Hanoi Algorithm

$\textsc{Hanoi}(n, src, dst, tmp)$:
  if $n > 0$
    $\textsc{Hanoi}(n-1, src, tmp, dst)$
    move disk n from src to dst
    $\textsc{Hanoi}(n-1, tmp, dst, src)$

Reduction = Delegation

Say we want to build a minimal DFA from a regular expression

- Reg Exp $\longrightarrow$ NFA (thompson)
- NFA $\longrightarrow$ DFA (subset)
- DFA $\longrightarrow$ min DFA (Moore)

3 Steps. Not important how any of those work, as long as we are guaranteed they work

A	L	G	O	R	I	T	H	M	S

Quicksort:

- choose a pivot element from the array
- partition the array into three subarrays: one with elements smaller than pivot, one the pivot itself, one with elements larger than pivot.
- Recursively quick sort the first and last subarray
- How to choose pivot?

Quicksort:

$\textsc{QuickSort}(A[1..n])$:
  if $(n > 1)$
    Choose a pivot element $A[p]$
    $r \leftarrow \textsc{Partition}(A, p)$
    $\textsc{QuickSort}(A[1..r-1])$
    $\textsc{QuickSort}(A[r+1..n])$

Partition (linear time):

$\textsc{Partition}(A[1..n], p)$:
  swap $A[p] \leftrightarrow A[n]$
  $i \leftarrow 0$
  $j \leftarrow n$
  while $(i < j)$
    repeat $i \leftarrow i+1$ until $(i \geq j$ or $A[i] \geq A[n])$
    repeat $j \leftarrow j-1$ until $(i \geq j$ or $A[j] \leq A[n])$
    if $(i < j)$
      swap $A[i] \leftrightarrow A[j]$
  swap $A[i] \leftrightarrow A[n]$
  return i

ALGORITHMS

Mergesort:

- Divide the input array into two subarrays of roughly equal size
- Recursively merge sort each of the subarrays
- Merge the two newly sorted subarrays into a single sorted array

Merge:

```
MERGE(A[1..n], m):
  i ← 1; j ← m+1
  for k ← 1 to n
      if j > n
          B[k] ← A[i]; i ← i+1
      else if i > m
          B[k] ← A[j]; j ← j+1
      else if A[i] < A[j]
          B[k] ← A[i]; i ← i+1
      else
          B[k] ← A[j]; j ← j+1

  for k ← 1 to n
      A[k] ← B[k]
```

Loop = recursion

- When writing actual code easier to unfold the recursion
- When proving correctness easier to use induction (=recursion)

Mergesort:

```
MERGESORT(A[1..n]):
  if n > 1
      m ← ⌊n/2⌋
      MERGESORT(A[1..m])
      MERGESORT(A[m+1..n])
      MERGE(A[1..n], m)
```

Base cases:

- When size of arrays to merge is 1
- When size of arrays is less than10 and then brute force
- It doesn't matter, no need to optimize

Running time of Quicksort

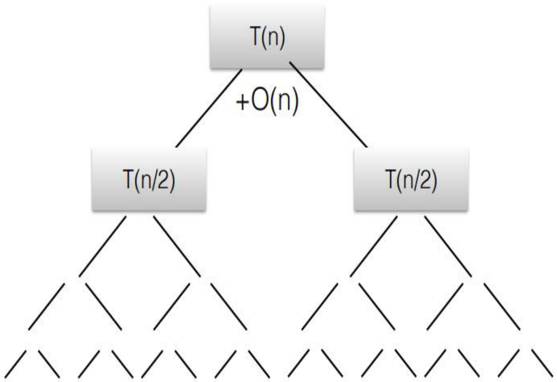
- What is the running time T(n) of quicksort?
- $O(n^2)$ time! (If I choose the smallest pivot)
  - $T(n)=O(n)+T(n-1)$

    $= O(n^2)$

Running time of Mergesort

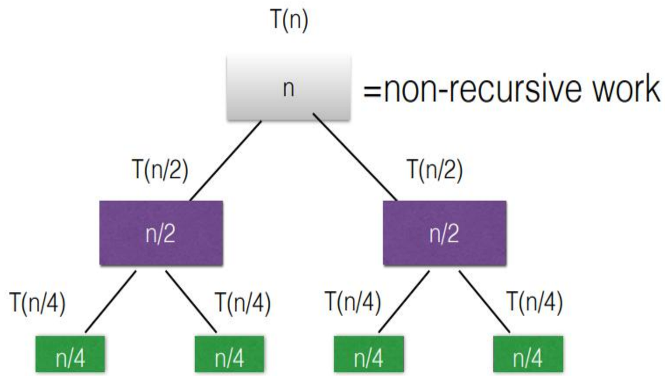
- What is the running time T(n) of mergesort?
- O(nlogn) time!
  - $T(n)=2T(n/2)+O(n)$
  - proof by induction if I know answer
  - recursion tree!

