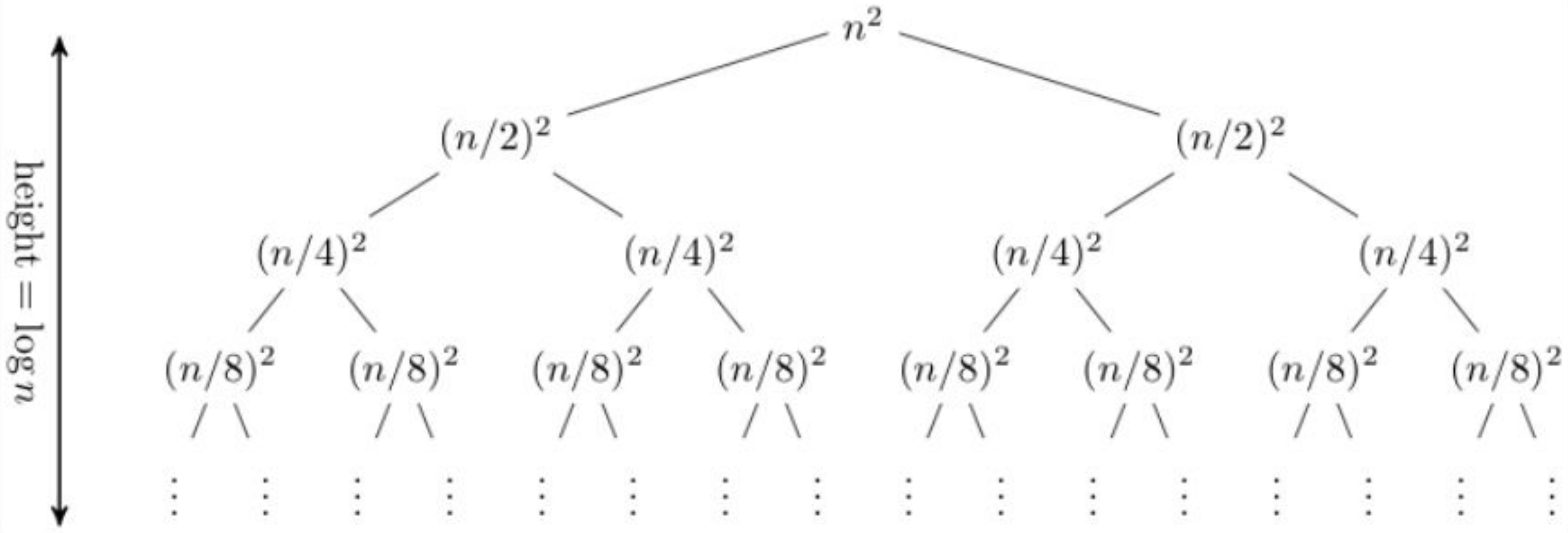
ECS 122A

Lecture 3

Recursion Trees

$T(n) = 2T(n/2) + n^2$.

The recursion tree for this recurrence has the following form:

