CSCI3160 Design and Analysis of Algorithms (2025 Fall)

Basic Techniques: Recursion, Repeating, and Geometric Series

Instructor: Xiao Liang[1]

Department of Computer Science and Engineering
Chinese University of Hong Kong

[1]These slides are primarily based on materials prepared by Prof. Yufei Tao (please refer to Prof. Tao's version from 2024 Fall for the original content). Some modifications have been made to better align with this year's teaching progress, incorporating student feedback, in-class interactions, and my own teaching style and research perspective.

Today we will discuss three basic techniques of algorithm design:

- Recursion
- Geometric Series.
- Repeating (till success)

We will use two famous problems to desmonstrate the ideas behind these techniques:

- Hanoi Tower: for recursion
- k-selection: for geometric series and repeating till success.

Recursion

Principle of recursion

When dealing with a subproblem (same problem but with a smaller input), consider it solved, and use the subproblem's output to continue the algorithm design.

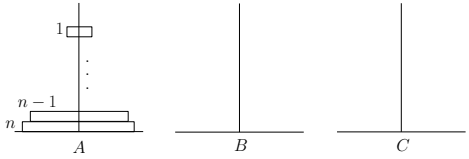
The Hanoi Tower Problem

Hanoi Tower

There are 3 rods A, B, and C.

On rod A, n disks of different sizes are stacked in such a way that no disk of a larger size is above a disk of a smaller size.

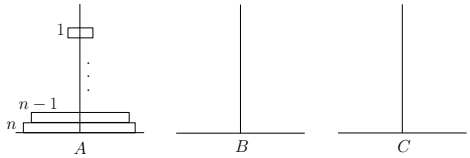
The other two rods are empty.



The Hanoi Tower Problem

Hanoi Tower

Permitted operation: Move the top-most disk of a rod to another rod.
Constraint: No disk of a larger size can be above a disk of a smaller size.



Goal: Design an algorithm to move all the disks to rod B.

On the Difficulty of the Hanoi Tower Problem

After some initial attempts, you will realize how hard this problem is.

Seems it is very challenging to design an *efficient* algorithm for that.

In this case, an important way of thinking in algorihtm design (and complexity theory) is to ask:

- Is there any *inherent* difficulty in solving the concerned problem?

Such a difficulty, if exists, is usually referred to as a "lower bound", "infeasibility", or "impossibility" result. It is an important branch of research for both algorithm design and computational complexity.
Some remarks:

- This question is so natural. But people usually forget to ask when they really get stuck with some hard problems.
- Don't insist on proving impossibility results *unless you have some intuition to start with*.

Intuition on the Difficulty

Our intuition on the inherent difficulty of the Hanoi Tower Problem:

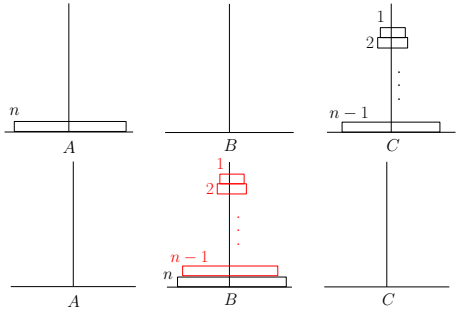
- Due to the constraint, we have to move the last disk to rob B, before we do that for other disks.
- However, to move the last disk, we have to move all other disks to rob C so we can start to move the last disk
- This problem exhibits a flavor of symmetry among the robs. And if we ignore the last disk, it is a new problem of $n-1$ disks. So, the last item essentially asks to solve the exactly same problem but with one less disk.

Importantly, the above intuition is *general*, i.e., it always applies no matter what algorithm you want to design.

Illustrating the Intuition

Subproblem: Same problem but with $n-1$ disks.
Consider the subproblem solved (i.e., assume you already have an algorithm for it).

