Algorithms

Runtime Analysis And Order Statistics

Dr. Mudassir Shabbir

LUMS University

January 27, 2026

Announcements

- HW 1 is due on Feb 02 at **11:59 AM**.
- Contestation Rule: You have 10 day after the grades are published to contest any grade.
- Midterm Exam/Long Quiz 1 on Fri 03/27, 2026 6:30p - 8:00p.
- Midterm Exam/Long Quiz 2 on Sat 02/21, 2026 2:00p - 3:30p.
- Guest lecture on Mon Feb 02, 2026.

Recap: Big Picture

- An Algorithm is a set of instructions to solve a problem.
- Time Complexity, via Asymptotic Notations, measures efficiency of an algorithm.
- Prune and Search is the first Algorithm design paradigm we studied.
- Median of medians Algorithm is an example of Prune and Search paradigm.
- We use convexity to solve $T(n) \leq T(n/5) + T(7n/10) + cn$.

Recap

- **Problem:** Find the kth 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 **kth** iteration is the **kth** 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)$^{\mathbf{th}}$** index of the array.
- **Time Complexity:** $O(n \log(n))$, where n is the size of the array.

Recap: PRUNE and SEARCH Paradigm

- **Algorithm 3: General Select Algorithm**
  - Guess g as an element from A.
  - Partition A into:
    - L: Elements less than g.
    - R: Elements greater than g.
  - If $|L| = k - 1$, return g.
  - Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
  - Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

$A = [\ 73\ 36\ 11\ 10\ 58\ 52\ 40\ 32\ 68\ 25\ 19\ 66\ 74\ 87\ 79\ 86\ 77\ ]$

The 9^{th} smallest in A? Guess : 74

$A = [\ 11\ 10\ 19\ 25\ 73\ 36\ 58\ 52\ 40\ 32\ 68\ 66\ ]$

The 9^{th} smallest in A? Guess : 25

$A = [\ 36\ 40\ 32\ 52\ 73\ 58\ 68\ 66\ ]$

The 5^{th} smallest in A? Guess : 52

$A = [\ 58\ 73\ 68\ 66\ ]$

[x] PRUNE n SEARCH ?

[x] Using a $\boxed{\text{Good Guess.}}$

The 1^{st} smallest in A? Guess : 58

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: General Select Algorithm

- Guess g: the approximate median!
- Partition A into:
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

Median of Medians: Approximate Median?

- Split the array into groups of 5
- Find the median of each group
- Recursively find the median of those medians

Median of Medians: Approximate Median?

- Split the array into groups of 5 $O(1)$ - nothing to do here
- Find the median of each group
- Recursively find the median of those medians

Median of Medians: Approximate Median?

- Split the array into groups of 5 $O(1)$ - nothing to do here
- Find the median of each group $O(n)$ - constant time per group
- Recursively find the median of those medians

Median of Medians: Approximate Median?

- Split the array into groups of 5 $O(1)$ - nothing to do here
- Find the median of each group $O(n)$ - constant time per group
- Recursively find the median of those medians $T(n/5)$ - size reduced by a factor of 5

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: General Select Algorithm

- Guess g: the approximate median!
- Partition A into:
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: General Select Algorithm

- Guess g: the approximate median!
- Partition A into:
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: General Select Algorithm

- Guess g: the approximate median! $a \times n + T(n/5)$
- Partition A into:
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: General Select Algorithm

- Guess g: the approximate median! $a \times n + T(n/5)$
- Partition A into: $b \times n$
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: General Select Algorithm

- Guess g: the approximate median! $a \times n + T(n/5)$
- Partition A into: $b \times n$
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L. $\leq T(7n/10)$
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: General Select Algorithm

- Guess g: the approximate median! $a \times n + T(n/5)$
- Partition A into: $b \times n$
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
  $\leq T(7n/10)$
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.
  $\leq T(7n/10)$

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: General Select Algorithm

- Guess g: the approximate median! $a \times n + T(n/5)$
- Partition A into: $b \times n$
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
  $\leq T(7n/10)$
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.
  $\leq T(7n/10)$

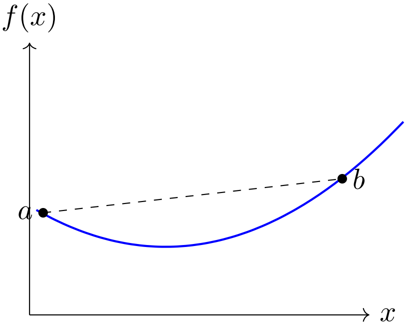
$a \times n + T(n/5) + T(7n/10) \leq c \times n + T(9n/10) = O(n)$.

