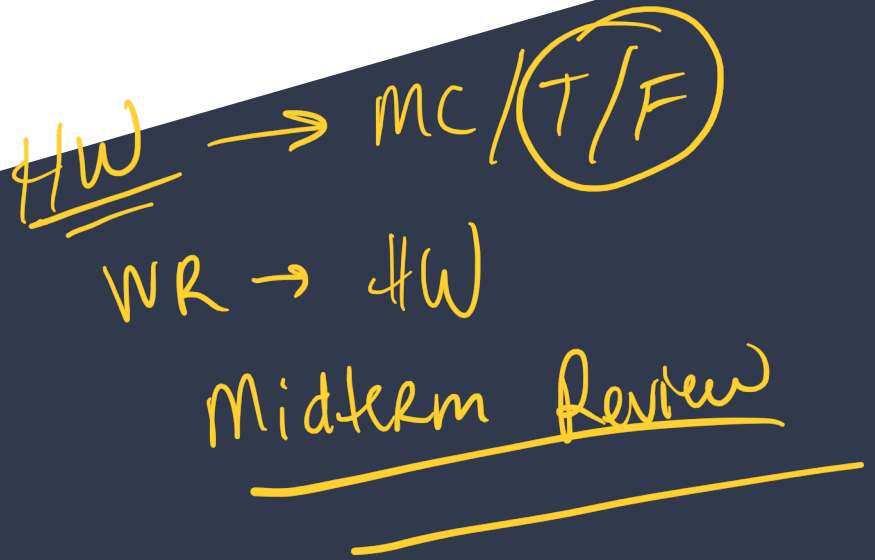
Algorithm Design & Analysis

Part 3: Recursive Algorithms and Recurrence Relations



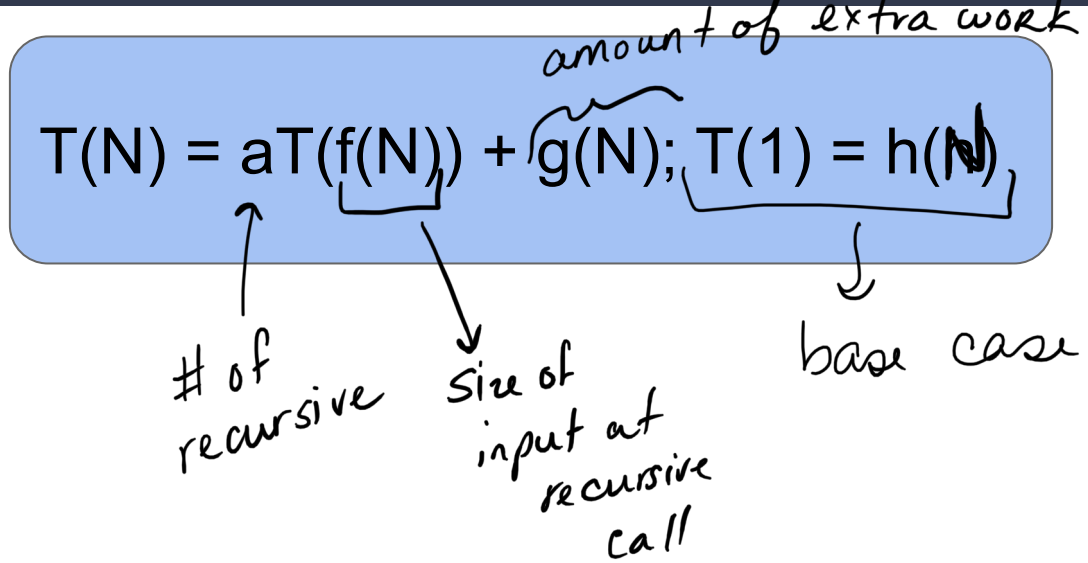
Recursive Algorithms

- A recursive algorithm consists of one or more base cases and one or more calls to itself on a smaller set of data than the original input.
- Example: Binary Search
  - Base Case: If the size of the search array is 1, then you check that one element.
  - Recursive Step: If not, you check the median and if you find what you're looking for, you're done. Otherwise, you do the process again (recursive call) on one half of the array.

