CST207
DESIGN AND ANALYSIS OF ALGORITHMS

Lecture 3: Probabilistic and Recursive Analysis

Lecturer: Dr. Yang Lu

Email: luyang@xmu.edu.my

Office: A1-432

Office hour: 2pm-4pm Mon & Thur

Analysis Method

Analyze an algorithm

- Best case
- Worst case
- Average case
  - Probabilistic analysis for randomized algorithm.
  - Recursive analysis for recursive algorithm.

PROBABILISTIC ANALYSIS

The Hiring Problem

Scenario:

- You are using an employment agency to hire a new office assistant.
- The agency sends you *one candidate each day*.
- You interview the candidate and must *immediately* decide whether or not to hire that person.
- If you hire, you must *immediately* fire your current office assistant.
- Cost to hire is c per candidate (includes cost to fire a current office assistant + hiring fee paid to agency).

The Hiring Problem

- You are committed to having hired, at all times, the best candidate seen so far.
  - You must fire the current office assistant and hire the candidate, if the candidate is better than the current office assistant.
  - Since you must have someone hired at all times, you will always hire the first candidate that you interview.
- Goal: Determine total hiring cost if there are n candidates.

Pseudocode

- If we hire m of n candidates finally, the cost will be cm,
- However, m varies with each run.
  - It depends on the order in which we interview the candidates.

```
int hire_assistant(int n,
                   const int c,
                   const int S[])
{
    int best = 0;
    int cost = c; // candidate 0 is a must-hire dummy candidate
    index i;
    for (i=1; i<=n; i++){
        if (S[i] > best){ // hire candidate i
            cost += c;
            best = S[i]
        }
    }
    return cost
}
```

Analysis of the Hiring Problem

- Best case
  - We just hire *one* candidate only.
    - The first is the best. Good luck thanks god.
  - Cost: $B(n) = c \in O(1)$.
- Worst case
  - We hire all n candidates.
    - Each candidate is better than the current hired one. What a tough life!
  - Cost: $W(n) = cn \in O(n)$.
- What is the average case $A(n)$?
  - Randomized algorithms.
  - Probabilistic analysis.

Probabilistic Analysis

- In general, we have no control over the order in which candidates appear.
- Assume that they *come in a random order*.
  - The interview score list S is equivalent to a permutation of the candidate numbers $< 1, 2, \ldots, n >$.
  - S is equally likely to be any one of the $n!$ permutations. We call this a ***uniform random permutation***.
    - Each of the possible $n!$ permutations appears with equal probability.

Probabilistic Analysis

- Basic idea:
  - Determine the distribution of inputs.
  - Analyze the algorithm and compute an expected cost.
  - The expectation is taken over the distribution of the possible inputs.
- Thus, averaging the cost over all possible inputs based on the input distribution.
- Unfortunately, for some problems, we cannot describe a reasonable input distribution, and in these cases we cannot use probabilistic analysis.

Distribution of Inputs

In many cases, we know very little about the distribution of inputs.

- In the hiring problem, it seems that the candidates are in a random order, but we have no way of knowing whether or not they really are.
  - Maybe the agent has roughly sorted them in an unknown order?
- In the sequential search problem, what if the key x is always one of the smallest numbers in the array S?
- In the phone book problem, what if we just frequently call the friends whose surname starts with "A"?

Randomized Algorithms

- We are not interested in how the inputs distribute. We are interested in how the algorithm performs.
- Thus, in order to analyze the average case of the hiring algorithm, we must have greater control over the order in which we interview the candidates.
- Under this scenario, we need randomized algorithms.
  - Make randomization within the algorithm, but not rely on the input distribution.

Review the Scenario of Hiring Problem

We change the scenario:

- The employment agency sends us a list of all n candidates in advance.
- On each day, we randomly choose a candidate from the list to interview (but considering only those we have not yet interviewed).
- Instead of relying on the candidates being presented to us in a random order, we take control of the process and *enforce a random order*.
  - Thus, we know that now the candidate order is truly from uniform random permutation.

Randomized Algorithms

- In general, we call an algorithm *randomized* if its behavior is determined not only by input but also by values produced by a *random-number generator*.
- Random-number generator
  - $\mathrm{RANDOM}(a, b)$ returns an integer r , where $a \leq r \leq b$ and each of the $b - a + 1$ possible values of r is equally likely.
  - In practice, RANDOM is implemented by a *pseudorandom-number generator*, which is a deterministic method returning numbers that "look" random and pass statistical tests.
    - e.g., random.random(), random.randint(a, b) in Python.

