Algorithms

Runtime Analysis And Order Statistics

Dr. Mudassir Shabbir

LUMS University

January 28, 2026

Announcements

- HW 1 is due on Feb 02 at **11:59 AM**.
- Midterm Exam/Long Quiz 1 on Sat 02/21, 2026 2:00p - 3:30p.
- Midterm Exam/Long Quiz 2 on Fri 03/27, 2026 6:30p - 8:00p.
- Guest lecture on Mon Feb 02, 2026.
- No office hours on Mon Feb 02, 2026.

Contestation Rule: You have 10 day after the grades are published to contest any grade.

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:

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

For Uniform Distribution:

$$\mathcal{P}(\omega_i) = \frac{1}{|\Omega|} \text{ for all } \omega_i \in \Omega.$$

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.

Uniform Distribution

If all outcomes in Ω are equally likely,

$$\Pr(E) = \frac{|E|}{|\Omega|}.$$

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.

Expectation of Geometric Distribution

A **geometric random variable** X with parameter p counts the number of Success/Fail trials until the first Success, where each trial has Success probability p. The probability mass function is

$$\Pr(X = k) = (1-p)^{k-1}p, \quad k = 1, 2, \ldots$$

The expected value of X is

$$\mathbb{E}[X] = \frac{1}{p}.$$

Example: Expected Tosses for First Head

Consider tossing a fair coin (success probability $p = \frac{1}{2}$) until the first head appears. Let X be the number of tosses until the first head.

Then X is a geometric random variable with parameter $p = \frac{1}{2}$. The

expected number of tosses is

$$\mathbb{E}[X] = \frac{1}{p} = \frac{1}{\frac{1}{2}} = 2.$$

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.

Randomized Selection Algorithm

Algorithm 3: Randomized Selection Algorithm

1. Pick g uniformly at random from A - repeat if $n/4 > \textit{rank}(g) \geq 3n/4$
2. Partition A into:
    - L: Elements less than g.
    - R: Elements greater than g.
3. If $|L| = k - 1$, return g.
4. Else if $|L| \geq k$, recursively SELECT k^{th} element from L.
5. Otherwise, recursively SELECT $(k - |L| - 1)^{th}$ element from R.

Which Pivots Are Acceptable?

$$A = \{a_1,\ a_2,\ a_3,\ a_4,\ a_5,\ a_6,\ a_7,\ a_8,\ a_9,\ a_{10},\ a_{11},\ a_{12},\ a_{13},\ a_{14},\ a_{15},\ a_{16}\}$$

Which Pivots Are Acceptable?

$$A = \{a_1,\ a_2,\ a_3,\ a_4,\ a_5,\ a_6,\ a_7,\ a_8,\ a_9,\ a_{10},\ a_{11},\ a_{12},\ a_{13},\ a_{14},\ a_{15},\ a_{16}\}$$

$$A_{\text{sorted}} = \{b_1,\ b_2,\ b_3,\ b_4,\ b_5,\ b_6,\ b_7,\ b_8,\ b_9,\ b_{10},\ b_{11},\ b_{12},\ b_{13},\ b_{14},\ b_{15},\ b_{16}\}$$

Which Pivots Are Acceptable?

$$A = \{a_1,\ a_2,\ a_3,\ a_4,\ a_5,\ a_6,\ a_7,\ a_8,\ a_9,\ a_{10},\ a_{11},\ a_{12},\ a_{13},\ a_{14},\ a_{15},\ a_{16}\}$$

$$A_{\text{sorted}} = \{b_1,\ b_2,\ b_3,\ b_4,\ b_5,\ b_6,\ b_7,\ b_8,\ b_9,\ b_{10},\ b_{11},\ b_{12},\ b_{13},\ b_{14},\ b_{15},\ b_{16}\}$$

$$\underbrace{\{b_1,\ b_2,\ b_3,\ b_4\}}_{\text{too small}}\ \underbrace{\{b_5,\ b_6,\ b_7,\ b_8,\ b_9,\ b_{10},\ b_{11},\ b_{12}\}}_{\text{good pivots}}\ \underbrace{\{b_{13},\ b_{14},\ b_{15},\ b_{16}\}}_{\text{too large}}$$