Analyzing Recursive Algorithms

- **The runtime of a call to a recursive algorithm on an input size of N is going to equal**
  - **the runtime of any recursive calls, plus**
  - **the runtime of any other work being done**
- We can model this with a recursive function called a recurrence relation.
- For example, binary search…
  - Let T(N) be the runtime of the algorithm on an array of size N.
  - In the worst case, we would have to do 1 recursive call on half the data, which would have a runtime of T(N/2).
  - Additionally, we also do a constant amount of “other work.”
  - So the total runtime would be $T(N) = T(N/2) + 1$ with a base case of $T(1) = 1$.

What does each part of this mean?



Solving Recurrence Relations: Expansion & Summation

Assume $N = 2^k$

$$T(N) = T(N/2) + 1 \; ; \; T(1) = 1$$

$$= T(N/4) + 1 + 1$$

$$= T(N/8) + 1 + 1 + 1$$

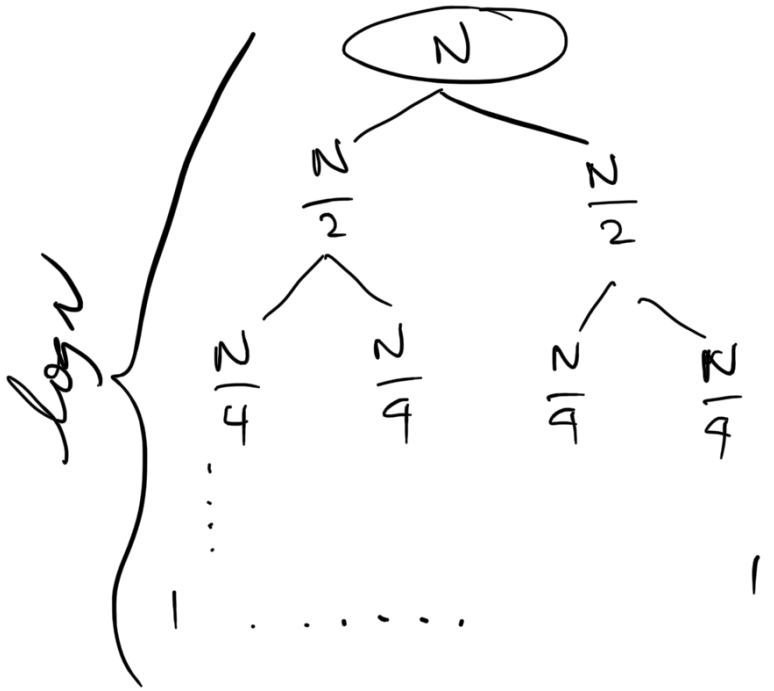
$$\vdots$$

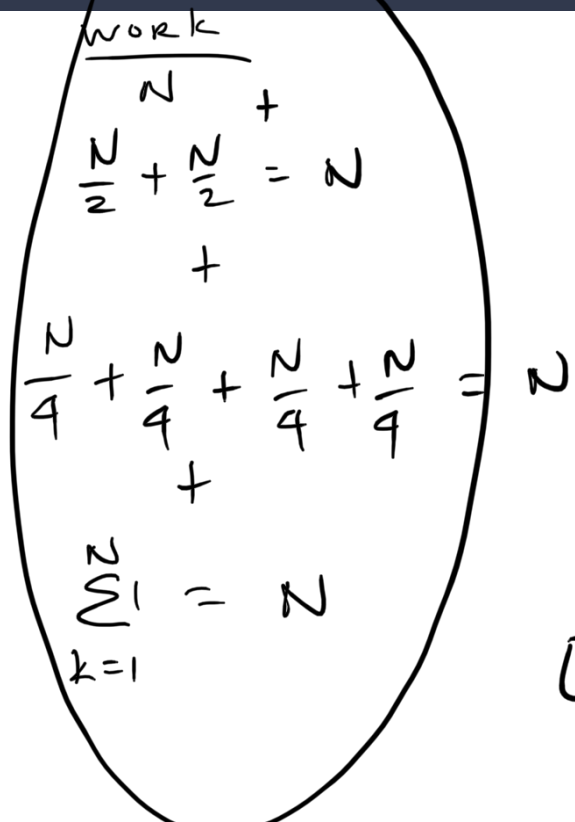
$$= T(N/N) + 1 + 1 \ldots + 1 + 1$$

$$= 1 + \sum_{k=1}^{\log_2 N} 1 = \log_2 N + 1$$

Solving Recurrence Relations: Using a Tree

$$T(N) = 2T(N/2) + N; \quad T(1) = 1$$





$O(N \log N)$

Solving Recurrence Relations: Formal Proof

$T(N) = 2T(\lfloor N/2 \rfloor) + N \; ; \; T(1) = 1$

Prove: $T(N)$ is $O(N \log N)$

Conjecture: $T(N) \leq N \log N + N$ for $N \geq 1$

Basis Step:

$T(1) = 1 \leq 1 \log 1 + 1$ ✓

Inductive Step:

IH: Assume that $T(j) \leq j \log j + j$ for $1 \leq j \leq k-1$.

$$T(k) = 2T(\lfloor k/2 \rfloor) + k$$
$$\leq 2\left(\lfloor k/2 \rfloor \log \lfloor k/2 \rfloor + \lfloor k/2 \rfloor\right) + k \quad \text{by the IH}$$
$$\leq 2\left(k/2 \log (k/2) + k/2\right) + k$$
$$= k \log (k/2) + k + k = k \log k - k \log 2 + k + k = k \log k + k$$

Solving Recurrence Relations: Using the Master Theorem

Given:

- a and b are integers
- $a \geq 1$ and $b > 1$