Indicator Random Variables

- We introduce **Indicator Random Variables**, which is a simple yet powerful technique for computing the expected value of a random variable.
- Given a sample space and an event A, we define the indicator random variable:

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs,} \\ 0 & \text{if } A \text{ does not occur.} \end{cases}$$

Indicator Random Variables

Lemma

For an event A, let $X_A = I\{A\}$. Then $E[X_A] = \Pr\{A\}$.

Proof:

By the definition of an indicator random variable and the definition of expected value, we have:

$$\begin{aligned} E[X_A] &= E[I\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\} \\ &= \Pr\{A\} \end{aligned}$$

Indicator Random Variables

Example 1

Determine the expected number of heads when we flip a coin for n times. The probability of flipping a head is 0.6.

- Denoting event H_i as flipping a head at the ith flip, we have $\Pr[H_i] = 0.6$.
- Let X be a random variable for the number of heads in n flips. We thus calculate $E[X]$.
- Define indicator random variables $X_i = I\{H_i\}$, for $i = 1, 2, \ldots, n$. Hence, $X = \sum_{i=1}^{n} X_i$.
- Lemma says that $E[X_i] = \Pr\{H_i\} = 0.6$ for $i = 1, 2, \ldots, n$. So, we finally have

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \Pr[H] = 0.6n.$$

Analysis of the Hiring Problem Using Indicator Random Variables

- Assume that the candidates arrive in a random order.
- Denote the event that candidate i is hired as H_i.
- Let X be a random variable that equals the number of times we hire a new office assistant.
- Define indicator random variables $X_i = I\{H_i\}$, for $i = 1, 2, \ldots, n$.
- Then we have $X = \sum_{i=1}^{n} X_i$.

Analysis of the Hiring Problem Using Indicator Random Variables

- By the lemma, we know that $E[X_i] = \Pr\{H_i\}$. We need to compute $\Pr\{H_i\}$.
- Candidate i is hired if and only if candidate i is better than each of candidates $1, 2, \ldots, i-1$.
- Assumption that the candidates arrive in random order => candidates $1, 2, \ldots, i$ arrive in random order => any one of these first i candidates is equally likely to be the best one so far.
- Thus, $\Pr\{H_i\} = 1/i$.

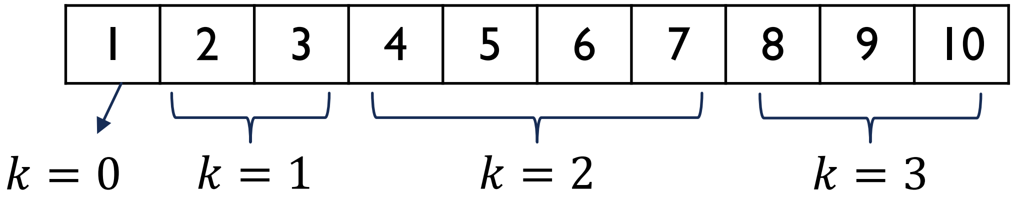
Analysis of the Hiring Problem Using Indicator Random Variables

$$
\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} E[X_i] \\
&= \sum_{i=1}^{n} \Pr\{H_i\} \\
&= \sum_{i=1}^{n} \frac{1}{i} \\
&\le \lg n + 1 \in O(\lg n)
\end{aligned}
$$

Thus, the expected hiring cost is $O(\lg n)$, which is much better than the worst case cost of $O(n)$.

$\sum_{i=1}^{n} \frac{1}{i}$ is called the nth harmonic number. It has a bound of $O(\lg n)$:

$$
\begin{aligned}
\sum_{i=1}^{n} \frac{1}{i} &\le \sum_{k=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^k-1} \frac{1}{2^k + j} \\
&\le \sum_{k=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^k-1} \frac{1}{2^k} \\
&= \sum_{k=0}^{\lfloor \lg n \rfloor} 1 \\
&\le \lg n + 1\,.
\end{aligned}
$$



Randomized Algorithms for the Hiring Problem

- The randomization is now in the algorithm, not in the input distribution.
- Given a particular input, we can no longer say what its hiring cost will be. Each time we run the algorithm, we can get a different hiring cost.
  - The execution depends on the random choices made.
- **No** particular input always elicits worst-case or best-case behavior.
- Bad behavior occurs only if we get "unlucky" numbers from the random number generator.