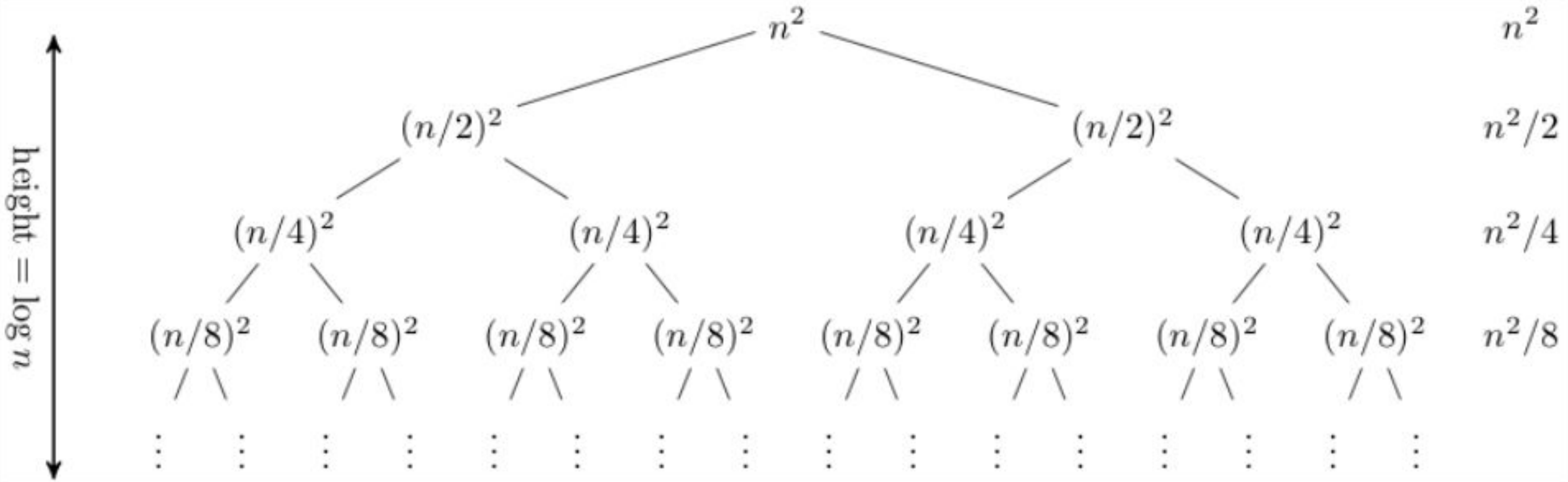


As n goes to infinity, the absolute value of r must be less than one for the series to converge. The sum then becomes

$$a + ar + ar^2 + ar^3 + ar^4 + \cdots = \sum_{k=0}^{\infty} ar^k = \frac{a}{1-r}, \text{ for } |r| < 1.$$

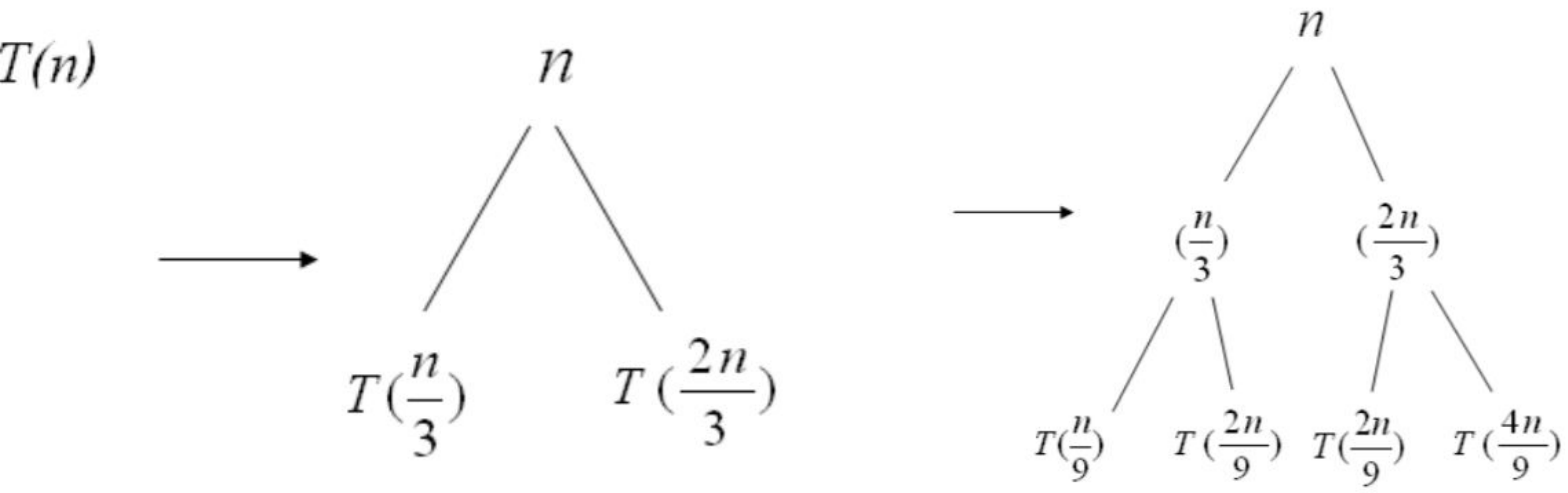
$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = \sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n = \frac{\frac{1}{2}}{1 - \frac{1}{2}} = 1.$$