Running time of Mergesort



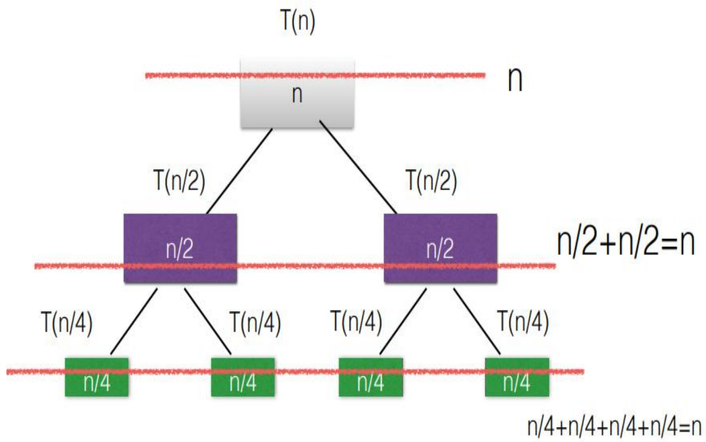
Complete binary tree

Running time of Mergesort



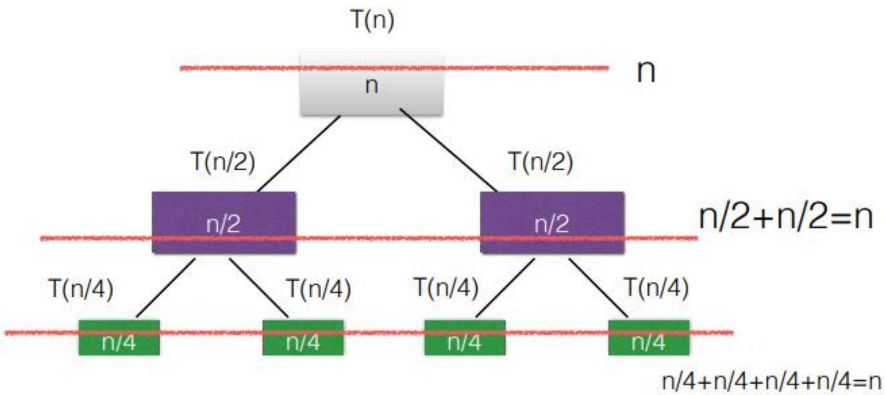
- Leave all the O() till the very end.
- Goal is to sum up all the quantities in all the nodes.

Running time of Mergesort



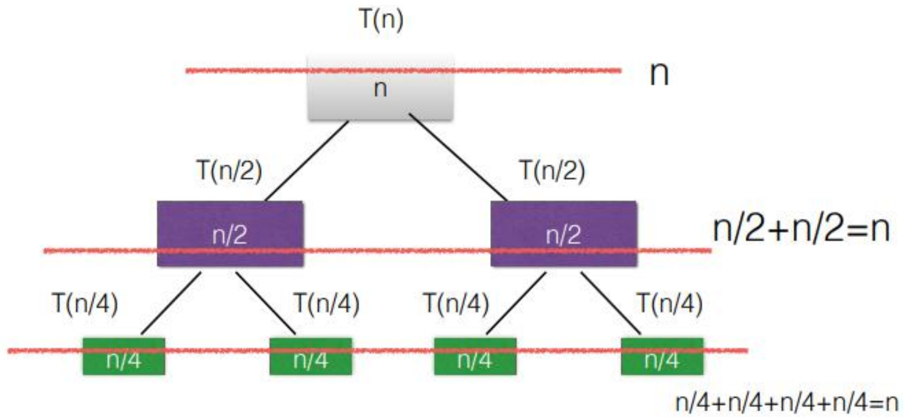
- T(n)=2T(n/2)+O(n)

Running time of Mergesort



- T(n)=2T(n/2)+O(n)
- Total amount of work at level k= total amount of work at level k-1 (induction).