Now, solve the problem with n disks as follows:



Formalizing our Intuition

Suppose that our algorithm performs $f(n)$ operations to solve a problem of size n. Clearly, $f(1) = 1$. By recursion, we can write

$$f(n) \quad \geq \quad 1 + 2 \cdot f(n-1)$$

Solving this recurrence gives $f(n) \geq 2^n - 1$.

Conclusion: The best time complexity (even for randomized algorithms) for solving the Tower of Hanoi problem with n disks is $\Omega(2^n)$.

Geometric Series and Repeating till Success

The k-Selection Problem

The k-Selection Problem: You are given a set S of n integers in an (possibly unsorted) array and an integer $k \in [1, n]$. Find the k-th smallest integer of S.

For example, suppose that $S = (53, 92, 85, 23, 35, 12, 68, 74)$ and $k = 3$. You should output 35.

Definition of Rank

We will introduce a definition to assist our subsequent discussion:

> The **rank** of an integer $v \in S$ is the number of elements in S smaller than or equal to v.

For example, suppose that $S = (53, 92, 85, 23, 35, 12, 68, 74)$. Then, the rank of 53 is 4, and that of 12 is 1.

Easy: The rank of v can be obtained in $O(|S|)$ time.

Consider the following task:

Task: Assume n to be a multiple of 3. Obtain a subproblem of size at most $2n/3$ with exactly the same result as the original problem.

Our goal is to produce a set S' and an integer k' such that

- $|S'| \leq 2n/3$
- $k' \in [1, |S'|]$
- The element with rank k' in S' is the element with rank k in S.

Note: it is possible that $k' \neq k$.

We will give an algorithm to accomplish the task in $O(n)$ expected time.

Next, we will focus on the following two claims:

1. There exists a (randomized) algorithm A_{sub} that accomplish the previous task in $O(n)$ expected time. (This step utilizes Repeating till Success.)
2. We can utilize A_{sub} to design a (randomized) algorithms that solves k-selection in $O(n)$ expected time. (This step utilizes Geometric Series.)

We will first see how the second item works. After that, we will present the algorithm A_{sub}.

Utilizing A_{sub}

- $A_{sub}(S, k) \to (S_1, k_1)$. Note that $|S_1| = \frac{2}{3} \cdot n$.
- $A_{sub}(S_1, k_1) \to (S_2, k_2)$. Note that $|S_2| = (\frac{2}{3})^2 \cdot n$.
- $A_{sub}(S_2, k_2) \to (S_3, k_3)$. Note that $|S_3| = (\frac{2}{3})^3 \cdot n$.
- ...

Stop until the t-th repetition such that $|S_t| = 1$, i.e.,

$$(\frac{2}{3})^t = \frac{1}{n}.$$

You can of course solve for t from the above equation to calculate the running time of the algorithm. But we will do it in an alternative way, utilizing the Geometric Series.

Geometric Series

A **geometric sequence** is an infinite sequence of the form

$$n, cn, c^2 n, c^3 n, \ldots$$

where n is a positive number and c is a constant satisfying $0 < c < 1$.

Also recall the formula for the sum of finite geometric series:

$$S_t = n \cdot \frac{1 - c^t}{1 - c}$$

It holds in general that

$$\sum_{t=0}^{\infty} c^t n = \lim_{t \to \infty} S_t = \lim_{t \to \infty} n \cdot \frac{1 - c^t}{1 - c} = \frac{n}{1 - c} = O(n).$$

The summation $\sum_{t=0}^{\infty} c^t n$ is called a **geometric series**.

Geometric series are extremely important for algorithm design.

Algorithm

Using the repeating technique, now you should be able to convert the problem to a subproblem with size at most $\lceil 2n/3 \rceil$ in $O(n)$ expected time. (Note: we have not shown that A_{sub} runs in in $O(n)$ expected time yet. We will do that toward the end.)

Now, apply the recursion technique. We have already obtained a (complete) algorithm solving the k-selection problem!

Think: How is this related to geometric series?

Running Time Analysis via Geometric Series

Assume: the running time of A_{sub} is linear, i.e., $O(n)$. (We will prove this later.)