Pseudocode for Randomized Hiring Problem

```
int randomize_permute(int n,
                      int S[])
{
    index i;
    for (i=1; i<=n; i++){
        exchange S[i] and S[random(i, n)];
    }
}

int randomized_hire_assistant(int n,
                              const int c,
                              const int S[])
{
    randomize_permute(n, S);
    return hire_assistant(n, c, S);
}
```

random integer in [i, n]

XIAMEN UNIVERSITY MALAYSIA 厦門大學馬來西亞分校

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY

厦门大学计算机科学系 Computer Science Department of Xiamen University

Process of Probabilistic Analysis

1. Check whether the algorithm is deterministic or randomized. If it is deterministic, modify it to randomized version.
2. Identify the random variable X which makes $E[X]$ the result that we want to calculate.
3. Identify the event H and its probability $\Pr\{H\}$.
4. Define indicator random variables $X_i = I\{H_i\}$, for $i = 1, 2, \ldots, n$.
5. Identify the relation between X and each indicator random variable X_i.
6. Use the lemma $E[X_i] = \Pr\{H_i\}$ and derive $E[X]$.

Examples of Probabilistic Analysis

Example 2: the Hat-Check Problem

- Each of n customers gives a hat to a hat-check person at a restaurant.
- The hat-check person gives the hats back to the customers in a random order.
- What is the expected number of customers that get back their own hat?

Examples of Probabilistic Analysis

Example 2 (cont'd)

- Let X be a random variable of the number of customers that get back their own hat, so that we want to compute $E[X]$.
- Denote the event that customer i gets back his own hat as H_i.
- Because there are n hats and the ordering of hats is random, each customer has a probability of $1/n$ of getting back his or her own hat. So we have $\Pr\{H_i\} = 1/n$.
- Define indicator random variables $X_i = I\{H_i\}$, for $i = 1, 2, \ldots, n$. We have $X = \sum_{i=1}^{n} X_i$.
- Now we can compute $E[X]$ by using the lemma:

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \Pr\{H_i\} = \sum_{i=1}^{n} \frac{1}{n} = 1.$$

Examples of Probabilistic Analysis

Example 3

- Assume that 12 passengers enter an elevator at the basement and independently choose to exit randomly at one of the 10 above-ground floors.
- What is the expected number of stops that the elevator will have to make?

Examples of Probabilistic Analysis

Example 3 (cont'd)

- Let X be a random variable of the number of stops that the elevator will have to make, so that we want to compute $E[X]$.
- Denote the event that the elevator stops at the ith level as H_i.
- $\Pr\{H_i\} = 1 - \Pr\{\overline{H_i}\} = 1 - (1 - 1/10)^{12} = 1 - (9/10)^{12}$.
  - $\overline{H_i}$: the elevator does not stop (no passenger exit) at the ith level.
- Define indicator random variables $X_i = I\{H_i\}$, for $i = 1, 2, \ldots, 10$. We have $X = \sum_{i=1}^{10} X_i$.
- Now we can compute $E[X]$ by using the lemma:

$$E[X] = E\left[\sum_{i=1}^{10} X_i\right] = \sum_{i=1}^{10} E[X_i] = \sum_{i=1}^{10} \Pr\{H_i\} = \sum_{i=1}^{10} (1 - 0.9^{12}) = 10(1 - 0.9^{12}) \approx 7.176.$$

Examples of Probabilistic Analysis

Example 4

- Let $A[1 \dots n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A.
- Suppose that each element of A is generated by randomly permutation. What is the expected number of inversions.

