Yinfeng Cong

CS 344

mtd 1

Problem 1 : (a).

solution: $f(n) = o(g(n))$ $\quad 2^{\log\log n} = \log n \Longrightarrow g(n) = \frac{n}{\log n}$

proof: we set $n > 2^{2^n}$

$\sqrt{\log n} > \sqrt{2^n} = (2^n)^{\frac{1}{2}}$

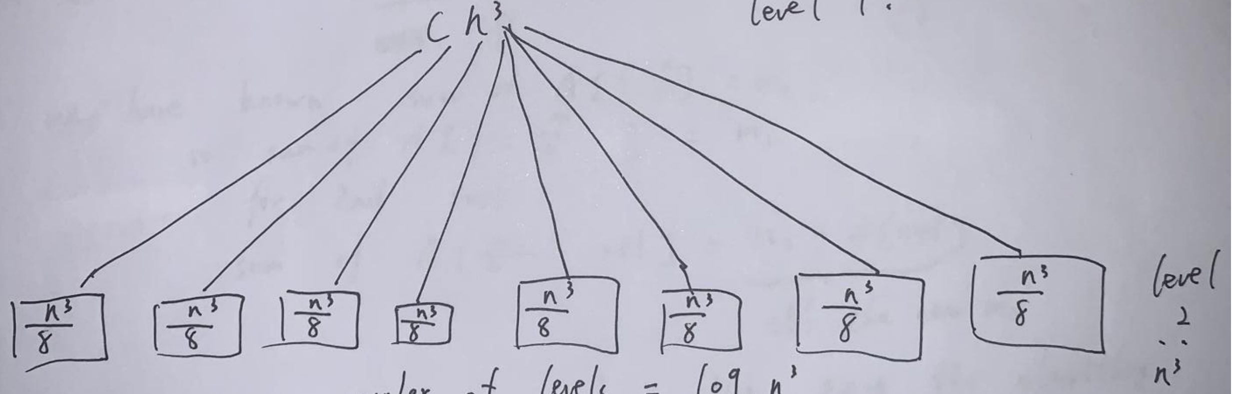
$\frac{n}{2^{(\log\log n)}} > \frac{n}{2^n} = \frac{2^{2^n}}{2^n} = 2^{2n-n} = 2^n$

2^n much bigger than $(2^n)^{\frac{1}{2}}$

we have $\lim_{n \to 0} \frac{(2^n)^{\frac{1}{2}}}{2^n} = 2^{-\frac{1}{2}n}$ $(n = +\infty)$

hence, the solution is proved

(b). $T(n) \leq 8T(\frac{n}{2}) + O(n^3)$



number of levels $= \log_8 n^3$

level i: Cn^3

Sum $=$ level i $\cdot$ number of levels $= Cn^3 \cdot \log_8 n^3$

$T(n) \leq O(n^3 \log n^3)$

Problem 2 : (a),

Induction hypothesis: for any number $n \geq 1$, and array A of size n, Total-Sum ($A[1:n]$) returns the sum of numbers from $A[1]$ to $A[n]$

Induction base: for $n=1$, sum $= A[1]$, thus it returns the correct answer for base case

Induction step: we assume that it is correct for $n \leq a$, and we prove it for $n = a+1$

we know the correctness of $A[1:\frac{n}{2}] = A[1:\frac{a}{2}]$ returns m_1

and $A[\frac{n}{2}+1 : n] = A[\frac{a}{2}+1 : a]$, it returns m_2

so it's correct for $A[1, n] = A[1, a]$

we now need to prove it for $A[1, n+1] = A[1, a+1]$

we divide it into $\underbrace{A[1 : \frac{n}{2}]}_{P}$ and $\underbrace{A[\frac{n}{2}+1 : n+1]}_{Q}$

we have known sum of $A[1:\frac{n}{2}] = m_1$,

so sum of $A[1:\frac{m}{2}] = m_1$

for 2nd part

sum of $A[\frac{m}{2}+1 : n+1] = \underbrace{m_2 + A[n+1]}_{\text{it's the new } m_2'}$

