})+T(\frac{n}{4})+5n \\ T(n)=Cn \end{cases} \Rightarrow Cn=C(\frac{n}{3})+C\cdot(\frac{n}{4})+5n$

$\frac{5}{12}C=5$

$C=12$

so $T(n)=12n=\Theta(n)=O(n)$

So above all, $\cancel{T(n)=\Theta(n)}$ $T(n)=O(n)$

(c) ~~True~~. False.

真子集

1° if $P\neq NP$, then we know that NP-Complete $\subsetneqq NP$

(actually, $NP\cap NP\text{-hard}=NP\text{-complete}$), then NP-complete $\subsetneqq NP$

2° if $P=NP$,

then $P=NP=NPC$. so $\forall$ problem $A\in NP\Rightarrow A\in NPC$.

So when $P=NP$, the statement is false.

(d) True.

Since $NP\subseteq EXP$, so all ~~the~~ NP problem could be solved in exponential time.

So the statement is correct.

(e) ~~True~~ False.

problem A

Since the 3-CNF-SAT is a NP-Complete problem. And if it has an $O(n^{2024})$ algorithm, then it is a P problem. ~~i.e. A~~

polynormial

(back page P2 contiues)

周宇琛 2021533042

P1(e) continues:

Since problem A (3-CNF-SAT) is NP-complete, and $A \in P$

So $\forall$ problem $B \in NP$, there must exist a reduction s.t. $B \le_p A$.

i.e. a mapping $f: B \to A$ s.t. $\forall$ instance X of B, there must exist an instance Y of A, s.t. $f(X) \to Y$

then for any true instance X, $f(X) = Y$ is a yes instance, for any true instance Y, the inverse mapping $X = g(Y) = f^{-1}(Y)$ is also a yes instance.

So we can solve problem B in polynormial time.

Suppose the reduction $B \le_p A$ requires the polynormial time $O(n^p)$

Then solving problem B requires $O(n^p) \cdot O(n^{2024}) = O(n^{2024+p})$

which is also a polynormial time, but not $O(n^{2024})$

Problem 2. Suppose each integer i appears $a[i]$ times.

construct the data structure:

$S[i]$ as the prefix sum of a.

i.e. $S[i] = \sum_{k=-1}^{i} a[k]$

To avoid the impact of negative index,

we define $S[-1] = 0$, where "-1" is the index, instead of last element of S.

So for each query, we want to count the numbers in $[x, y]$

i.e. we want $\sum_{i=x}^{y} a[i] = \sum_{i=-1}^{y} a[i] + \sum_{i=-1}^{x-1} a[i]$

$$= S[y] - S[x-1].$$

So with the data structure: prefix sum, we can construct "a" is $O(n)$ time, and "s" in $O(k)$ time.

the for each query, we just require $O(1)$ time.

Total ~~with~~ space complexity is $O(n+k)$.

周宇琛 2021533042

Problem 3: We can use devide and conquer to find ~~the~~ a local maximum of range $[l, r]$

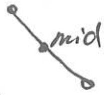
1° get the mid point of the interval

$$mid \leftarrow \lfloor \frac{l+r}{2} \rfloor$$

2° then for the mid point, it has 4 situations.

<1> $a_{mid-1} \le a_{mid} \ge a_{mid+1}$

then we get a local maximum, and return "mid".



<2> $a_{mid-1} \ge a_{mid} \ge a_{mid+1}$

then the local maximum must be in the range $[l, mid-1]$, so we resursively solve the subproblem, the local maximum in $[l, mid-1]$ must be a local maximum in $[l, r]$

<3> $a_{mid-1} \le a_{mid} \le a_{mid+1}$

Similarly to <2>, we just need to recursively find the local maximum in $[mid+1, r]$, and the reasons are the same as <2>

<4> $a_{mid-1} \ge a_{mid} \le a_{mid+1}$

i.e. mid is a local minimum, so we can either find the local maximum in $[l, mid-1]$ or $[mid+1, r]$, choose random one range will find the local maximum

3° from the analysis above, we could find the local maximum in $O(\log n)$ time.

4° For the edge situation, we need to specificly judge if $a_1 \ge a_2$ or $a_n \ge a_{n-1}$

Prove correctness:

Since edge situation is concerned, then for a range, there must exist a local maximum. Situation <2> and <3>, the half-side range must exist a local maximum, ~~for~~ even for the extramly cases. i.e. the sequence a is monotone. And for situation <4>, no matter how extreme it is, the local minima's left and right side both exist at least a local maximum, so we can just take any half-side to recursively find. So it is a correct algorithm.