Examples of Probabilistic Analysis

Example 4 (cont'd)

- Let X be a random variable of the total number of inverted pairs in A, so that we want to compute $E[X]$.
- Denote the event $i < j$ and $A[i] > A[j]$ as H_{ij}.
- Given two distinct random numbers, the probability that the first is bigger than the second is $1/2$. We have $\Pr\{H_{ij}\} = 1/2$.
- Define indicator random variables $X_{ij} = I\{H_{ij}\}$, for $1 \le i < j \le n$. We have $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$.
- Now we can compute $E[X]$ by using the lemma:

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} = \frac{n(n-1)}{2} \cdot \frac{1}{2} = \frac{n(n-1)}{4}.$$

number of pairs in the array

RECURSIVE ANALYSIS

Recursive Analysis

- Fibonacci sequence is defined by

$$
\begin{aligned}
f_0 &= 0 \\
f_1 &= 1 \\
f_n &= f_{n-1} - f_{n-2}, \qquad \text{for } n \geq 2
\end{aligned}
$$

- 0, 1, 1, 2, 3, 5, 8, 13, 21….

```
int fib(int n)
{
    if (n <= 1)
        return n;
    else
        return fib(n-1) + fib(n-2);
}
```

Recursive algorithm to calculate the nth Fibonacci term

Recursive Equation

- For a recursive algorithm, its every-case time complexity $T(n)$ can be written as a recursive equation.
- For example, the recursive equation for calculating the nth Fibonacci term:

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1, \\ T(n-1) + T(n-2) + 1 & \text{if } n > 1. \end{cases}$$

- **Goal of recursion analysis**: obtain an asymptotic bound Θ or O from the the recursive equation of a recursive algorithm.

Overview of Recursive Analysis Methods

- Substitution method
  - Guess a bound;
  - Prove our guess correct using Mathematical Induction.
- Recursion-tree method
  - Convert the recursion into a tree;
  - Best used to generate a good guess.
- Master method
  - A theorem with three cases;
  - In each case, the result can be directly obtained without calculation.

Technicalities

In practice, we *neglect certain technical details* when we state and solve recursion. It won't affect the final asymptotic results.

- Suppose n is an integer in $T(n)$.
- Omit floors and ceiling.
  - E.g. $T(n) = 2T(\lceil n/2 \rceil)$, and $T(n) = 2T(\lfloor n/2 \rfloor)$ are equivalent to $T(n) = 2T(n/2)$.
- As n is sufficiently small, we regard $T(n) = T(1)$, where $T(1)$ denotes the constant.
  - We can simply set $T(1) = 1$.

Substitution Method

1. Guess the form of the solution.
2. Use *mathematical induction* to find the constants and show that the solution works.

Substitution Method

Example 5

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

1. Guess $T(n) \in O(\lg n)$.
2. Prove: $T(n) \leq cn\lg n$:
   - **Basis**: When $n = 2, T(2) = 2T(1) + 2 = 4 \leq c2\lg 2$, for choosing $c = 2$.
   - **Inductive step**: Suppose $T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor)$.

$$\begin{aligned} T(n) &\leq 2\,c(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) + n \\ &\leq cn\lg(n/2) + n \\ &= cn\lg n - cn\lg 2 + n \\ &= cn\lg n - cn + n \\ &\leq cn\lg n \;\; (\text{for } c \geq 1) \end{aligned}$$

We usually don't need to set $n = 1$ for the induction basis because it sometimes doesn't work (e.g. can't prove $T(1) = 1 \leq c1\lg 1 = 0$). The asymptotic analysis only requires us to prove for $n \geq N$. It is ok to set $n = 2$ or $n = 3$ at basis step.

Substitution Method

Example 6

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

1. Guess $T(n) \in O(n)$.
2. Prove: $T(n) \leq cn$:
   - **Basis**: When $n = 1, T(1) = 1 \leq c$, for choosing any $c \geq 1$.
   - **Inductive step**: Suppose $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor$ and $T(\lceil n/2 \rceil) \leq c\lceil n/2 \rceil$.

   $$T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 = cn + 1$$

   - $T(n) \leq cn + 1$ can't imply $T(n) \leq cn$. How can we do?
     (loose) (tight)

Substitution Method

- Sometimes the guess is correct, but somehow the math doesn't seem to work out in the induction.
- Usually, the problem is that the inductive *assumption isn't strong enough* to prove the detailed bound.
- Revise the guess by *subtracting a lower-order term* often permits the math to go through.