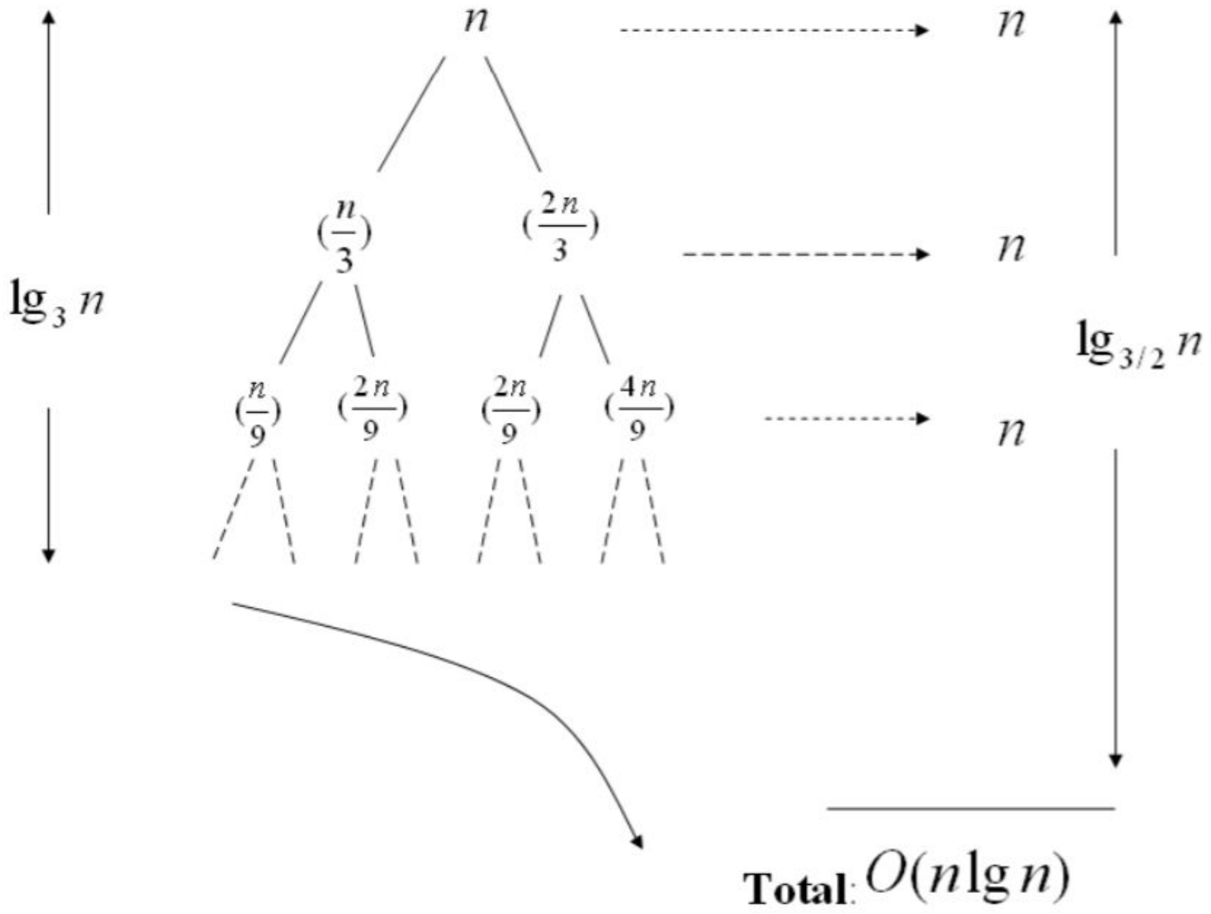
In this case, it is straightforward to sum across each row of the tree to obtain the total work done at a given level:



This a geometric series, thus in the limit the sum is $O(n^2)$. The depth of the tree in this case does not really matter; the amount of work at each level is decreasing so quickly that the total is only a constant factor more than the root.

