Algorithms, Design & Analysis

Lecture 02: **Finding Median**

Assad & Hamza

Information Technology University

June 19, 2025

About Your Fellows

- Hi there! We are **Assadullah Farrukh** and **M.Hamza Naveed**.
- We are Associate Students at ITU.

Recap

- Problem: Find the kth smallest element in a distinct array?

Recap

- **Problem:** Find the k^{th} smallest element in an array with distinct elements.
- **Algorithm 1:**
  - Find the smallest element in the array, remove it, and store it in a variable.
  - Repeat this process **k** times.
  - The value in the variable after the $\mathbf{k^{th}}$ iteration is the $\mathbf{k^{th}}$ smallest element.
- **Time Complexity:** $O(kN)$, where N is the size of the array.

Recap

- **Problem:** Find the k^{th} smallest element in an array with distinct elements.
- **Algorithm 2:**
  - Do Merge sort in the array
  - Return the element at the **(k-1)th** index of the array.
- **Time Complexity:** *O(Nlog(N))*, where *N* is the size of the array.

Recap

- **Problem:** Find the kth smallest element in an array with distinct elements.
- **Algorithm 3: Guess Select Algorithm (commonly known as the Randomized Select Algorithm)**
  - **Input:** Array A and k, where k is the position of the desired element.
  - Guess g as an element from A. Uniformly Randomly pick a number.
  - Partition A into:
    - L: Elements less than g.
    - R: Elements greater than g.
  - Compute the rank of g using L and R.
  - If $|L| = k - 1$, return g.
  - Else if $|L| > k$, recursively select k^{th} element from L.
  - Otherwise, recursively select $(k - |L| - 1)^{th}$ element from R.

Types of Algorithms

Randomized algorithms can be classified into two main types:

- **Monte Carlo Algorithm**
- **Las Vegas Algorithm**

Monte Carlo Algorithm

Characteristics

- Stops in a fixed amount of time but does not guarantee a result.
- The outcome is probabilistic, relying on randomness.

- **Fixed Runtime:** Executes within a pre-determined, fixed duration regardless of input size.
- **Randomized Outcome:** The result is a random variable and may not always be found.

Las Vegas Algorithm

Characteristics

- Always produces a correct result, but runtime is not fixed.
- The execution time depends on randomness and may vary.

- **Guaranteed Correctness:** The algorithm always delivers a correct result, regardless time taken.
- **Variable Runtime:** The runtime can fluctuate depending on the random choices made during execution.

Optimality of Las Vegas Algorithm

Which Algorithm is Optimal?

Oh, the **Las Vegas Algorithm** is optimal! But why? Let's find out.

- **Why is it Optimal?** Because we **do not stop until the job is done**. The algorithm keeps running until it produces a **correct result**.
- **Guaranteed Accuracy:** Since the algorithm doesn't terminate until it finds a valid solution, the outcome is always **correct and reliable**.

Algorithm Used in Randomized Select

Which Algorithm was used in yesterday's randomized select algorithm?

Algorithm Used in Randomized Select

Which Algorithm was used in yesterday's randomized select algorithm?

The **Las Vegas Algorithm** was used in yesterday's Randomized Select Algorithm!

- **Why Las Vegas Algorithm?** Because it **provides a guaranteed solution**, ensuring correctness in the result.
- **Reliability:** Unlike Monte Carlo algorithms, Las Vegas algorithms **never compromise on accuracy**, making them ideal for Randomized Select.

Worst-Case Time Complexity of Randomized Select

What is the Worst Case?

The worst-case scenario occurs when the algorithm repeatedly selects the **least optimal pivot**, leading to unbalanced partitions.

- **Worst-Case Time Complexity:** In the worst case, the time complexity of our Randomized Select becomes $\mathbf{O(n^2)}$. This happens when each partitioning step reduces the problem size by only one element.
- **Why Does This Happen?** The random pivot selection may consistently divide the array unevenly, causing the algorithm to behave like a naive selection algorithm.

Worst-Case Time Complexity Derivation

Recursive Relation

The recurrence relation for the worst-case time complexity of Randomized Select is:

$$\mathbf{T(n) = cn + T(n\text{-}1)}$$

where c is a constant representing the time taken for partitioning.

Base Case

$$\mathbf{T(1) = a}$$

where a is a constant representing the time for the smallest input size.

Worst-Case Time Complexity Derivation

Derivation

Expanding the recurrence:

$$T(n) = cn + c(n-1) + c(n-2) + \cdots + c(1) + a$$

Simplifies to:

$$T(n) = c\sum_{i=1}^{n} i + a = c\frac{n(n+1)}{2} + a$$

Resulting in the Worst-Case Complexity: $T(n) = O(n^2)$**.**

The worst case occurs when the pivot selection repeatedly divides unevenly causing the size to only go down by 1.