Recap: Convex Functions

Definition

A function f is **convex** if for all a, b and $\alpha \in [0, 1]$,

$$f(\alpha a + (1-\alpha)b) \leq \alpha f(a) + (1-\alpha) f(b)$$



Recap: A Useful Property

Assumption

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

Claim

For $0 \leq \alpha \leq 1$,

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

- Apply convexity with $b = 0$
- Use $f(0) = 0$

Recap: Superadditivity of Convex Functions

Claim

For all $x, y \geq 0$,

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

Recap: Superadditivity of Convex Functions

Claim

For all $x, y \geq 0$,

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

- $f(x) + f(y) = f(x \frac{x+y}{x+y}) + f(y \frac{x+y}{x+y})$

Use $f(\alpha(x+y)) \leq \alpha f(x+y)$

- $f(x) + f(y) \leq \frac{x}{x+y} f(x+y) + \frac{y}{x+y} f(x+y)$
- $f(x) + f(y) \leq f(x+y)$

Recap: Applying This to the Recurrence

- Let $f(n) = T(n)$
- Assume T is non-decreasing and convex

Recap: Applying This to the Recurrence

- Let $f(n) = T(n)$
- Assume T is non-decreasing and convex

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

Recap: Applying This to the Recurrence

- Let $f(n) = T(n)$
- Assume T is non-decreasing and convex

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

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

Recap: Solving the Final Recurrence

Final Recurrence

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

Recap: Solving the Final Recurrence

Final Recurrence

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

- Total work:

$$cn + c(9n/10) + c(9^2 n/10^2) + \cdots$$

Recap: Solving the Final Recurrence

Final Recurrence

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

- Total work:

$$cn + c(9n/10) + c(9^2 n/10^2) + \cdots$$

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

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: General Select Algorithm

- Guess g: the approximate median!
- Partition A into:
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

$T(n) = O(n)$ in the worst case when using the median of medians.

Recap: PRUNE and SEARCH Paradigm

Algorithm 3: **General Select Algorithm** Any alternative?

- Guess g: the approximate median!
- Partition A into:
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

$T(n) = O(n)$ in the worst case when using the median of medians.

Randomized Selection Algorithm

Algorithm 3: Randomized Selection Algorithm

- Pick g uniformly at random from A
- Partition A into:
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

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 $O(n^2)$. This happens when each partitioning step reduces the problem size by only one element.

Average Time Complexity?

Basics of Probability Theory

Random Experiments

A **random experiment** is a process whose outcome cannot be predicted with certainty, even if the process is repeated under identical conditions.

Examples:

- Tossing a coin
- Rolling a die
- Choosing a pivot uniformly at random from an array

Sample Space and Outcomes

The **sample space** Ω is the set of all possible outcomes of a random experiment.

An **outcome** ω is a single element of Ω.

Sample Space and Outcomes

The **sample space** Ω is the set of all possible outcomes of a random experiment.

An **outcome** ω is a single element of Ω.

Example: If we choose a random index from $\{1, 2, \ldots, n\}$,

$$\Omega = \{1, 2, \ldots, n\}.$$

Sample Space and Outcomes

The **sample space** Ω is the set of all possible outcomes of a random experiment.

An **outcome** ω is a single element of Ω.

Example: If we toss two coins,

$$\Omega = \{HH, HT, TH, TT\}.$$

Sample Space and Outcomes

The **sample space** Ω is the set of all possible outcomes of a random experiment.

An **outcome** ω is a single element of Ω.

