Lecture 25

- Harder to determine running time for recursive functions

- Running time is represented by an equation in terms of its value on a smaller input

Ex. $T(n) = T(n-1) + c$

① Find an explicit formula of the expression

② Bound the recurrance by an expression that involves n

ex. Recursive algorithm that loops through the input to eliminate one item

$T(n) = T(n-1) + 1$ $\quad T(n) = T(n-1) + n$, etc.

ex. Recursive algorithm that halves the input

$T(n) = T(n/2) + c$ $\quad T(n) = T(n/2) + n$, etc.

ex. Recursive algorithm that splits input into 2 halves

$T(n) = 2T(n/2) + 1$, etc

We consider the recursive prog. shown below, which computes the factorial func $n!$

```
int fact(int n)
  if (n<=1)
     return 1
  else
     return n*fact(n-1);
```

constant time : C_1

cost time : constant time C_2 + time taken by func. fact

same problem of size $(n-1)$

$$T(n) = C_2 + T(n-1), \ n > 1$$

$$T(1) = C_1$$

Methods for Solving Recurrances

1. Substitution/Induction Method
2. Iteration Method
3. Recursion-tree method
4. Master Method - skip

Substitution/Induction
Iteration
Recursion-Tree

① Substitution / Induction Method

* use mathematical induction

Ex. Solve $T(n) = T(n-1) + C_2$, $n > 1$

$T(1) = C_1$ (Factorial)

Note that $T(1) = C_1$

$T(2) = T(1) + C_2 = C_1 + C_2$

$T(3) = T(2) + C_2 = C_1 + C_2 + C_2$

$= C_1 + 2(C_2)$

$\vdots$

This is not a proof! $\longrightarrow$ $T(n) = C_1 + (n-1)C_2$

Induction:

Base case : $T(1) = C_1$

Assume that $T(k) = C_1 + (k-1)C_2$ for $k < n$

We prove that $T(k+1) = C_1 + kC_2$ (Easy)

$$T(k+1) \overset{(*)}{=} T(k) + C_2$$
$$= C_1 + (k-1)C_2 + C_2$$
$$= C_1 + k \cdot C_2$$

The running time for the recursive program for fact. $(n!)$ is $O(n)$

Ex. Solve $T(1)=1$,

$T(n) = 3T(n-1)+4, n>1$

Answer = $T(n) = 3^n - 2$

Binary Search

Ex. Solve $T(1) = 0$,

$$T(n) = T(n/2)+1, \quad n>1$$

Assuming n is a power of 2)