Professor's Question

Question:

The professor asked the class:

What is the average runtime of an Algorithm?

The class was quiet, thinking... until Wasif spoke up.

Wasif's Reply

Wasif's Response:

Wasif hesitantly replied:

Average Runtime:

$$\frac{\text{Best Case} + \text{Worst Case}}{2}$$

The professor smiled, but was it the correct answer?

Derivation of Average Runtime

Derivation Steps:

1. **Average Runtime:** $\frac{\text{Best Case}+\text{Worst Case}}{2}$
2. **Recurrence Relation for Best Case:** $T(n) = cn + T\left(\frac{n}{2}\right)$
3. **Expanding Recurrence:**

$$T(n) = cn + c\frac{n}{2} + c\frac{n}{4} + \cdots + 2$$

4. **Summing Series:**

$$T(n) = a + cn \sum_{i=1}^{k} \left(\frac{1}{2}\right)^i$$

5. **Simplifying:** $\leq a + 2cn = O(n)$

Derivation of Average Runtime

6. Combining Best and Worst:

$$\frac{O(n^2) + O(n)}{2} = O(n^2)$$

The derivation concludes that the average runtime is $O(n^2)$ which is the same as the worst case runtime so this can't be right Professor appreciated Wasif for his try moved on to explain what Average run time actually is by giving a hint that it is also called expected average runtime.

Expected Average Time Complexity

Expected Runtime Derivation:

1. **Average Runtime (Expected):**

$$\mathbb{E}(T(n)) = Cn + T(\mathbb{E}(L)) \quad \text{or} \quad \mathbb{E}(T(n)) = Cn + T(\mathbb{E}(R))$$

2. **Explanation:**
   - The expected time complexity is the time taken by using expectation so now the recursive relation is like this as the next recursion will be for the L or R which ever is now supposed to have the kth value.

Now all of our faces showed the Professor like he had lost us some where so...

Expectation: A Class Discussion

The Question

The professor asked the class:

"You do know what expectation and random variables are?"

But despite knowing the answer, no one could recall what even those words were!

Professor's Explanation

The professor began explaining the concept of expectation, step by step. The curiosity in the room grew!

Example: Coin Toss Experiment

Scenario

Consider the random variable X: the number of heads in the toss of two coins. Possible values are $0, 1, 2$.

Probability Distribution

$$\mathcal{P}(X = 0) = \frac{1}{4}, \quad \mathcal{P}(X = 1) = \frac{1}{2}, \quad \mathcal{P}(X = 2) = \frac{1}{4}$$

Expected Value Calculation

Calculation

Using the formula:

$$\mathbb{E}(X) = 0 \cdot \mathcal{P}(X = 0) + 1 \cdot \mathcal{P}(X = 1) + 2 \cdot \mathcal{P}(X = 2)$$

$$\mathbb{E}(X) = 0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} = 1$$

Conclusion

The expected number of heads is **1** in two coin tosses.

The Formula of Expectation

Definition of Expectation

The professor wrote the formula for expectation:

$$\mathbb{E}(X) = \sum_i x_i \cdot \mathcal{P}(x_i)$$

Class Reaction

More than half of the class suddenly exclaimed:

"Oh yeah, we know this!"

Back to Expected Average Time Calculation

- Recurrence Relation:

$$\mathbb{E}(T(n)) = Cn + T(\mathbb{E}(L)) \quad \text{or} \quad \mathbb{E}(T(n)) = Cn + T(\mathbb{E}(R))$$

- Assumption: L is always the smallest among the partitions L and R.

$$T(n) = cn + T(|L|)$$

- Sorted Array:

$$a_1, a_2, \ldots, a_n$$

- If k is the median, then g is a good guess only if the array A was sorted. In that case, the guess lies between $a_{n/4}$ and $a_{3n/4}$.

Back to Expected Average Time Calculation

- Guessing Method:

$$\mathcal{P}\left(g \in \left(a_{\frac{n}{4}}, a_{\frac{3n}{4}}\right)\right) = \frac{1}{2}$$

  If you are willing to see how this $\frac{1}{2}$ came, click here.

- Expected Value:

$$\mathbb{E} = \frac{1}{\mathcal{P}}$$

$$\mathbb{E}(w) = 2$$

- Therefore, it is expected to take 2 guesses to find g such that it is a good guess within the range $\left(a_{\frac{n}{4}}, a_{\frac{3n}{4}}\right)$.