The expected running time of our eventual algorithm (which invokes A_{sub} until $|S_t| = 1$) will be (note that we also use the linearity of the expectation operator here)

$$a \cdot n + a \cdot \frac{2}{3} \cdot n + a \cdot (\frac{2}{3})^2 \cdot n + \ldots + a \cdot (\frac{2}{3})^t \cdot n$$

where a is the constant hidden in the big-O notation (here we need to make it explicit). This is no larger than

$$\begin{aligned} & a \cdot n + a \cdot \frac{2}{3} \cdot n + a \cdot (\frac{2}{3})^2 \cdot n + \ldots + \ldots \\ = \; & a \cdot n + a \cdot \sum_{i=1}^{\infty} (\frac{2}{3})^i \cdot n \\ = \; & a \cdot n + a \cdot O(n) \qquad \text{(by Geometric Series)} \\ = \; & O(n) \end{aligned}$$

Repeat till Success

Algorithm for the Subproblem

The algorithm A_{sub}:

1. Take an element $v \in S$ uniformly at random.
2. Divide S into S_1 and S_2 where
   - S_1 = the set of elements in S less than or equal to v;
   - S_2 = the set of elements in S greater than v.
3. If $|S_1| \geq k$, then return $S' = S_1$ and $k' = k$;
   else return $S' = S_2$ and $k' = k - |S_1|$.

The algorithm **succeeds** if $|S'| \leq 2n/3$, or **fails** otherwise.

Repeat the algorithm until it succeeds.

Algorithm for the Subproblem

Lemma: The algorithm succeeds with probability at least $1/3$.

Proof: The algorithm always succeeds when the rank of v falls in $[\frac{n}{3}, \frac{2}{3}n]$ (think: why?). This happens with a probability at least $1/3$, by the fact that v is taken from S uniformly at random. □

In general, if an algorithm succeeds with a probability **at least** $c > 0$, then the number of repeats needed for the algorithm to succeed for the first time is **at most** $1/c$ in expectation.

Running time of A_{sub}

The algorithm A_{sub}:

1. Take an element $v \in S$ uniformly at random. ($O(1)$ in our model. Why?)
2. Divide S into S_1 and S_2 where
   - S_1 = the set of elements in S less than or equal to v;
   - S_2 = the set of elements in S greater than v.

   ($O(n)$ in our model.)
3. If $|S_1| \geq k$, then return $S' = S_1$ and $k' = k$;
   else return $S' = S_2$ and $k' = k - |S_1|$.
   ($O(n)$ in our model.)

Thus:

- Each execution cost $O(n)$ (deterministic) time.
- By the previous lemma, we need to repeat it for 3 times *in expectation* until it succeeds.
- This implies that the expected running time is $O(n)$ (think: why? - Linearity of expectation).

Closing Remark (1/2)

It may seem almost magical that we can suddenly define the algorithm A_{sub}, and it just happens to work so well. At first glance, it feels as though the solution appears out of thin air, without any clear path leading to its discovery.

In reality, the techniques we've introduced so far seems to primarily serve as tools for analysis, rather than *direct aids in constructing the algorithm itself*. They help us understand why the algorithm works, prove its correctness, and analyze its efficiency—but they don't necessarily guide us toward finding the algorithm in the first place.

Closing Remark (2/2)

This phenomenon is actually common in algorithm design. For many non-trivial algorithms, there is often a component that appears almost mysterious or unmotivated at first. These seemingly magical insights are not accidents — they are the product of scientific creativity, intuition, and experience.

In fact, this creative leap is what distinguishes routine problem-solving from true algorithmic innovation. The "magic" is often the result of deep understanding, abstraction, and pattern recognition developed over time. Once the core idea is discovered, the rest of the work—rigorous analysis, optimization, and formal proof—can follow in a more systematic way.

So while it may feel surprising that A_{sub} works so well, this is a reminder of a deeper truth: in algorithm design, insight often precedes explanation.