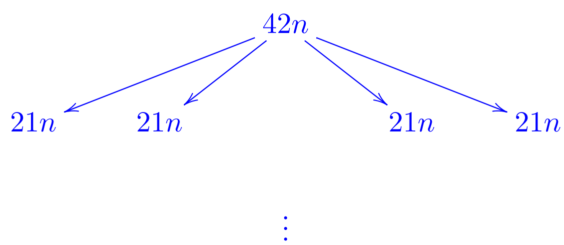
CS 170 Dis 1

Released on 2017-01-27

1 (★★) Recurrence Relations

(a) $T(n) = 4T(n/2) + 42n$

Solution: Use the master theorem. Or:



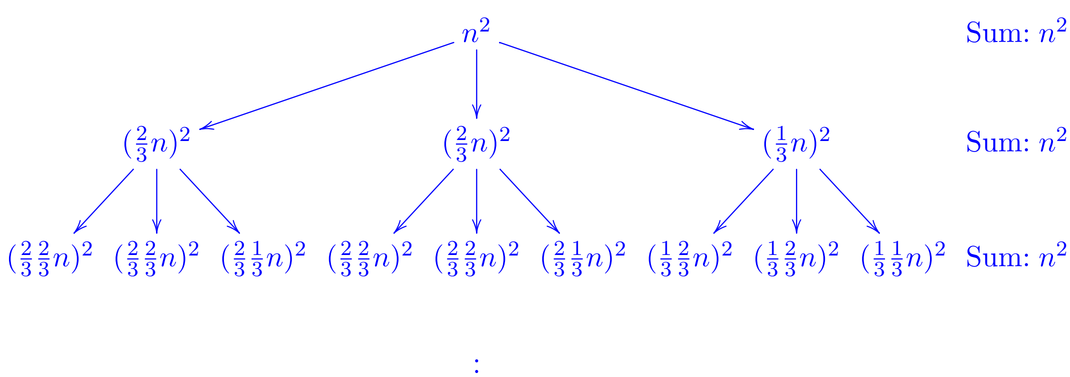
The first level sums to $42n$, the second sums to $84n$, etc. The last row dominates, and we have $\log n$ rows, so we have $42 \cdot 2^{\log n} \cdot n = \Theta(n^2)$.

(b) $T(n) = 4T(n/3) + n^2$

Solution: Use the master theorem (the case $d > \log_b a = \log_3 4$), or expand like the previous question. The answer is $\Theta(n^2)$.

(c) $T(n) = 2T(2n/3) + T(n/3) + n^2$

Solution: This one is a bit tricky:



So the answer is: $\Theta(n^2 \log n)$.

(d) $T(n) = 3T(n/4) + n \log n$

Solution: We end up with $\sum_{i=0}^{\log_4 n} (3/4)^i n \log(n/4^i)$. We can lower-bound this by $n \log n$ by taking the first term, and upper-bound it by $n \log n$ be replacing $\log(n/4^i)$ by $\log n$, so this is $\Theta(n \log n)$.

2 (★★) Sorted Array

Given a sorted array A of n (possibly negative) distinct integers, you want to find out whether there is an index i for which $A[i] = i$. Give a divide-and-conquer algorithm that runs in time $O(\log n)$. Provide only the main idea and the runtime analysis.

Solution: Along the same lines as binary search, start by examining $A[\frac{n}{2}]$. Clearly, if $A[\frac{n}{2}]$ is $\frac{n}{2}$ then we have a satisfactory index; if $A[\frac{n}{2}] > \frac{n}{2}$ then no element in the second half of the array can possibly satisfy the condition because each integer is at least one greater than the previous integer, and hence the difference of $A[\frac{n}{2}] - \frac{n}{2}$ can not decrease by continuing through the array; and if $A[\frac{n}{2}] < \frac{n}{2}$ then by the same logic no element in the first half of the array can satisfy the condition. While the algorithm has not terminated or left an empty array, we discard the half of the array that cannot hold an answer and repeat the same check. At each step we do a single comparison and discard at least half of the remaining array (or terminate), so this algorithm takes $O(\log n)$ time.

3 (★★) Counting inversions

This problem arises in the analysis of *rankings*. Consider comparing two rankings. One way is to label the elements (books, movies, etc.) from 1 to n according to one of the rankings, then order these labels according to the other ranking, and see how many pairs are "out of order".