Substitution Method

Example 6 (again)

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

1. Guess $T(n) \in O(n)$
2. Prove: $T(n) \leq cn - b$:
   - **Basis**: When $n = 1, T(1) = 1 \leq c - b$, for choosing any $c \geq 1 + b$.
   - **Inductive step**: Suppose $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor - b$ and $T(\lceil n/2 \rceil) \leq c\lceil n/2 \rceil - b$.

$$\begin{aligned} T(n) &\leq c\lfloor n/2 \rfloor - b + c\lceil n/2 \rceil - b + 1 \\ &= cn - 2b + 1 \\ &\leq cn - b \text{ (for } b \geq 1) \end{aligned}$$

   - $T(n) \leq cn - b$ can derive $T(n) \leq cn$. Therefore $T(n) \in O(n)$ is proved.

Substitution Method

Example 7

$$T(n) = 8T(n/2) + 5n^2$$

1. Guess $T(n) \in O(n^3)$.
2. Prove: $T(n) \leq cn^3$:
   - **Basis**: When $n = 1, T(1) = 1 \leq c$, for choosing any $c \geq 1$.
   - **Inductive step**: Suppose $T(n/2) \leq c(n/2)^3$.

$$\begin{aligned} T(n) &\leq 8c(n/2)^3 + 5n^2 \\ &= cn^3 + 5n^2 \end{aligned}$$

   - $T(n) \leq cn^3 + 5n^2$ can't prove $T(n) \leq cn^3$. We should subtract a lower-order term.

Substitution Method

Example 7 (cont'd)

$$T(n) = 8T(n/2) + 5n^2$$

1. Guess $T(n) \in O(n^3)$.
2. Prove: $T(n) \le cn^3 - bn^2$:
   - **Basis**: When $n = 1, T(1) = 1 \le c - b$, for choosing any $c \ge 1 + b$.
   - **Inductive step**: Suppose $T(n/2) \le c(n/2)^3 - b(n/2)^2$.

$$\begin{aligned} T(n) &\le 8[c(n/2)^3 - b(n/2)^2] + 5n^2 \\ &= cn^3 - 2bn^2 + 5n^2 \\ &= cn^3 - bn^2 - bn^2 + 5n^2 \\ &\le cn^3 - bn^2 \text{ (for } b \ge 5) \end{aligned}$$

   - $T(n) \le cn^3 - bn^2$ can derive $T(n) \le cn^3$. Therefore $T(n) \in O(n^3)$ is proved.

Changing Variables

Sometimes, a little algebraic manipulation can make an unknown recursion similar to one you have seen before.

Example 8

$$T(n) = 2T\left(\lfloor\sqrt{n}\rfloor\right) + \lg n$$

- Renaming $m = \lg n$ yields $n = 2^m$ and:

$$T(2^m) = 2T\left(2^{m/2}\right) + m.$$

- We can now rename $S(m) = T(2^m)$ to produce the new recursion:

$$S(m) = 2S(m/2) + m,$$

which has a solution of $S(m) \in O(m \lg m)$. Changing back from $S(m)$ to $T(n)$, we obtain:

$$T(n) = T(2^m) = S(m) \in O(m \lg m) = O(\lg n \lg \lg n).$$

Substitution Method

How to make a good guess:

- Bad News:
  - No general way to guess the correct solutions to recursion.
  - Good guess = E (experience) + C (creativity) + L (luck).
- Good News:
  - Recursion tree often generates good guesses.

Recursion Tree

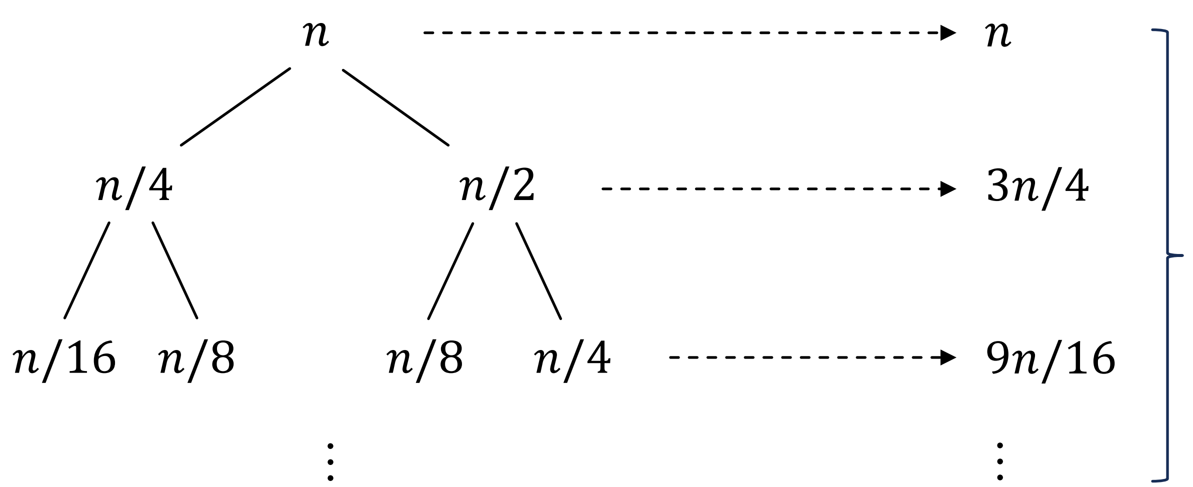
Each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.

