Algorithms Lecture Notes Week 1

TA: Zainab Usman

Runtime Analysis Philosophy

When studying algorithms, we measure how the number of operations grows when input size n becomes large.

The main goal is to understand growth behavior, not exact running time.

This idea leads to asymptotic analysis, which describes performance trends for large input.

Asymptotic Notations

Big O Upper Bound

$$f(n) = O(g(n))$$

Big O represents an upper bound on growth. It describes the worst case behavior. It guarantees that execution time will not grow faster than some constant multiple of $g(n)$ for large n.

Example: two nested loops usually produce $O(n^2)$ behavior.

Big Omega Lower Bound

$$f(n) = \Omega(g(n))$$

Big Omega provides a lower bound. It represents the best possible growth behavior that an algorithm can achieve.

Example: binary search best behavior is $\Omega(\log n)$.

Big Theta Tight Bound

$$f(n) = \Theta(g(n))$$

Big Theta means the function is bounded both above and below by constant multiples of $g(n)$ for large n.

Formally, there exist constants c_1, c_2, n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \quad \forall n \geq n_0.$$

Important intuition: Big Theta does not mean equality. It means the function is sandwiched between two scaled versions of $g(n)$. In class discussion, students noted that we often say "equals", but mathematically it means the same growth trend, not the same value.

Common Understanding

Big O describes worst behavior.

Big Theta describes tight growth behavior.

Big Omega describes best behavior.

Algorithms may not behave the same in all cases. Linear scan behaves similarly across cases, while randomized algorithms may vary widely.

The Selection Problem

Now that we know how to measure performance, we apply it to a real problem.

Given n distinct numbers and an integer k, find the k th smallest element.

If $k = 1$, the answer is the minimum.

If $k = n$, the answer is the maximum.

If $k = n/2$, the answer is the median.

Sorting solves this in $O(n \log n)$ time, but we can do better. Selection can be solved in linear time.

Deterministic Linear Time Selection

The Median of Medians algorithm guarantees worst case linear complexity.

The idea is to carefully choose a pivot so that recursion always shrinks fast.

Algorithm Procedure

1. Divide elements into groups of five.
2. Sort each group and extract its median.
3. Recursively find the median of those medians to obtain pivot p.
4. Partition the array around pivot p.
5. Recurse only on the side that contains the desired element.

Why the Algorithm Works

Instead of choosing a random pivot, we build a strong pivot. This guarantees that many elements are smaller and many are larger, preventing deep recursion.

Approximate Median Theory

A pivot M is called an approximate median if

At least n/c elements are smaller than M.

At least n/c elements are larger than M.

Here $c \geq 2$.

If $c = 2$, we obtain the true median.

Larger c gives weaker balance.

However, balance cannot be arbitrarily weak. To keep the algorithm linear, the recursive part must shrink enough so that total work decreases each step.

The mechanism specifically needs the recursive portion to stay below about $8n/10$. If c becomes too large, recursion may shrink too slowly and performance degrades.

Median of Medians guarantees about $3n/10$ elements on each side, which safely satisfies this condition.

- **Theoretical Minimum Split:** To maintain $O(n)$, we only strictly need M to be larger than $2n/10$ elements on each side. The algorithm provides about $3n/10$, giving a safety margin that keeps recursion shrinking fast.

Counting Argument

There are $n/5$ groups.

There are $n/5$ medians.

At least $n/10$ medians are less than or equal to pivot M.

Each contributes at least two smaller elements from its group.

So we obtain

$$\frac{n}{10} + 2 \cdot \frac{n}{10} = \frac{3n}{10}.$$

Thus many elements are removed in each recursion.

Convexity Concepts

To analyze the recursion mathematically, we use convexity.

A function f is convex if

$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2).$$

The term $\alpha x_1 + (1-\alpha)x_2$ is a weighted average. It always lies between x_1 and x_2 when $0 \leq \alpha \leq 1$.

- **Weighted Mean Logic:** The coordinate $\alpha x_1 + (1-\alpha)x_2$ is guaranteed to stay between the two chosen x values, allowing the secant line connecting them to remain above the curve for convex functions.

Visual intuition: imagine two points on a curve. Draw a straight line between them. For convex functions, the curve always stays below that straight connecting line, called the secant line. Tangents drawn at any point also remain below the curve.

- **The Tangent Check:** A local way to verify convexity is to ensure that any tangent line at a point lies below the function everywhere.

Example

Let $f(x) = e^x$.

For $x_1 = 0.5$, $x_2 = 1.5$, and $\alpha = 0.3$,

$$f(1.2) = e^{1.2},$$

$$0.3e^{0.5} + 0.7e^{1.5}.$$

The point on the curve is smaller than the point on the secant line, confirming convexity.

Useful Properties

Assume f is convex and $f(0) = 0$.

Then

$$f(\alpha a) \leq \alpha f(a),$$

$$f(x) + f(y) \leq f(x + y).$$

These allow us to combine recursive terms.

Solving the Recurrence

From the selection algorithm we get

$$T(n) \leq T(n/5) + T(7n/10) + cn.$$

Using subadditivity,

$$T(n/5) + T(7n/10) \leq T(9n/10).$$

So

$$T(n) \leq T(9n/10) + cn.$$

Unrolling gives

$$cn + c(0.9n) + c(0.9^2 n) + \cdots$$

This geometric series converges because the ratio is less than one.

Therefore

$$T(n) = O(n).$$

Recurrence Safety Condition

We need the total recursive size to stay smaller than the original input.

$$\frac{n}{5} + \text{largest recursive part} < n.$$

Since $n/5 = 2n/10$, the remaining part must stay below $8n/10$.

Median of Medians guarantees about $3n/10$, so recursion shrinks fast enough to ensure linear growth.

Mystery Algorithm

Consider

$$mystery(p) = 1 + mystery(\sqrt{p}).$$

This example is important because it shows that some algorithms grow extremely slowly.

After k steps,

$$p^{1/2^k} \leq 2.$$

Solving gives

$$k = O(\log \log p).$$

This demonstrates that asymptotic analysis focuses on behavior after some threshold value n_0. Before that point, constants dominate, but after it, growth trends matter most.