We are given a sequence of n distinct numbers $a_1, \cdots, a_n$. We say that two indices $i < j$ form an inversion if $a_i > a_j$ that is if the two elements a_i and a_j are "out of order". Provide a divide and conquer algorithm to determine the number of inversions in the sequence $a_1, \cdots, a_n$ in time $O(n \log n)$ (*Hint:* Modify merge sort to count during merging)

Solution:

(i) **Main idea**

There can be a quadratic number of inversions. So, our algorithm must determine the total count without looking at each inversion individually.

The idea is to modify merge sort. We split the sequence into two halves $a_1, \cdots, a_m$ and $a_{m+1}, \cdots, a_n$ and count number of inversions in each half while sorting the two halves separately. Then we count the number of inversions (a_i, a_j), where two numbers belong to different halves, while combining the two halves. The total count is the sum of these three counts.

(ii) **Psuedocode**

```
procedure COUNT(A)
    if length[A] = 1 then
        return A, 0
    B, x ← COUNT(first half of A)
    C, y ← COUNT(rest of A)
    D ← empty list
    z ← 0
    while B is not empty and C is not empty do
        if B is empty then
```

Append C to D and remove elements from C
else if C is empty **then**
Append B to D and remove elements from B
else if $B[1] < C[1]$ **then**
Append $B[1]$ to D and remove $B[1]$ from B
else
Append $C[1]$ to D and remove $C[1]$ from C
$z \leftarrow z+$ length$[B]$
return $D, x+y+z$

(iii) **Proof of correctness** Consider now a step in merging. Suppose the pointers are pointing at elements b_i and c_j. Because B and C are sorted , if b_i is appended to D, no new inversions are encountered, since b_i is smaller than everything left in list C, and it comes before all of them. On the other hand, if c_j is appended to D, then it is smaller than all the remaining elements in B, and it comes after all of them, so we increase the count of inversions by the number of elements remaining in B.

(iv) **Running time analysis** In each recursive call, we merge the two sorted lists and count the inversions in $O(n)$. The running time is given by $T(n) = 2T(n/2) + O(n)$ which is $O(n \log n)$ by the master theorem.

4 (★) Complex numbers review

Briefly justify your answers to parts (b) and (c).

(a) Write each of the following numbers in the form $\rho(\cos\theta + i\sin\theta)$ (for real ρ and θ):

(i) $-\sqrt{3}+i$

(ii) The three 3-rd roots of unity

(iii) The sum of your answers to the previous item

(b) Let $sqrt(x)$ represent one of the complex square roots of x, so that $sqrt(x) = \pm\sqrt{x}$. What are the possible values of $sqrt(sqrt(-1))$?

You can use any notation for complex numbers, e.g., rectangular, polar, or complex exponential notation.

Solution:

(a) (i) $-\sqrt{3}+i = 2(\cos\frac{5\pi}{6} + i\sin\frac{5\pi}{6})$

(ii) $(\cos 0 + i\sin 0), (\cos\frac{2\pi}{3} + i\sin\frac{2\pi}{3}), (\cos\frac{4\pi}{3} + i\sin\frac{4\pi}{3})$

(iii) 0

(b) $\sqrt{-1} = \pm i$;
$\sqrt{i} = \pm\frac{\sqrt{2}}{2}(1+i)$, $\sqrt{i} = \pm\frac{\sqrt{2}}{2}(1-i)$.

Alternatively, $-1 = \cos\pi + i\sin\pi = \cos 3\pi + i\sin 3\pi$.
So, $\sqrt{\cos\pi + i\sin\pi} = \{(\cos\frac{\pi}{2} + i\sin\frac{\pi}{2}), (\cos\frac{3\pi}{2} + i\sin\frac{3\pi}{2})\}$.
Therefore:
$\sqrt{(\cos\frac{\pi}{2} + i\sin\frac{\pi}{2})} = \sqrt{(\cos\frac{5\pi}{2} + i\sin\frac{5\pi}{2})} = \{(\cos\frac{\pi}{4} + i\sin\frac{\pi}{4}), (\cos\frac{5\pi}{4} + i\sin\frac{5\pi}{4})\}$, and
$\sqrt{(\cos\frac{3\pi}{2} + i\sin\frac{3\pi}{2})} = \sqrt{(\cos\frac{7\pi}{2} + i\sin\frac{7\pi}{2})} = \{(\cos\frac{3\pi}{4} + i\sin\frac{3\pi}{4}), (\cos\frac{7\pi}{4} + i\sin\frac{7\pi}{4})\}$.