主題

日期

星期

Homework 2 2.6, 2.8, 3.5, 3.6

谢泽钰 2020012544

2.6 (a) $f(n) = n^3$

Proof. For any $i, j \le n$ satisfying $i < j$

We need $|j-i| = j-i$ steps to add up entries A[i] through A[j]

So the total times of sum operation is

$$\sum_{1\le i<j\le n}(j-i) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(j-i) = \sum_{i=1}^{n-1}\frac{(n-i)(n-i+1)}{2}$$

$$\le \sum_{i=1}^{n-1}\frac{n^2}{2} = \frac{n^2(n-1)}{2} \le \frac{1}{2}n^3$$

In other words, we have $g(n) = O(f(n)) = O(n^3)$

($g(n)$ stands for the number of the adding operation)

(b) $$\sum_{1\le i<j\le n}(j-i) = \sum_{i=1}^{n-1}\frac{(n-i)(n-i+1)}{2} \ge \sum_{i=1}^{n-1}\frac{(n-i)^2}{2}$$

$$= \frac{1}{2}\sum_{i=1}^{n-1}(n-i)^2 = \frac{1}{2}\sum_{i=1}^{n-1} i^2 = \frac{1}{12}n(n-1)(2n-1) \le \frac{1}{6}n^3$$

In other words, $g(n) = \Omega(f(n)) = \Omega(n^3)$

This shows an asymptoticaly tight bound of $\Theta(f(n))$ on the running time.

(c) We can optimize the algorithm to $O(n^2)$ by preloading the sum of some subarraies

The algorithm is as belows:

主題

日期

星期

```
Preload:
  For i = 1, 2, ..., n
     if i = 1     S[i] = A[i]
     else   S[i] = S[i-1] + A[i]
   End for
 Main Part:
    For i = 1, 2, ..., n
      For j = i+1, i+2, ..., n
         if i = 1      B[i,j] = S[j]
         else      B[i,j] = S[j] - S[i-1]
      Endfor
    Endfor
```

The reload part has a running time of $O(n)$ while the main part has a running time of $O(n^2)$
So we lead to the conclusion that this algorithm has a running time of $O(n^2)$

2.8 (a) The strategy is as follows:
First we need to find a positive integer t such that $(t-1)^2 < n \leq t^2$ ($n \geq 1$ so the satisfied t always occurs)
Then we try throwing the first jar at the position of
$1, 1+t, 1+2t, \cdots, 1+(t-1)t$
And record when the jar breaks

主題

日期

星期

Assume the jar breaks at the position of $(1+at)$ $(0\le a<t)$
If $a=0$ then we know the highest safe rung is 0
Otherwise $a\ge 1$, we now throw the second jar on the rung of $1+(a-1)t+1$, $1+(a-1)t+2$, $\cdots$, at
Assume the jar breaks at the position of $1+(a-1)t+b$ $(1\le b<t)$ Then we know the highest safe rung is $1+(a-1)t+b-1=(a-1)t+b$
Finally let's count the number of our experiments.

In the first loop we take the experiments for at most $(t-1)\le\sqrt{n}$ times

In the second loop we take the experiments for at most $(t-1)\le\sqrt{n}$ times

So the running time of this algorithm is no more than $\sqrt{n}+\sqrt{n}=2\sqrt{n}$, satisfying the inequation $\lim\limits_{n\to\infty}\frac{f(n)}{n}=0$

as for $\lim\limits_{n\to\infty}\frac{f(n)}{n}\le\lim\limits_{n\to\infty}\frac{2\sqrt{n}}{n}=0$

(b) For any $k>2$, we can find the only positive integer t such that $(t-1)^k\le n<t^k$

And then we have at most k loops as follows:

In the first loop we respectively try these rungs:

1, $1+t^{k-1}$, $1+2t^{k-1}$, $\cdots$, $1+(t-1)t^{k-1}$

And get the smallest a_1 where $1+a_1t^{k-1}$ breaks

主題

日期

星期

the jar

If $a_1 = 1$ then the highest safe rung results in 0

Otherwise we have to start the second loop:

$1+(a_1-1)t^{k-1}$, $1+(a_1-1)t^{k-1}+t^{k-2}$, $\cdots$, $1+(a_1-1)t^{k-1}+(t-1)t^{k-2}$

And similarly we can get a_2 as the smallest rung from which the jar breaks

...

Just after at most k terms of such loops can we reach the ending of the algorithm, which means we get the highest safe rung.

During this process, the number of our manipulations

$f(n) = a_1 + a_2 + \cdots + a_k \leq tk \leq k \cdot n^{\frac{1}{k}}$

This is just rough stimulation since we assume $k \ll n$. Otherwise if $k > n$, we can just try the rungs one by one, resulting a constant number c of the experiments.

But that doesn't matter with our focusing problem.

It's simple of the conclusion $\lim_{n\to\infty} \frac{f_k(n)}{f_{k-1}(n)} = 0$

when considering the magnitude $\frac{kn^{\frac{1}{k}}}{(k-1)n^{\frac{1}{k-1}}} = O(n^{-\frac{1}{k(k-1)}})$

And $\lim_{n\to\infty} n^{-\frac{1}{k(k-1)}} = 0$

主題

日期

星期

3.5. Let n = the number of nodes

Let x_n and y_n respectively stands for the number of nodes with two children and with no children.

Our goal is to prove that $x_n = y_n - 1$ for any positive integer n

We prove this conclusion with induction of n

First, when n=1, there's only one variety of binary trees, which is also called a single node.

In this case we have $x_1 = 0$, $y_1 = 1$, satisfied

Now suppose for all sorts of binary tree with n nodes, we have $x_n = y_n - 1$

Consider add the No.(n+1) node onto this tree, obviously, we know it cannot be the root. So there are two circumstances, listed below:

I. The new node's parent is a leaf originally

In this case x_{n+1} and y_{n+1} remains to be the same as x_n and y_n

II. The new node's parent is not a leaf node

In this case it must have only one child before.

And it lead to $x_{n+1} = x_n + 1$, $y_{n+1} = y_n + 1$

Whichever cases it is, $x_n - y_n$ remains unchanged.

As a result, according to the induction method, the claim is proved.

主題

日期

星期

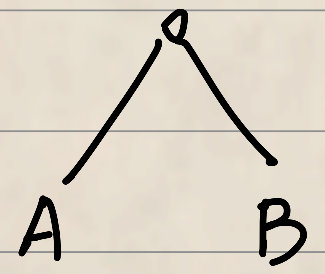
3.6. We can still use induction methods

Let d be the number of the binary tree's levels.

When $d=1$, the tree is a single node.

Suppose the claim is right for any d in $\{1,2,\cdots,t\}$.

Then for $d=t+1$, the binary tree has two varieties:



I

II

In these two graphs, A.B are independant binary trees with less than $t+1$ levels.

In case I. Both BFS and DFS add two edges as shown.

In case II. Both BFS and DFS add one edge as shown.

As a result, we lead to the conclusion that BFS and DFS create the same tree