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





n
$(\frac{n}{3})$
$(\frac{2n}{3})$
$(\frac{n}{9})$
$(\frac{2n}{9})$
$(\frac{2n}{9})$
$(\frac{4n}{9})$
n
n
n
$\lg_3 n$
$\lg_{3/2} n$
Total: $O(n \lg n)$

Note About Big-O w

Why is this true? Look at O(n) ..

$$
\begin{aligned}
&given\\
&f_1(n) = O(g_1(n))\\
&f_2(n) = O(g_2(n))\\
&then\\
&f_1(n) + f_2(n) = O(\boldsymbol{max}(g_1(n), g_2(n)))
\end{aligned}
$$

Theorem 4.1 (Master Theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

Then $T(n)$ can be bounded asymptotically as follows:

1. If $f(n) = O(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{log_b a})$.

2. If $f(n) = \Theta(n^{log_b a})$, then $T(n) = \Theta(n^{log_b a}\ lgn)$.

3. If $f(n) = \Omega(n^{log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Example 1

$$\begin{aligned} T(n) &= T(n/5) + 1 \\ n^{log_b a} &= n^{log_5 1} = 1 \\ f(n) &= 1 \end{aligned}$$

$$n^{log_b a} \leftrightarrow f(n) : \text{case } 2 \Rightarrow T(n) = \Theta(lgn)$$

Simplified Masters

A recurrence relation of the following form:

$$\begin{aligned} T(n) &= c \quad n < c_1 \\ &= aT(n/b) + \Theta(n^i), \quad n \geq c_1 \end{aligned}$$

Has as its solution:

1) If $a > b^i$ then $T(n) = \Theta(n^{\log_b a})$ (Work is increasing as we go down the tree, so this is the number of leaves in the recursion tree).

2) If $a = b^i$ then $T(n) = \Theta(n^i \log_b n)$ (Work is the same at each level of the tree, so the work is the height, $\log_b n$, times work/level).

3) If $a < b^i$ then $T(n) = \Theta(n^i)$ (Work is going down as we go down the tree, so dominated by the initial work at the root).

Let's take a look at divide and conquer algorithms

Reminder

Divide and Conquer

Divide: Divide the problem into a number of subproblems that are smaller instances of the same problem.

Conquer: the subprob. By solving them recursively (if sub problem is small enough solve it in a straightforward manner)

Combine: combine the solutions of the subproblems into the solution of the original problem.

Alg.: MERGE-SORT(A, p, r)

if p < r

then $q \leftarrow \lfloor (p + r)/2 \rfloor$

MERGE-SORT(A, p, q)

MERGE-SORT(A, q + 1, r)

MERGE(A, p, q, r)

Matrix-matrix multiplication

- Problem:

  Given $n \times n$ *matrices* A *and* B, *compute the product* $C = A \cdot B$.

- Traditional method: (i, j, k)-triple-loop

```
for i = 1 to n
    for j = 1 to n
        C(i,j) = 0
        for k = 1:n
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
        end
    end
end
```

Complexity:

$$T(n) = \sum_{i=1}^{n}(\sum_{j=1}^{n}(\sum_{k=1}^{n} 2)) = 2n^3 = \Theta(n^3)$$

Matrix Multiplication

Imagine that A and B are each partitioned into four square sub-matrices, each submatrix having dimensions $\frac{n}{2} \times \frac{n}{2}$.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

, where

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

2. Complexity:

$$T(n) = 8 \cdot T(\frac{n}{2}) + \Theta(n^2) = \Theta(n^3).$$

Same cost as the traditional method, No improvement

Strassen "observed" that:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{bmatrix}$$

, where

$$\begin{aligned} P_1 &= A_{11}(B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12})B_{22} \\ P_3 &= (A_{21} + A_{22})B_{11} \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21})(B_{11} + B_{12}) \end{aligned}$$

Matrix-matrix multiplication

- Correctness: straightfoward verification
- Strassen's method – complexity

$$T(n) = 7 \cdot T(\frac{n}{2}) + \Theta(n^2) = \Theta(n^{\lg 7}).$$

The maximum-subarray problem

Problem:

Input: *an array* $A[1...n]$ *of (positive/negative) numbers.*

Output: *(1) Indices* i *and* j *such that the subarray* $A[i...j]$ *has the greatest sum of any nonempty contiguous subarray of* A*, and (2) the sum of the values in* $A[i...j]$.