Sol. By repeated substitution, we have

$$T(1) = 0 = \log 1$$

$$T(2) = T(2/2)+1 = T(1)+1 = 0+1 = \log 2$$

$$\cancel{T(3) = T(3/2)+1 = T(2)+1}$$

NOT ACCEPTABLE

$$T(4) = T(4/2)+1 = T(2)+1 = 1+1 = 2 = \log_2 2^2 = \log 4$$

$\vdots$

$$T(n) = \log(n)$$

Prove that $T(n) = \log(n)$ by induction

Base case: $T(1) = 0 = \log 1$ ✓

Inductive Hypothesis: Assume that $T(k) = \log k$ for all $k < m$

we now show that $T(m) = \log m$

Now, $T(m) = T(m/2) + 1$

$\log_2 1 = 0$

$$= \log\left(\frac{m}{2}\right) + 1$$

$$= \log(m) \cancel{- \log(2) + 1}$$

$$= \log(m)$$

$$\therefore T(n) = O(\log n)$$

Ex. Let $T(n) = T(n-1) + T(n-2) + 1$

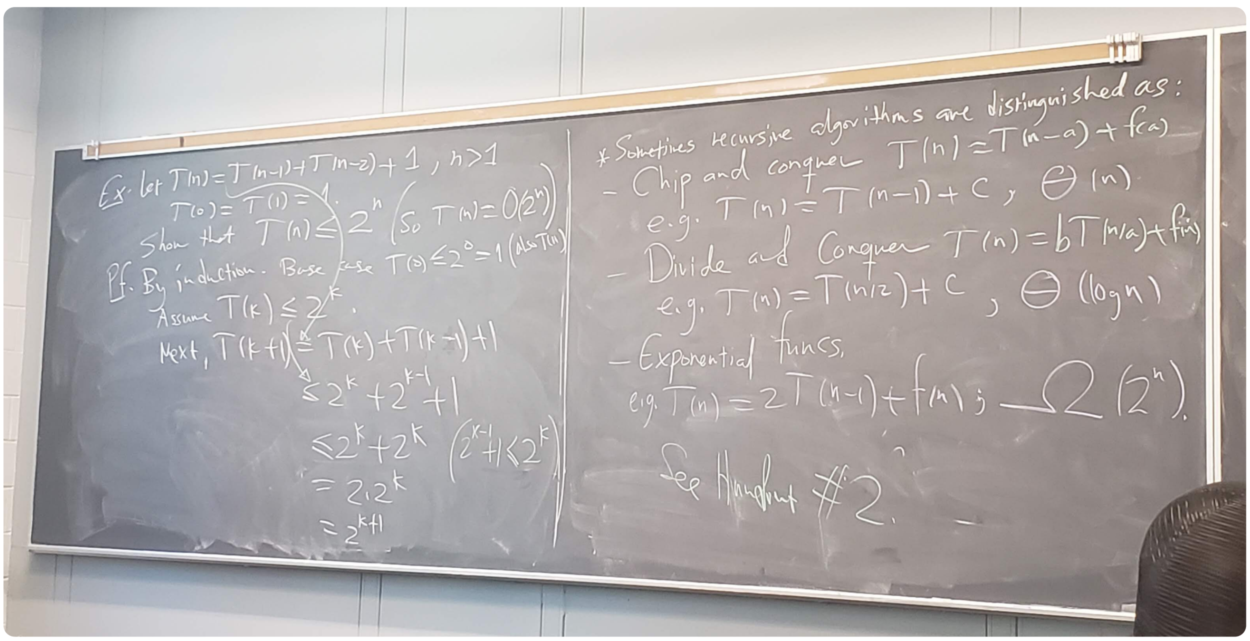
n

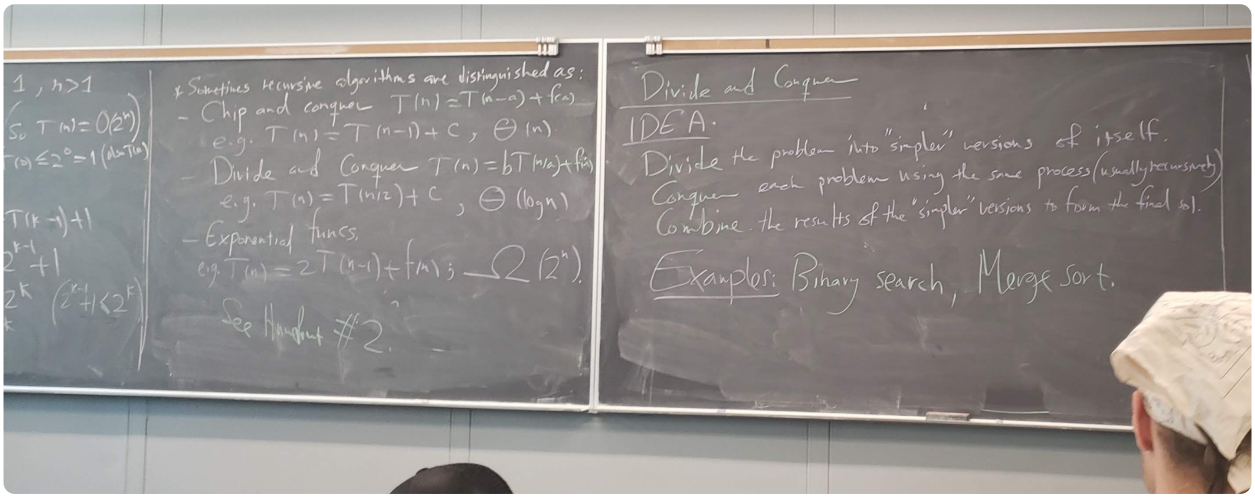
$T(0) = T(1) = 1$

Show that $T(n) \le 2^n$

(so $T(n) = O(2^n)$)

Proof 1




* Sometimes recursive algorithms are distinguished as:
- Chip and conquer T(n) = T(n-a) + f(n)
e.g. T(n) = T(n-1) + C, Θ(n)
- Divide and Conquer T(n) = bT(n/a) + f(n)
e.g. T(n) = T(n/2) + C, Θ(log n)
- Exponential funcs.
e.g. T(n) = 2T(n-1) + f(n); Ω(2^n).
See Handout #2.
Divide and Conquer
IDEA.
Divide the problem into "simpler" versions of itself.
Conquer each problem using the same process (usually recursively)
Combine the results of the "simpler" versions to form the final sol.
Examples: Binary search, Merge sort.