Running time of Mergesort



- T(n)=2T(n/2)+O(n)
- Total amount of work = n x (height of the tree) =n logn

Recursive Algorithm

```
Hanoi(n, src, dest, tmp):
    if (n > 0) then
        Hanoi(n − 1, src, tmp, dest)
        Move disk n from src to dest
        Hanoi(n − 1, tmp, dest, src)
```

Recursive Algorithm

```
Hanoi(n, src, dest, tmp):
    if (n > 0) then
        Hanoi(n − 1, src, tmp, dest)
        Move disk n from src to dest
        Hanoi(n − 1, tmp, dest, src)
```

$T(n)$: time to move n disks via recursive strategy

Recursive Algorithm

```
Hanoi(n, src, dest, tmp):
    if (n > 0) then
        Hanoi(n − 1, src, tmp, dest)
        Move disk n from src to dest
        Hanoi(n − 1, tmp, dest, src)
```

$T(n)$: time to move n disks via recursive strategy

$$T(n) = 2T(n-1) + 1 \qquad n > 1 \qquad \text{and } T(1) = 1$$

Analysis

$$\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
&= 2^2T(n-2) + 2 + 1 \\
&= \ldots \\
&= 2^iT(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 1 \\
&= \ldots \\
&= 2^{n-1}T(1) + 2^{n-2} + \ldots + 1 \\
&= 2^{n-1} + 2^{n-2} + \ldots + 1 \\
&= (2^n - 1)/(2 - 1) = 2^n - 1
\end{aligned}$$

Solving Recurrences

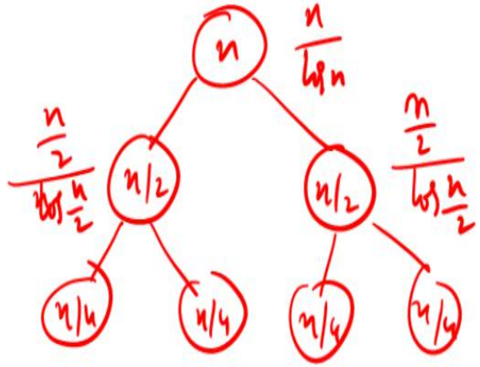
Two general methods:

1. Recursion tree method: need to do sums
   1. elementary methods, geometric series
   2. integration
2. Guess and Verify
   1. guessing involves intuition, experience and trial & error
   2. verification is via induction

Recurrence: Example I

1. Consider $\mathbf{T(n) = 2T(n/2) + n/\log n}$.

$T(2) = 1$



$n \log\log n$

$$\frac{n}{\log n} + 2\cdot\frac{\frac{n}{2}}{\log\frac{n}{2}} + 4\cdot\frac{\frac{n}{4}}{\log\frac{n}{4}} + \cdots \quad n\left[\frac{1}{\log n} + \frac{1}{\log n - 1} + \frac{1}{\log n - 2} + \cdots 1\right]$$

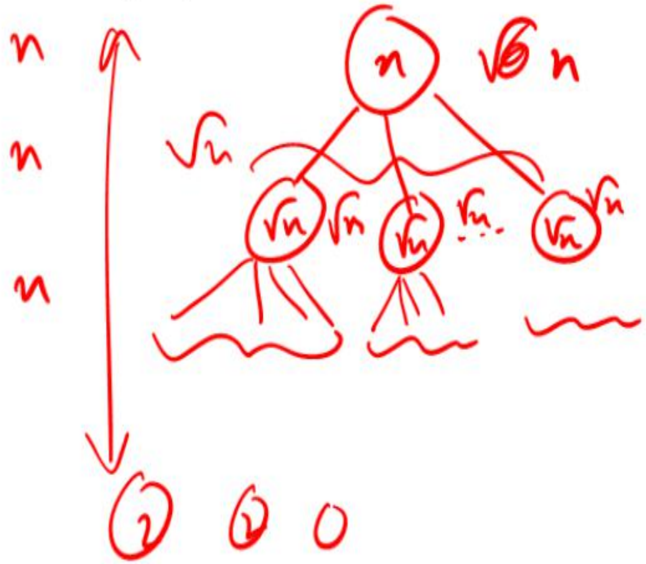
Recurrence: Example I

1. Consider $\mathbf{T(n) = 2T(n/2) + n/\log n}$.
2. Construct recursion tree, and observe pattern. $\mathbf{i}$th level has $\mathbf{2^i}$ nodes, and problem size at each node is $\mathbf{n/2^i}$ and hence work at each node is $\frac{n}{2^i}/\log\frac{n}{2^i}$.
3. Summing over all levels

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log n - 1} 2^i \left[\frac{(n/2^i)}{\log(n/2^i)}\right] \\ &= \sum_{i=0}^{\log n - 1} \frac{n}{\log n - i} \\ &= n \sum_{j=1}^{\log n} \frac{1}{j} = nH_{\log n} = \Theta(n \log\log n) \end{aligned}$$

Recurrence: Example III

1. Consider $\mathbf{T(n) = \sqrt{n}T(\sqrt{n}) + n}$.



$n \log\log n$

Recurrence: Example III

1. Consider $\mathbf{T(n) = \sqrt{n}T(\sqrt{n}) + n}$.
2. Using recursion trees: number of levels $\mathbf{L = \log\log n}$
3. Work at each level? Root is $\mathbf{n}$, next level is $\mathbf{\sqrt{n} \times \sqrt{n} = n}$. Can check that each level is $\mathbf{n}$.
4. Thus, $\mathbf{T(n) = \Theta(n \log\log n)}$

Multiplying Numbers

Problem: Given two $\mathbf{n}$-digit numbers $\mathbf{x}$ and $\mathbf{y}$, compute their product.