1. We sum all the *per-node costs* within each level of the tree to obtain a set of *per-level costs*;
2. We sum all the *per-level costs* to determine the total cost of all levels of the recursion.

Recursion Tree

Example 9

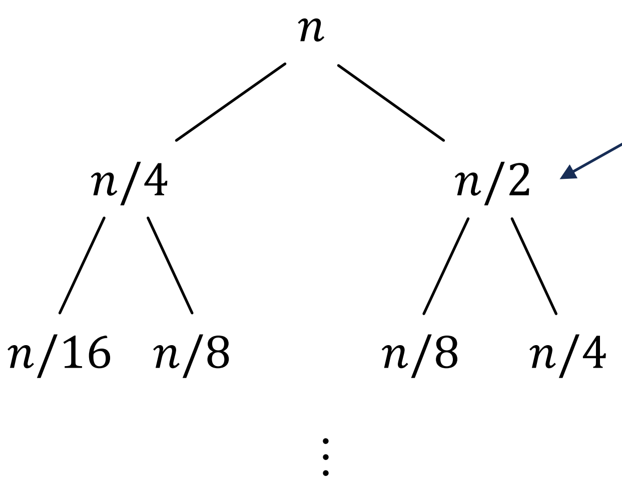
$$T(n) = T(n/4) + T(n/2) + n$$



What is the height of the tree?

Height of Recursion Tree

Example 9 (cont'd)



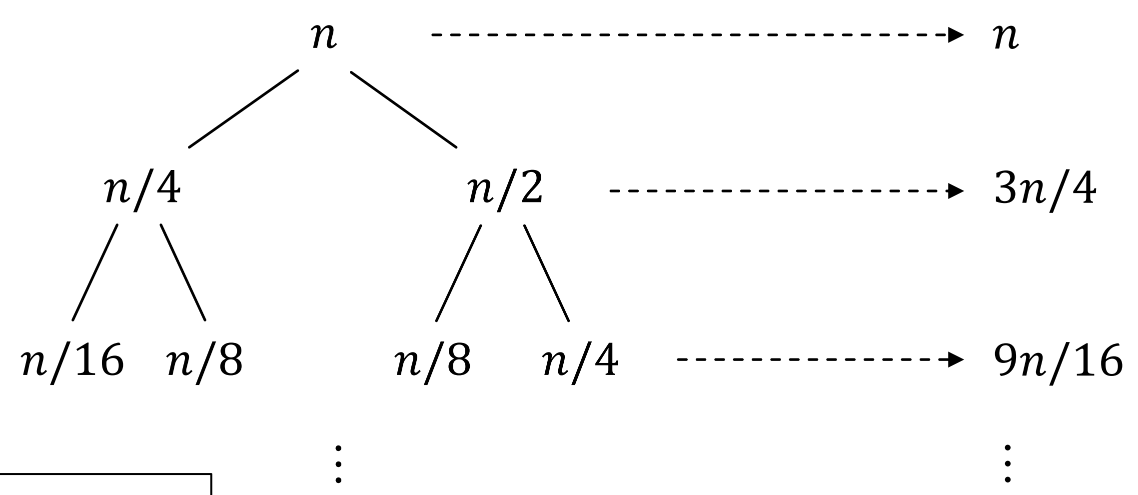
The decreasing rate of $n/2$ is slower than $n/4$

1. Determine the slowest deceasing rate.
2. Denote height of the recursion tree as k.
3. The node at the leaf of the tree is 1. Therefore $\frac{n}{2^k} = 1$ and $k = \lg n$.

