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

  \[ T(n) = T(n-1) + c \]

  1. Find an explicit formula of the expression
  2. Bound the recurrence by an expression that involves \( n \)

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

    \[ T(n) = T(n-1) + 1 \quad T(n) = T(n-1) + n, \text{ etc.} \]

  ex. Recursive algorithm that halves the input

    \[ T(n) = T(n/2) + c \quad T(n) = T(n/2) + n, \text{ etc.} \]

  ex. Recursive algorithm that splits input into 2 halves

    \[ T(n) = 2T(n/2) + 1, \text{ etc.} \]
We consider the recursive program shown below, which computes the factorial function $n!$.

```c
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**

$T(n) = C_2 + T(n-1)$, $n > 1$
$T(1) = C_1$

Methods for solving Recurrences:

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

Substitution/Induction
Iteration
Recursion-Tree
Substitution/Induction Method
   * use mathematical induction

Ex. Solve $T(n) = T(n-1) + C_2$, $n > 1$
    $T(1) = C_1$ (Factorial)

Note that
   $T(1) = C_1$
   $T(2) = T(1) + C_2 = C_1 + C_2$
   $T(3) = T(2) + C_2 = C_1 + C_2 + C_2$
     = $C_1 + 2C_2$
   ...$T(n) = C_1 + (n-1)C_2$

This is not a proof!

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)

$T(k+1) = T(k) + C_2$
     = $C_1 + (k-1)C_2 + C_2$
     = $C_1 + kC_2$

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 $T(1) = 0$,

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

Assuming $n$ is a power of 2.

Sol. By repeated substitution, we have

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

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

$T(3) = T(3/2) + 1 = T(1) + 1 = 0 + 1 = 1 = \log 2$

**NOT ACCEPTABLE**

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

$\vdots$

$T(n) = \log (n)$
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, $T(m) = T(m/2) + 1$

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

$$= \log(m) - \log(2) + 1$$

$$= \log(m)$$

$\therefore T(n) = \Theta(\log n)$
Example: Let \( T(n) = T(n-1) + T(n-2) + 1 \) for \( n \geq 2 \)

\[ T(0) = T(1) = 1 \]

Show that \( T(n) \leq 2^n \)

(\( T(n) = O(2^n) \))

**Proof:**

\[ T(n) = T(n-1) + T(n-2) + 1 \]

Induction:

Base case: \( T(0) = 1 \) and \( T(1) = 1 \), which is true.

Inductive step: Assume \( T(k) \leq 2^k \) for \( k < n \).

Next,\[ T(n) = T(n-1) + T(n-2) + 1 \]

\[ \leq 2^k + 2^k + 1 \]

\[ \leq 2^k + 2^k \left( \frac{2^m}{2^k} \right) \]

\[ = 2^{k+1} \]

By induction, \( T(n) \leq 2^n \).
Divide and Conquer

**IDEA:**
Divide the problem into "simpler" versions of itself.
Conquer each problem using the same process/recursively.
Combine the results of the "simpler" versions into the final answer.

**Examples:** Binary search, Merge sort.

Some recursive algorithms are distinguished as:
- **Chop and compare** $T(n) = T(n/2) + f(n)$
  - e.g., $T(n) = T(n/2) + C, \Theta(n)$
- **Divide and Conquer** $T(n) = \Theta(n \log n)$
  - e.g., $T(n) = T(n/2) + C, \Theta(\log n)$

Suppose $n = 2^k$:
- Example functions
  - $T(n) = \Theta(n^{3/2}) - \Theta(2^n/n^2)$

See Handout #2.