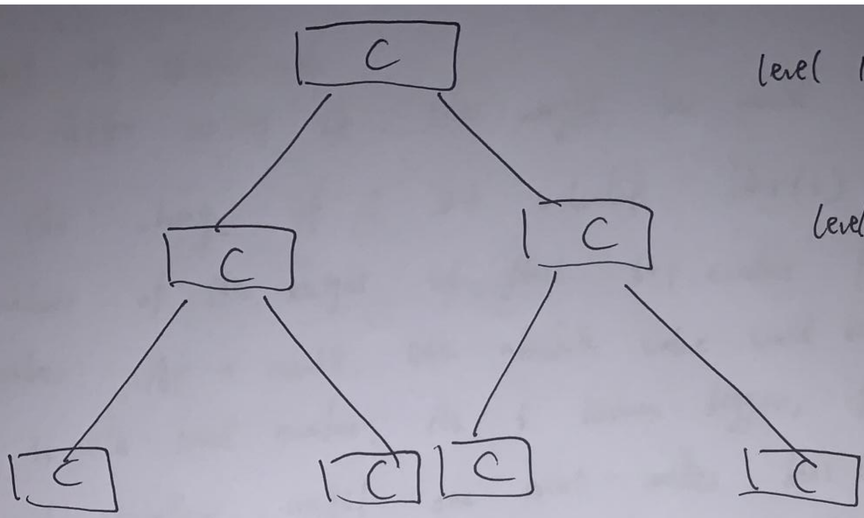
so, we know sum $= m_1 + m_2'$, which prove the algorithm.

(b) we get $T(n) \leq 2T(\frac{n}{2}) + O(1)$

we divide into 2 parts, and spends $O(1)$ additional time to output the solution

(recursion tree in next page)

[This is the same kind of problem as Hw1 problem 3]



level 1: C

level 2: 2C

level 3: 4C

level i: $2^{i-1} \cdot C$

number of levels: $\log_2 n + 1$

$$\text{sum} = \sum_{0}^{\log_2 n + 1} 2^{i-1} \cdot C = C \cdot \frac{2^{\log_2 n + 1} - 1}{2 - 1} \approx 2^{\log n} = n$$

$$T(n) \leq O(n)$$

Problem 3: (a).

Algorithm: Use merge sort to sort array A, and array w will change with A as well (because it's the weight of the array in A)

For $i = 1$ to n,

$W_i = \sum W[1] + \cdots W[i-1]$

$W_{n-i} = \sum W[i] + \cdots W[n]$

bias[i] $= | W_i - W_{n-i} |$

If bias[i+1] > bias[i] for the first time,

output bias[i],

break.

(because from first, bias[i] should > bias[i+1], and there's a point i that makes bias[i] < bias[i+1], and this bias[i] would be the answer)

Problem 3: (b).

Proof of correctness:

After sorting A, the weight w would change as well following the change of A. we calculate bias[i] as the absolute value of the weight of first i-1 numbers $\boxed{\text{minus}}$ the last n-i numbers. As a result, the absolute value would be extremely big for i is a small number. As i becomes bigger, the bias would be smaller and smaller until one point makes bias[i+1] > bias[i]. which means from here bias[i+1] > bias[i] is true. So bias[i] here would be the smallest bias. which prove the correctness.

Problem 3: (c).

Algorithm: Running merge sort we used $O(n\log n)$ time, and the we used $O(n)$ for the for loop.

So total runtime $= O(n) + O(n\log n) = O(n\log n)$

Problem 4 : (a).

Algorithm: build an array $D[1:n^2]$

for $i = 1$ to n, $j = 1$ to n.

Put all numbers $A[i] \cdot B[j]$ into array D

Then, use counting sort:

① Design a new array E and initalize it to all 0.

② For $m = 1$ to n^2, increase $E[D[i]]$ by one

Then, for $k = 1$ to n,

if $E[C[k]] \geq 1$,

out put "yes",

if $E[C[k]] < 1$,

$k{+}{+}$.

If the for loop ends, and we didn't out put yes, then output "no".

proof of correctness:

The array E contains all the location of elements in D, and $C[k]$ is the numbers that contained in C. So we search if the location of $C[k]$ in E is equal or bigger than 1, and we can see if in the array D has a element that is equal to $C[k]$.

Runtime analysis: We used $O(n^2)$ time to put all $A[i] \cdot B[j]$ into array D, and counting sort used $O(\ )$ time. Then, we use $O(n)$ time to check if $C[k]$ has a position in array E.

Total runtime $= O(n^2) + O(n^2) + O(n) = O(n^2)$

which it's correct.

Problem 4 : (b).