Recursion Tree

Example 9 (cont'd)

$$
\begin{aligned}
T(n) &= n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \cdots + \left(\frac{3}{4}\right)^{\lg n} n \\
&= 4n\left[1 - \left(\frac{3}{4}\right)^{\lg n + 1}\right] \\
&\le 4n \in O(n)
\end{aligned}
$$



Sum of the first N terms of a *geometric sequence*

$\boxed{\frac{a_1(1-r^N)}{1-r}, \text{ where } a_1 = 1, r = 3/4, N = \lg n + 1}$

Recursion Tree

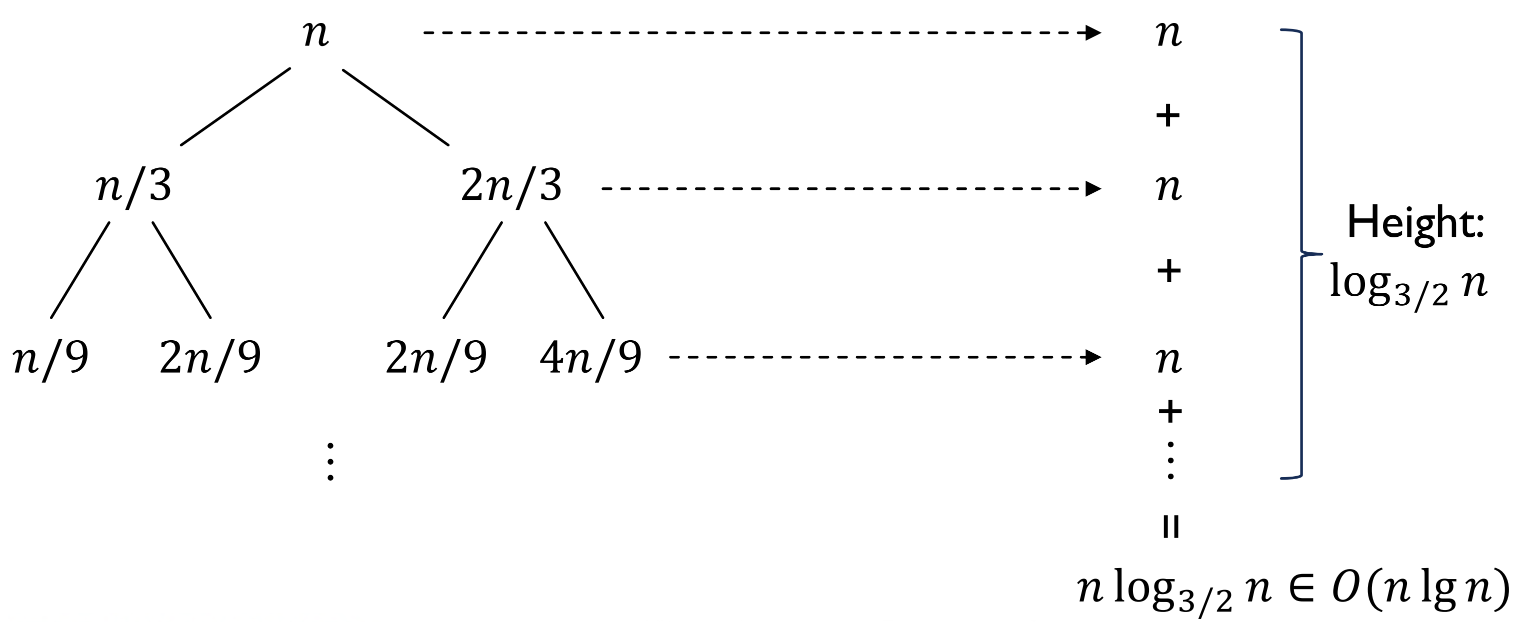
Example 10

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

Calculate height:

$$\left(\frac{2}{3}\right)^k n = 1$$

$$k = \log_{3/2} n$$



Master Method

The 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 recursion

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

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically with three cases:

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

Master Method

What does the master theorem mean?

- In each of the three cases, we are comparing $f(n)$ with $n^{\log_b a}$.
- Intuitively, the solution to the recursion is determined by the order of the two functions.
  - If, as in case 1, $n^{\log_b a}$ has high order, then the solution is $T(n) \in \Theta(n^{\log_b a})$.
  - If, as in case 2, the two functions are the same order, we multiply by a logarithmic factor, and the solution is $T(n) \in \Theta(n^{\log_b a} \lg n)$.
  - If, as in case 3, $f(n)$ has high order, then the solution is $T(n) \in \Theta(f(n))$.