Which Pivots Are Acceptable?

$$A = \{a_1,\ a_2,\ a_3,\ a_4,\ a_5,\ a_6,\ a_7,\ a_8,\ a_9,\ a_{10},\ a_{11},\ a_{12},\ a_{13},\ a_{14},\ a_{15},\ a_{16}\}$$

$$A_{\text{sorted}} = \{b_1,\ b_2,\ b_3,\ b_4,\ b_5,\ b_6,\ b_7,\ b_8,\ b_9,\ b_{10},\ b_{11},\ b_{12},\ b_{13},\ b_{14},\ b_{15},\ b_{16}\}$$

$$\underbrace{\{b_1,\ b_2,\ b_3,\ b_4\}}_{\text{too small}}\ \underbrace{\{b_5,\ b_6,\ b_7,\ b_8,\ b_9,\ b_{10},\ b_{11},\ b_{12}\}}_{\text{good pivots}}\ \underbrace{\{b_{13},\ b_{14},\ b_{15},\ b_{16}\}}_{\text{too large}}$$

Key Observation

Any element a_i whose rank satisfies

$$4 < \mathit{rank}(a_i) \leq 12$$

leads to a recursive subproblem of size at most $\frac{3n}{4}$.

Good vs Bad Pivots

Definition

A pivot g is **good** if:

$$\frac{n}{4} \leq \textit{rank}(g) \leq \frac{3n}{4}$$

Good vs Bad Pivots

Definition

A pivot g is **good** if:

$$\frac{n}{4} \leq \textit{rank}(g) \leq \frac{3n}{4}$$

- Guarantees recursive call on at most $\frac{3n}{4}$ elements
- Eliminates highly unbalanced partitions

Probability of Selecting a Good Pivot

- Total possible pivots: n
- Good pivots: middle $\frac{n}{2}$ elements

Probability of Selecting a Good Pivot

- Total possible pivots: n
- Good pivots: middle $\frac{n}{2}$ elements

$$\Pr[\text{good pivot}] = \frac{n/2}{n} = \frac{1}{2}$$

Probability of Selecting a Good Pivot

- Total possible pivots: n
- Good pivots: middle $\frac{n}{2}$ elements

$$\Pr[\text{good pivot}] = \frac{n/2}{n} = \frac{1}{2}$$

Expected Number of Trials

Expected number of random selections until a good pivot:

$$\mathbb{E}[\text{trials}] = \frac{1}{1/2} = 2$$

Cost per Recursive Level

- Each attempt to find a good pivot costs $\Theta(n)$
- Expected number of attempts is constant

Cost per Recursive Level

- Each attempt to find a good pivot costs $\Theta(n)$
- Expected number of attempts is constant

Expected Cost

$$\mathbb{E}[\text{work before recursion}] = \Theta(n)$$

Cost per Recursive Level

- Each attempt to find a good pivot costs $\Theta(n)$
- Expected number of attempts is constant

Expected Cost

$$\mathbb{E}[\text{work before recursion}] = \Theta(n)$$

Key Insight

Rejecting bad pivots does not change asymptotic cost.

Expected Recurrence

- After selecting a good pivot:
- Recursive call on at most $\frac{3n}{4}$ elements

Expected Recurrence

- After selecting a good pivot:
- Recursive call on at most $\frac{3n}{4}$ elements

Expected Recurrence

$$\mathbb{E}[T(n)] \leq \mathbb{E}[T(3n/4)] + cn$$

for some constant $c > 0$.