Algorithm: (a). Build an array $D[1:n^2]$

for $i=1$ to n, $j=1$ to n.

put all numbers $A[i]\cdot B[j]$ into array D.

(b). Create a hash table T of size $n^2=m$. using a near-universal random hash family, and by handling collisions using the chaining method

(c). For $k=1$ to n, search if $C[k]$ belogs to the hash table T.

If $C[k]$ belongs to T,

output yes.

If $C[k]$ doesn't belong to T,

k++,

After ending for loop, if we didn't find $C[k]$ belongs to T, output no.

Proof of correctness: We have proved the correctness of hash table in the past lecture, so we know it's correct. T contains all the variables in D (which is the multiply of A and B), so we can go over $C[k]$ to see if it can be found in T. If it can be found which means for some $D[x]=C[k]$. If it's not found there's no x makes $D[x]=C[k]$.

Runtime analysis: We used $O(n^2)$ time to put all $A[i]\cdot B[j]$ into array D, and we used $O(m+n^2)=O(n^2)$ to create the hash. At last, we used $O(n)$ to search $C[k]$ in T.

Runtime $= O(n^2)+O(n^2)+O(n) = O(n^2)$

which it's also correct.

Problem 5: (a).

For any integers $n \geq 1$, define:

$K(n)$: the minimum number of coins required to have a total prize n, if it's not possible to purchase the item using any combinations of coins, we define $K(n) = +\infty$

The solution to the problem can be obtained by retaining $K(n)$.

(b) Recursive formula:

$$K(n) = \begin{cases} +\infty & \text{for } n < 0 \\ 0 & \text{for } n = 0 \\ 1 + \min\{K(n - C[1]), K(n - C[2]) \cdots\cdots K(n - C[n])\} & \text{otherwise} \end{cases}$$

+∞

Proof of correctness:

① there's no way to purchase an item that has negative prize, so $K(n) = +\infty$ for $n < 0$.

②. If the prize $= 0$ which means that we can not pick any coin to buy it, so $K(0) = 0$.

③. Because we know that we use the coins from n coins, and each of them can only be used once. If we pick coin $C[1]$, then we end up using one coin and have to purchase the remaining amount which is $n - C[1]$, In this case, the number of coins will be $1 + K(n - C[1])$, similarly for picking any other coin $C[i]$. Our goal is to use minimum number of coins, taking minimum of n coins, give us the correct answer.

Problem 5: (c).

we use memoization here, we have n kind of coins.

we store an array $D[1:n]$ intialized with "undefined"

$memCoin(M)$: 1. if $M < 0$, return $+\infty$

2. if $M = 0$, return 0

3. if $D[i] \neq$ 'undefined'; return $D[i]$

4. Otherwise, let $D[i] = 1 + \min\{memcoin(M - C[1]) \cdots memcoin(M - C[n])\}$

5. return $D[i]$

(d). runtime: Our algorithm runs in $O(M \cdot n)$ time, as there are M subproblems and each subproblem takes $O(n)$ time to get the minimized numbers of coins. So total runtime $= O(M \cdot n)$

Problem 6: extra

Algorithm: We use binary search in this problem to find the point j.

We divide both A and B for i times.

first we pick $A_1[1:\frac{n}{2}]$ $A_2[\frac{n}{2}+1:n]$

$B_1[1:\frac{n}{2}]$ $B_2[\frac{n}{2}+1:n]$

If $A[\frac{n}{2}] < B[\frac{n}{2}]$,

we divide A_2 and B_2 again, and search mid point of it.

If $A[\frac{n}{2}] > B[\frac{n}{2}]$,

we divide A_1 and B_1 and search their mid point.

Until we find the proper point j. that

$A[j] < B[j]$

$A[j+1] > B[j+1]$

Problem 6: extra

Proof of correctness:

we divide the arrays for several times and compare their mid point to see which part of the arrays should we divide again. If $A[i] < B[i]$ means that we need to find the transition of $B[j] > A[j]$, so we search 2nd part. If $A[i] > B[i]$ means that we need to find the transition that $A[j] < B[j]$, so we search 1st. We divide the arrays for many times until we get the transition point. $A[j] < B[j]$, $A[j+1] > B[j+1]$

Runtime: There are 2 Binary search in the algorithm both for A and B. So run time $= O(\log n) + O(\log n) = O(\log n)$.