Grade School Multiplication

Compute "partial product" by multiplying each digit of $\mathbf{y}$ with $\mathbf{x}$ and adding the partial products.

$$\begin{array}{r} 3141 \\ \times 2718 \\ \hline 25128 \\ 3141 \\ 21987 \\ 6282 \\ \hline 8537238 \end{array}$$

A Trick of Gauss

Carl Friedrich Gauss: 1777–1855 "Prince of Mathematicians"

Observation: Multiply two complex numbers: $(a + bi)$ and $(c + di)$

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

How many multiplications do we need?

Only 3! If we do extra additions and subtractions.
Compute $ac, bd, (a + b)(c + d)$. Then
$(ad + bc) = (a + b)(c + d) - ac - bd$

Time Analysis of Grade School Multiplication

1. Each partial product: $\Theta(n)$
2. Number of partial products: $\Theta(n)$
3. Addition of partial products: $\Theta(n^2)$
4. Total time: $\Theta(n^2)$

A Trick of Gauss

Carl Friedrich Gauss: 1777–1855 "Prince of Mathematicians"

Observation: Multiply two complex numbers: $(a + bi)$ and $(c + di)$

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

Divide and Conquer

Assume $\mathbf{n}$ is a power of $\mathbf{2}$ for simplicity and numbers are in decimal.

Split each number into two numbers with equal number of digits

1. $x = x_{n-1}x_{n-2}\dots x_0$ and $y = y_{n-1}y_{n-2}\dots y_0$
2. $x = x_{n-1}\dots x_{n/2}0\dots 0 + x_{n/2-1}\dots x_0$
3. $x_L = 10^{n/2}x_L$ where $x_L = x_{n-1}\dots x_{n/2}$ and $x_R = x_{n/2-1}\dots x_0$
4. Similarly $y = 10^{n/2}y_L + y_R$ where $y_L = y_{n-1}\dots y_{n/2}$ and $y_R = y_{n/2-1}\dots y_0$

Example

$$\begin{aligned} 1234 \times 5678 &= (100 \times 12 + 34) \times (100 \times 56 + 78) \\ &= 10000 \times 12 \times 56 \\ &\quad +100 \times (12 \times 78 + 34 \times 56) \\ &\quad +34 \times 78 \end{aligned}$$

Divide and Conquer

Assume n is a power of 2 for simplicity and numbers are in decimal.

1. $x = x_{n-1}x_{n-2}\dots x_0$ and $y = y_{n-1}y_{n-2}\dots y_0$
2. $x = 10^{n/2}x_L + x_R$ where $x_L = x_{n-1}\dots x_{n/2}$ and $x_R = x_{n/2-1}\dots x_0$
3. $y = 10^{n/2}y_L + y_R$ where $y_L = y_{n-1}\dots y_{n/2}$ and $y_R = y_{n/2-1}\dots y_0$

Therefore

$$\begin{aligned} xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

$T(n) = 4T(\frac{n}{2}) + 6n \quad T(1) = 1$

Time Analysis

$$\begin{aligned} xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

4 recursive multiplications of number of size $n/2$ each plus 4 additions and left shifts (adding enough 0's to the right)

$$T(n) = 4T(n/2) + O(n) \qquad T(1) = O(1)$$

Improving the Running Time

$$\begin{aligned} xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

Gauss trick: $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

$T(n) = 3T(\frac{n}{2}) + n$

Improving the Running Time

$$\begin{aligned} xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

Gauss trick: $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

Recursively compute only $x_L y_L, x_R y_R, (x_L + x_R)(y_L + y_R)$.

Time Analysis

Running time is given by

$$T(n) = 3T(n/2) + O(n) \qquad T(1) = O(1)$$

which means

Improving the Running Time

$$\begin{aligned} xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

Gauss trick: $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

Recursively compute only $x_L y_L, x_R y_R, (x_L + x_R)(y_L + y_R)$.

Time Analysis

Running time is given by

$$T(n) = 3T(n/2) + O(n) \qquad T(1) = O(1)$$

which means $T(n) = O(n^{\log_2 3}) = O(n^{1.585})$