Solving the Recurrence

Unrolling the recurrence:

$$\mathbb{E}[T(n)] \leq cn + c\frac{3n}{4} + c\left(\frac{3}{4}\right)^2 n + \cdots$$

Solving the Recurrence

Unrolling the recurrence:

$$\mathbb{E}[T(n)] \leq cn + c\frac{3n}{4} + c\left(\frac{3}{4}\right)^2 n + \cdots$$

Geometric Series

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

Solving the Recurrence

Unrolling the recurrence:

$$\mathbb{E}[T(n)] \leq cn + c\frac{3n}{4} + c\left(\frac{3}{4}\right)^2 n + \cdots$$

Geometric Series

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

$$\sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i = \frac{1}{1 - 3/4} = 4$$

Final Expected Time Bound

$$\mathbb{E}[T(n)] \leq 4cn$$

Final Expected Time Bound

$$\mathbb{E}[T(n)] \leq 4cn$$

Final Result

$$\boxed{\mathbb{E}[T(n)] = \Theta(n)}$$

Final Expected Time Bound

$$\mathbb{E}[T(n)] \leq 4cn$$

Final Result

$$\boxed{\mathbb{E}[T(n)] = \Theta(n)}$$

Takeaway

Randomization yields linear time *in expectation*, even though worst-case time is quadratic.

What if we didn't reject bad pivots?

Algorithm 3: Randomized Selection Algorithm

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

Randomized Selection: Running Time as a Random Variable

Let $T(n)$ denote the running time of **Randomized-Select** on an array of size n.

Then we can write a recurrence:

$$T(n) = T(\text{size of subarray containing } k) + cn$$

where cn accounts for the partitioning step.

Randomized Selection: Running Time as a Random Variable

Let $T(n)$ denote the running time of **Randomized-Select** on an array of size n.

Then we can write a recurrence:

$$T(n) = T(\text{size of subarray containing } k) + cn$$

where cn accounts for the partitioning step.

Observation: The size of the recursive subarray is a *random variable*.

Expected Running Time: Step 1

Let X be the size of the subarray we recurse into.

Then the recurrence becomes:

$$T(n) = T(X) + cn$$

Expected Running Time: Step 1

Let X be the size of the subarray we recurse into.

Then the recurrence becomes:

$$T(n) = T(X) + cn$$

Taking expectations on both sides:

$$\mathbb{E}[T(n)] = \mathbb{E}[T(X)] + cn$$

Expected Running Time: Step 1

Let X be the size of the subarray we recurse into.

Then the recurrence becomes:

$$T(n) = T(X) + cn$$

Taking expectations on both sides:

$$\mathbb{E}[T(n)] = \mathbb{E}[T(X)] + cn$$

Now we need to compute $\mathbb{E}[T(X)]$.

Distribution of Pivot Position

Suppose the pivot is chosen uniformly at random from n elements.

Then the pivot's rank i satisfies:

$$\Pr(\text{pivot rank} = i) = \frac{1}{n}, \quad i = 1, 2, \ldots, n$$

Distribution of Pivot Position

Suppose the pivot is chosen uniformly at random from n elements.

Then the pivot's rank i satisfies:

$$\Pr(\text{pivot rank} = i) = \frac{1}{n}, \quad i = 1, 2, \ldots, n$$

The size of the subarray we recurse into is either $i - 1$ or $n - i$ depending on k.

Expected Recursive Size (Simplified Bound)

To simplify the analysis, note that the size of the recursive subarray is at most:

$$\max(i - 1, n - i)$$

Expected Recursive Size (Simplified Bound)

To simplify the analysis, note that the size of the recursive subarray is at most:

$$\max(i-1, n-i)$$

So we can write a rough upper bound:

$$\mathbb{E}[T(n)] \leq cn + \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[T(\max(i-1, n-i))]$$

Bounding the Recursive Term

Observe that at least half of the pivots satisfy:

$$\frac{n}{4} \leq \max(i-1, n-i) \leq \frac{3n}{4}$$

Bounding the Recursive Term

Observe that at least half of the pivots satisfy:

$$\frac{n}{4} \leq \max(i-1, n-i) \leq \frac{3n}{4}$$

Hence, the "large" recursive call is at most $\frac{3n}{4}$ with probability $\geq \frac{1}{2}$.

Bounding the Recursive Term

Observe that at least half of the pivots satisfy:

$$\frac{n}{4} \leq \max(i-1, n-i) \leq \frac{3n}{4}$$

Hence, the "large" recursive call is at most $\frac{3n}{4}$ with probability $\geq \frac{1}{2}$. So we can write a simpler inequality:

$$\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(n)] + \frac{1}{2}\mathbb{E}[T(3n/4)]$$

Bounding the Recursive Term

Observe that at least half of the pivots satisfy:

$$\frac{n}{4} \leq \max(i-1, n-i) \leq \frac{3n}{4}$$

Hence, the "large" recursive call is at most $\frac{3n}{4}$ with probability $\geq \frac{1}{2}$. So we can write a simpler inequality:

$$\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(n)] + \frac{1}{2}\mathbb{E}[T(3n/4)]$$

Solve for $\mathbb{E}[T(n)]$ next.