- c is a positive real number
- d is a nonnegative real number
- $N = b^k$ where k is a positive integer
- $T(N)$ is an increasing function
- $T(N) = aT(N/b) + cN^d$

$T(N) = T(N/2) + 1$

$a = 1$, $b = 2$, $d = 0$

$1 = 2^0 = 1$

$O(N^0 \log N) = O(\log N)$

$T(N) = 2T(N/2) + 1 \rightarrow$ $a = 2, b = 2, d = 0$

$2 > 2^0 = 1$

$O(N^{\log_2 2}) = O(N)$

Then:

- if $a < b^d$, $T(N)$ is $O(N^d)$
- ✓ if $a = b^d$, $T(N)$ is $O(N^d \log N)$
- if $a > b^d$, $T(N)$ is $O(N^{\log_b a})$

$T(N) = 2T(N/2) + N$

$a = 2, b = 2, d = 1$

$2 = 2^1$

$O(N^1 \log N) = O(N \log N)$

Write and analyze a recursive binary search algorithm.

Input: Sorted array A of size N and a search element x
Output: the location of x in A OR -1 if it doesn't exist
find(x, A, 0, N-1)

```
proc find(element x, Array A, int i, int j)
    if i = j then if A[i] = x then i else -1
    m = (i+j)/2
    if A[m] = x then m
    else if A[m] < x then find(x, A, m+1, j)
    else find(x, A, i, m-1)
```

$T(N) = T(N/2) + 1$; $T(1) = 1$

Write and analyze a recursive linear search algorithm.

Input: Array A of size N, element x
Output: location of x in A OR -1

find(x, A, 0)

```
proc find(element x, Array A, int i):
    if i = N-1 then if A[i] = x then i else -1
    if A[i] = x then i
    else find(x, A, i+1)
```

$$T(N) = T(N-1) + 1 \; ; \quad T(1) = 1$$

$$= T(N-2) + 1 + 1 = T(N-2) + 2$$

$$= T(N-3) + 1 + 1 + 1 = \ldots = T(1) + 1 \ldots + 1 + 1$$

$O(N)$

Write and analyze a recursive algorithm for calculating the n^{th} power of 2.

```
pow(n):
if n=0 then 1
2 * pow(n-1)
```

$T(N) = T(N-1) + 1$

$\rightarrow O(N)$

```
pow(n):
if n=0 then 1
if n=1 then 2
x = pow(⌊n/2⌋)
if n is even:
    x * x
else x*x*2
```

$T(N) = T(N/2) + 1$

$\rightarrow O(\log N)$

Write and analyze a recursive algorithm for calculating the n^{th} fibonacci number.