So above all, the algorithm is correct, and its time complexity is $O(\log n)$.

周守谋 2021533042

Problem 4:

We can define the dynamic programming's state:

dp[i] to be the maximum ~~sub~~ sum-product,

of the sequence $a_1, \cdots, a_i$.

Then for the state end with a_i, we can convert it from the subproblem end with a_{j-1} ($1 \le j \le i-1$)

and we multiply the $a_j, a_{j+1}, \cdots a_i$

And for dp[j], we can recursively get its value.

So we can write up the process:

initially, set $dp[0]=0$ ⟹ start multiplying from j to i

for $i=1,2,\cdots,n$

$$dp[i] = \max_{1\le j\le i} \left\{ dp[j-1] + \prod_{k=j}^{i} a_k \right\}$$

which represent the multiply $a_j, \cdots, a_i$, and consider the subproblem end up with $j-1$

And $\prod_{k=j}^{i} a_k$ could be precomputed in $O(n)$ time, and in dp equation, it can be computed in $O(1)$ time.

Let $S[0]\leftarrow 1$, $S[i]=S[i-1]*a[i]$

so $\prod_{k=j}^{i} a_k = \frac{S[i]}{S[j-1]}$

So above all, the algorithm can be ~~comp~~ done with dp.

the time complexity $= O(n^2)$

as dp[i], the outer loop requires $O(n)$, and the "max" requires a inner loop, which also requires $O(n)$.

周宇琛 2021533042

Problem 5:

Since we are getting the sum of dices, so we can get that

<1> if $m < n$ or $m > 6n$, then it must have no solution

i.e. the number of ways is 0

<2> So we only need to consider the cases that.

$n \le m \le 6n$.

So $O(n) = O(m) = O(\min(n,m))$

And we set up a dp algorithm.

Let dp[i][j] represents for the first i dices, the sum number is j's number of ways.

So the initial condition is that $dp[0][0]=0$. and $dp[1][k]=1$, $k=1,\cdots,6$

And for dp[i][j], we can consider the i-th dice:

it can have value of $1,2,\cdots,6$.

So we can transform it from the total $(i-1)$ dices with value $j-1,\cdots,j-6$

So the dp algorithm is:

$dp[0][0]=1$, $dp[1][k]=1$, $(k=1,2,3,\cdots,6)$

for $i=2,\cdots,n$:

~~dp[i][j] = dp[i-1][~~

$$dp[i][j] = \sum_{k=1}^{6} dp[i-1][j-k]$$

And the final solution is dp[n][m].

So the total time complexity is $O(nm)$.

Since $n \le m \le 6m$, so $O(n)=O(m)=O(\min(n,m))$

So we have the time complexity $O(n \cdot \min(n,m))$

周宇琛 2021533042

Problem 6:

We can use greedy algorithm to get the minimun number of cuts.

For a $n \times m$ rectangle, we cut it to get a $\min(n,m) * \min(n,m)$ square.

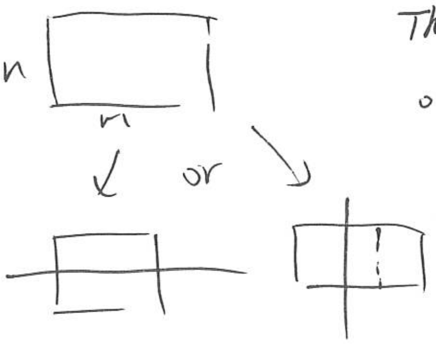
So W.L.O.G, we assumethat $n \le m$
(if $n > m$) we just rotate the rectangle $90°$. and swap(n,m))

The we can cut the rectangle to get a ~~$n \times m$~~ $n \times n$ square and a $(m-n) \times n$ rectangle.

Then we can recursively cut the $(m-n) \times n$ rectangle with this method. until $n = m$.

This greedy algorithm will get a optimal solution, i.e. minimum number of cuts.

Correctness: Support there is a cut with less number



Then it will get 2: ~~two~~ $a \times m$, $(n-a) \times m$ rectangles
or 2: $n \times a$, $n \times (m-a)$ rectangles. where $1 \le a < n$.
it no longer get less rectangles
It generate more rectangles and less squares.
And these more rectangles also do not save any cuts.
So our algorithm is optimal.

So the time complexity is $O(\min(n,m))$