Note: Maximum subarray might not be unique, though its value is, so we speak of **a** maximum subarray, rather than **the** maximum subarray.

The maximum-subarray problem

Example 1:

Day	0	1	2	3	4
Price	10	11	7	10	6
Change $A[...]$		1	-4	3	-4

maximum-subarray: $A[3]$ $(i = j = 3)$ and Sum $= 3$

Example 2:

Day	0	1	2	3	4	5	6
Price	10	11	7	10	14	12	18
Change $A[...]$		1	-4	3	4	-2	6

maximum-subarray: $A[3...6]$ $(i = 3, j = 6)$ and Sum $= 11$.

The maximum-subarray problem

Algorithm 1: Solve by Brute-Force:

- Total number of subarrays $A[i...j]$:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{1}{2}n(n-1) = \Theta(n^2)$$

  plus the arrays of length $= 1$.

- Check all subarrays: $\Theta(n^2)$

The maximum-subarray problem

Algorithm 2: Solve by Divide-and-Conquer:

- Generic problem: Find a maximum subarray of $A[low...high]$
- Initial call: $low = 1$ and $high = n$
- DC strategy:
  1. Divide $A[low...high]$ into two subarrays of as equal size as possible by finding the midpoint mid
  2. Conquer:
     (a) finding maximum subarrays of $A[low...mid]$ and $A[mid + 1...high]$
     (b) finding a max-subarray that crosses the midpoint
  3. Combine: returning the max of the three
- This strategy works because any subarray must either lie entirely in one side of midpoint or cross the midpoint.

The maximum-subarray problem

```
MaxSubarray(A,low,high)
if high == low    // base case: only one element
   return (low, high, A[low])
else
   // divide
   mid = floor( (low + high)/2 )
   // conquer
   (leftlow,lefthigh,leftsum) = MaxSubarray(A,low,mid)
   (rightlow,righthigh,rightsum) = MaxSubarray(A,mid+1,high)
   (xlow,xhigh,xsum) = MaxXingSubarray(A,low,mid,high)
   // combine
   if leftsum >= rightsum and leftsum >= xsum
      return (leftlow,lefthigh,leftsum)
   else if rightsum >= leftsum and rightsum >= xsum
      return (rightlow,righthigh,rightsum)
   else
      return (xlow,xhigh,xsum)
   end if
end if
```

The maximum-subarray problem

```
MaxXingSubarray(A,low,mid,high)
leftsum = -infty; sum = 0   // Find max-subarray of A[i..mid]
for i = mid downto low
    sum = sum + A[i]
    if sum > leftsum
       leftsum = sum
       maxleft = i
    end if
end for
rightsum = -infty; sum = 0  // Find max-subarray of A[mid+1..j]
for j = mid+1 to high
    sum = sum + A[j]
    if sum > rightsum
       rightsum = sum
       maxright = j
    end if
end for
// Return the indices i and j and the sum of two subarrays
return (maxleft,maxright,leftsum + rightsum)
```

The maximum-subarray problem

Remarks:

1. Initial call: `MaxSubarray(A,1,n)`
2. Base case is when the subarray has only 1 element.
3. Divide by computing `mid`.
   Conquer by the two recursive calls to `MaxSubarray`. and a call to `MaxXingSubarray`
   Combine by determining which of the three results gives the maximum sum.
4. Complexity:

$$
\begin{aligned}
T(n) &= 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n) + \Theta(1) \\
&= \Theta(n \lg n)
\end{aligned}
$$

5. Question: What does `MaxSubarray` return when all elements of A are negative?