Expected Recurrence Analysis

- Recurrence for $T(n)$:

$$\mathbb{E}(T(n)) = c \cdot 2 \cdot n + T\left(\frac{3n}{4}\right)$$

- The $\frac{3n}{4}$ term appears because, in the worst case, even if the guess was a good guess, the larger partition (assumed to be L) could still contain $\frac{3n}{4}$ elements.

- Expanding the recurrence:

$$\mathbb{E}(T(n)) = c \cdot 2 \cdot n + c \cdot 2 \cdot \frac{3n}{4} + c \cdot 2 \cdot \left(\frac{3}{4}\right)^2 n + \ldots$$

Expected Recurrence Analysis

- This forms a geometric series with a common ratio of $\frac{3}{4}$:

$$\sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i \leq 4$$

- Simplifying the series:

$$T(n) \leq O(n)$$

- Thus, even in the worst case where the good guess leads to an imbalance, the overall time complexity remains linear.

Median of Medians Algorithm For Selection

1. Divide array A into chunks of size 5.

$$A_1, A_2, A_3, \ldots, A_{\lceil n/5 \rceil}$$

2. Find the median of each chunk using insertion sort.

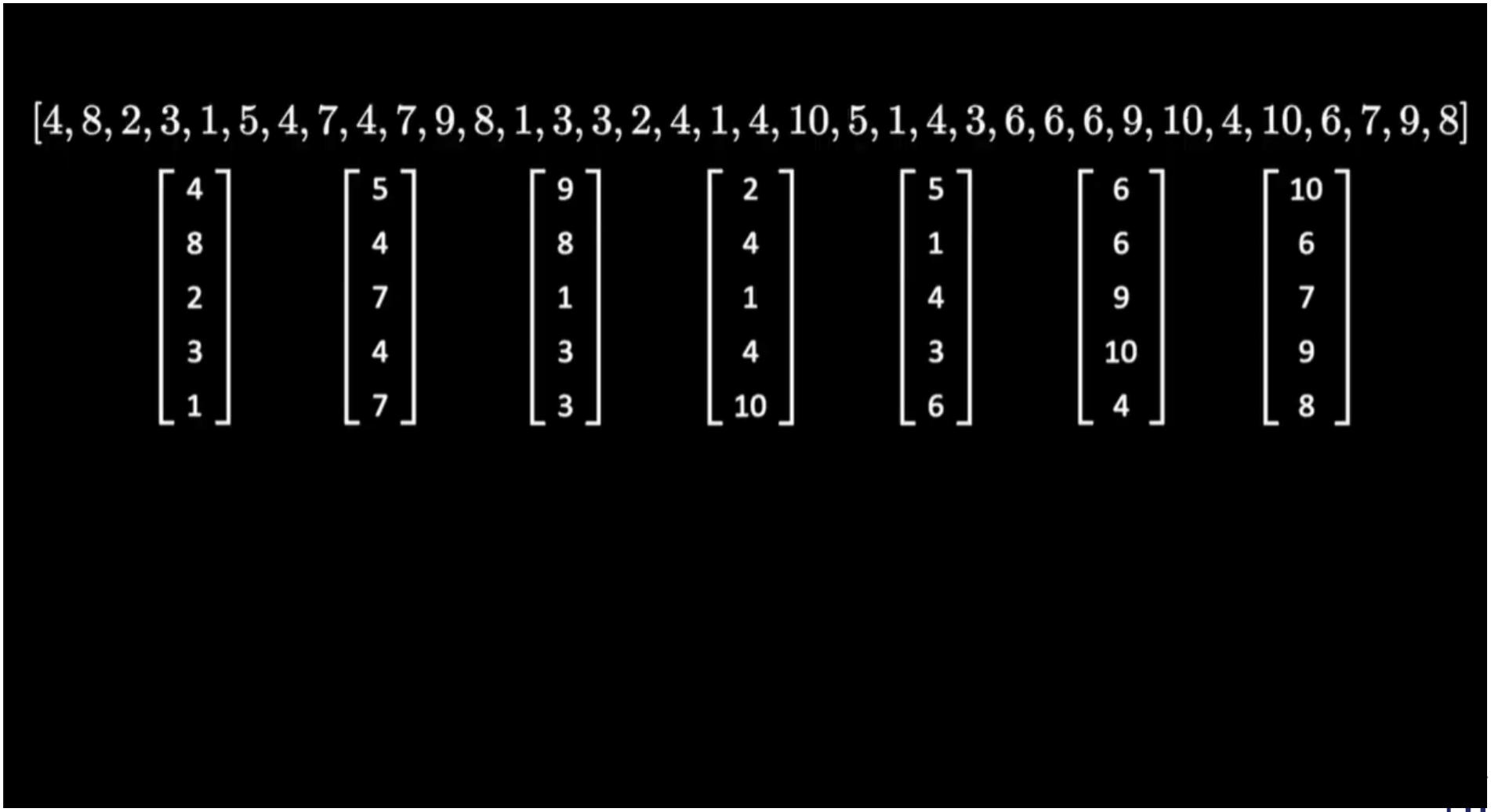
$$m_1, m_2, \ldots, m_{\lceil n/5 \rceil}$$