Example: If we roll a six-sided die,

$$\Omega = \{1, 2, 3, 4, 5, 6\}.$$

Probability Distribution

A **probability distribution** $\mathcal{P}$ assigns a probability to each outcome in the sample space Ω with the following properties:

- For each outcome, $\mathcal{P}(\omega_i) \geq 0$.
- The sum of the probabilities of all outcomes equals 1:

$$\sum_{\omega_i \in \Omega} \mathcal{P}(\omega_i) = 1.$$

Probability Distribution

Example: If we roll a fair six-sided die, the sample space is

$$\Omega = \{1, 2, 3, 4, 5, 6\}.$$

The probability distribution is

$$\mathcal{P}(i) = \frac{1}{6} \text{ for } i = 1, 2, 3, 4, 5, 6.$$

Example: If we toss two fair coins, the sample space is

$$\Omega = \{HH, HT, TH, TT\}.$$

The probability distribution is

$$\mathcal{P}(HH) = \mathcal{P}(HT) = \mathcal{P}(TH) = \mathcal{P}(TT) = \frac{1}{4}.$$

Events

An **event** is a subset $E \subseteq \Omega$.

The event E occurs if the outcome $\omega \in E$.
Example: For the sample space:

$$\Omega = \{HH, HT, TH, TT\}.$$

An event could be:

$$E = \{HT, TH, HH\},$$

which represents the event of getting at least one head.

Random Variables

A **random variable** is a function

$$X : \Omega \to \mathbb{R}.$$

It assigns a numerical value to each outcome of a random experiment.

Expectation (Definition)

Let X be a discrete random variable.

The **expected value** of X is defined as

$$\mathbb{E}[X] = \sum_{x} x \cdot \Pr(X = x),$$

where the sum is over all values x that X can take.

Expectation (Key Properties)

Expectation satisfies several important properties:

- **Linearity:**
$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$
- Holds *regardless of independence*

This property is fundamental in analyzing randomized algorithms.

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.

Expected Value Calculation: Fair Die

Scenario

Let X be the outcome of a fair six-sided die: possible values are $1, 2, 3, 4, 5, 6$.

Calculation

Uniform distribution: $\Pr(X = i) = \frac{1}{6}$ for $i = 1, \ldots, 6$.

$$\mathbb{E}[X] = \sum_{i=1}^{6} i \cdot \frac{1}{6} = \frac{1}{6}(1 + 2 + 3 + 4 + 5 + 6) = \frac{21}{6} = 3.5$$

Expected Value Calculation: Sum of Two Dice (Direct)

Scenario

Roll two independent fair dice. Let $S = D_1 + D_2$ be their sum.

Calculation (Definition)

The distribution of S over $\{2, 3, \ldots, 12\}$ has probabilities proportional to $1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1$ out of 36. Using $\mathbb{E}[S] = \sum_s s \Pr(S = s)$:

$$\frac{2 \cdot 1 + 3 \cdot 2 + 4 \cdot 3 + 5 \cdot 4 + 6 \cdot 5 + 7 \cdot 6 + 8 \cdot 5 + 9 \cdot 4 + 10 \cdot 3 + 11 \cdot 2 + 12 \cdot}{36}$$

Expected Value Calculation: Sum of Two Dice

Scenario

Roll two independent fair dice. Let $S = D_1 + D_2$ be their sum.

Calculation (Linearity)

By linearity of expectation and identical distributions:

$$\mathbb{E}[S] = \mathbb{E}[D_1] + \mathbb{E}[D_2] = 3.5 + 3.5 = 7.$$

No need to enumerate the 36 outcomes.

Randomized Selection: The Setup

In randomized selection:

- The pivot is chosen uniformly at random,
- The recursion depends on the pivot's rank,
- The running time becomes a random variable.

Our goal is to analyze the **expected running time**.

What We Will Analyze Next

Next, we will:

- Define a recurrence for the running time,
- Take expectations on both sides,
- Prove that randomized selection runs in $\Theta(n)$ expected time.

This will rely almost entirely on the definitions introduced so far.

Algorithm 3: Randomized Selection Algorithm

- Pick g uniformly at random from A
- Partition A into:
  - L: Elements less than g.
  - R: Elements greater than g.
- If $|L| = k - 1$, return g.
- Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
- Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.