Master Method

Take a deeper look of the master theorem. Beyond this intuition of comparing order of functions, there are some technicalities that must be understood.

- In case 1, not only must $f(n)$ have lower order than $n^{\log_b a}$, its order must be *polynomially lower*.
  - The order of $f(n)$ must be asymptotically lower than $n^{\log_b a}$ by a factor of n^{ϵ} for some constant $\epsilon > 0$.
- In case 3, not only must $f(n)$ have higher order than $n^{\log_b a}$, its order must be *polynomially higher*, and in addition *satisfy the "regularity" condition* that $af(n/b) \leq cf(n)$.
  - The order of $f(n)$ must be asymptotically higher than $n^{\log_b a}$ by a factor of n^{ϵ} for some constant $\epsilon > 0$.

Master Method

- The three cases do not cover all the possibilities for $T(n)$.
- There is a gap between cases 1 and 2 when the order of $f(n)$ is lower than $n^{\log_b a}$ but not polynomially lower.
- Similarly, there is a gap between cases 2 and 3 when the order of $f(n)$ is higher than $n^{\log_b a}$ but not polynomially higher.
- If the function $f(n)$ falls into one of these gaps, or if the regularity condition in case 3 fails to hold, the master method cannot be used to solve the recursion.

Master Method

Example 11

$$T(n) = 9T(n/3) + n$$

- We have $a = 9, b = 3, f(n) = n$, and thus we have $n^{\log_b a} = n^{\log_3 9} = n^2$.
- We thus compare n and n^2.
- Since $f(n) = n \in O(n^{\log_3 9 - \epsilon})$ for $\epsilon = 1$, we can apply case 1 of the master theorem and conclude that the solution is $T(n) \in \Theta\left(n^{\log_b a}\right) = \Theta(n^2)$.

Master Method

Example 12

$$T(n) = T(2n/3) + 1$$

- We have $a = 1, b = 3/2, f(n) = 1$, and thus we have $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$.
- We thus compare 1 and 1.
- Since $f(n) = 1 \in \Theta(1)$, and thus the solution to the recursion is $T(n) \in \Theta(\lg n)$.

Master Method

Example 13

$$T(n) = 3T(n/4) + n \lg n$$

- We have $a = 3, b = 4, f(n) = n \lg n$, and thus we have $n^{\log_b a} = n^{\log_4 3} \approx n^{0.793}$.
- We thus compare $n \lg n$ and $n^{\log_4 3}$.
- Since $f(n) = n \lg n \in \Omega(n) = \Omega(n^{\log_4 3+\epsilon})$ for $\epsilon \approx 0.2$, case 3 applies if we can show that the regularity condition holds for $f(n)$.
- For sufficiently large n, $af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)n \lg n = cf(n)$ for $c = 3/4$.
- Consequently, by case 3, the solution to the recursion is $T(n) \in \Theta(n \lg n)$.

Master Method

The master method does not apply to the recursion in the following example.

Example 14

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

- Even though it has the proper form: $a = 2, b = 2, f(n) = n \lg n$, and $n^{\log_b a} = n$.
- We thus compare $n \lg n$ and n.
- It might seem that case 3 should apply, since the order of $f(n) = n \lg n$ is asymptotically higher than n. The problem is that it is *not polynomially higher*.
- We can't find a constant $\epsilon > 0$ such that $f(n) = n \lg n \in \Omega(n^{1+\epsilon}) = \Omega(n \cdot n^{\epsilon})$

Try to compare the order between $\lg n$ and n^{ϵ}

Conclusion

After this lecture, you should know:

- What is a randomized algorithm.
- How to use probabilistic analysis to analyze the average case of an algorithm.
- What is a recursive equation.
- How to draw a recursive tree.
- How to derive the asymptotic result from the recursive equation (three methods).

Assignment 1

- Assignment 1 is released. The deadline is **18:00, 4th May**.

Thank you!

Reference:

- Chapter 4&5, Thomas H. Cormen, Introduction to Algorithms, Second Edition.

Acknowledgement: Thankfully acknowledge slide contents shared by Prof. Yiu-ming Cheung