3. Recursively find the median of medians and now that is our guess:

$$g = \text{Select}(M, n/10)$$

4. Partition the array around g and recursively solve.

Visual Representation of the Algorithm



Randomized Select Algorithm

- Time Complexity:

$$T(n) = c\left(\frac{n}{5}\right) + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$$

- Explanation:
  - $c\left(\frac{n}{5}\right)$: Time taken to sort all the groups of size 5 and finding their medians.
  - $T\left(\frac{n}{5}\right)$: Time taken to recursively find the median of medians from the array of medians.
  - $T\left(\frac{7n}{10}\right)$: The next recursive call to the randomized select algorithm, which processes at most $\frac{7n}{10}$ elements, based on the partitioning.

Convex Function

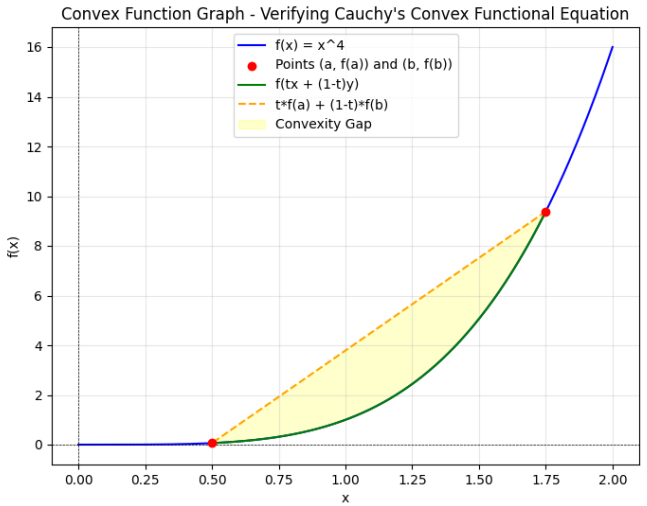
Definition of a Convex Function

If f is convex and $0 \leq t \leq 1$, then:

- $t \cdot f(a) + (1-t) \cdot f(b)$ represents a line segment.
- For $\forall a \leq x \leq b$, if the line segment lies on or above $f(x)$:

$$f(a+b) \geq f(a) + f(b) \qquad \text{(Eq. 1)}$$

This will be proven by the professor in the next class. For now, we assume it to be true.



Simplification

Using Eq. (1), we can simplify as follows:

$$T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \leq T\left(\frac{n}{5} + \frac{7n}{10}\right) \text{ or } T\left(\frac{9n}{10}\right)$$

Therefore:

$$T(n) \leq c\left(\frac{n}{5}\right) + T\left(\frac{9n}{10}\right)$$

Expanding recursively:

$$T(n) \leq c \cdot \frac{n}{5} + c \cdot \frac{n}{5}\left(\frac{9}{10}\right) + c \cdot \frac{n}{5}\left(\frac{9}{10}\right)^2 + \ldots$$

Simplification

The sum can be expressed as:

$$\sum_{i=0}^{\infty} c \cdot \frac{n}{5} \left(\frac{9}{10}\right)^i$$

Factoring $c \cdot \frac{n}{5}$, we get:

$$T(n) = c \cdot \frac{n}{5} \sum_{i=0}^{\infty} \left(\frac{9}{10}\right)^i$$

Using the formula for the sum of an infinite geometric series:

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r}, \quad \text{where } r = \frac{9}{10}$$

We get:

$$= c \cdot \frac{n}{5} \cdot \frac{1}{1 - \frac{9}{10}}$$

Simplifying further:

$$= c \cdot \frac{n}{5} \cdot 10 = c \cdot 2 \cdot n = O(n)$$

Conclusion

- Recurrence Relations:
$$T(n) = cn + T(|L|)$$
- Randomized Selection:
  - Median-of-medians selection guarantees linear time.
- Convex Functions:
  - Useful for bounding and analyzing recurrences.

Homework

- If the list was not distinct, then what should be done?
- What is important about chunks of size 5? For example, try sizes 3 and 7.
- What is the time complexity of the algorithm if we were to make chunks of size $\sqrt{n}$?