Solving the Inequality

From

$$\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(n)] + \frac{1}{2}\mathbb{E}[T(3n/4)],$$

Solving the Inequality

From

$$\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(n)] + \frac{1}{2}\mathbb{E}[T(3n/4)],$$

subtract $\frac{1}{2}\mathbb{E}[T(n)]$ from both sides:

$$\frac{1}{2}\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(3n/4)]$$

Solving the Inequality

From

$$\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(n)] + \frac{1}{2}\mathbb{E}[T(3n/4)],$$

subtract $\frac{1}{2}\mathbb{E}[T(n)]$ from both sides:

$$\frac{1}{2}\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(3n/4)]$$

Multiply both sides by 2:

$$\mathbb{E}[T(n)] \leq 2cn + \mathbb{E}[T(3n/4)]$$

Solving the Inequality

From

$$\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(n)] + \frac{1}{2}\mathbb{E}[T(3n/4)],$$

subtract $\frac{1}{2}\mathbb{E}[T(n)]$ from both sides:

$$\frac{1}{2}\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(3n/4)]$$

Multiply both sides by 2:

$$\mathbb{E}[T(n)] \leq 2cn + \mathbb{E}[T(3n/4)]$$

This is a standard geometric recursion.

Unfolding the Recursion

$$\mathbb{E}[T(n)] \leq 2cn + \mathbb{E}[T(3n/4)]$$

Unfolding the Recursion

$$\mathbb{E}[T(n)] \leq 2cn + \mathbb{E}[T(3n/4)]$$

Unfold recursively:

$$\mathbb{E}[T(n)] \leq 2cn + 2c\frac{3n}{4} + 2c\left(\frac{3}{4}\right)^2 n + \dots$$

Unfolding the Recursion

$$\mathbb{E}[T(n)] \leq 2cn + \mathbb{E}[T(3n/4)]$$

Unfold recursively:

$$\mathbb{E}[T(n)] \leq 2cn + 2c\frac{3n}{4} + 2c\left(\frac{3}{4}\right)^2 n + \ldots$$

This is a geometric series:

$$\mathbb{E}[T(n)] \leq 2cn\sum_{k=0}^{\infty}\left(\frac{3}{4}\right)^k = 2cn \cdot 4 = 8cn$$

Unfolding the Recursion

$$\mathbb{E}[T(n)] \leq 2cn + \mathbb{E}[T(3n/4)]$$

Unfold recursively:

$$\mathbb{E}[T(n)] \leq 2cn + 2c\frac{3n}{4} + 2c\left(\frac{3}{4}\right)^2 n + \dots$$

This is a geometric series:

$$\mathbb{E}[T(n)] \leq 2cn\sum_{k=0}^{\infty}\left(\frac{3}{4}\right)^k = 2cn \cdot 4 = 8cn$$

Conclusion: $\mathbb{E}[T(n)] = O(n)$

Divide & Conquer Strategy

Divide & Conquer Strategy

Idea

Divide & Conquer works by:

1. **Divide**: Split the problem into smaller subproblems.
2. **Conquer**: Solve each subproblem recursively.
3. **Combine**: Merge subproblem solutions to get the final answer.

Divide & Conquer Strategy

Idea

Divide & Conquer works by:

1. **Divide**: Split the problem into smaller subproblems.
2. **Conquer**: Solve each subproblem recursively.
3. **Combine**: Merge subproblem solutions to get the final answer.

Example: Merge Sort

- Divide array in half
- Recursively sort each half
- Merge sorted halves

Divide & Conquer Strategy

Idea

Divide & Conquer works by:

1. **Divide**: Split the problem into smaller subproblems.
2. **Conquer**: Solve each subproblem recursively.
3. **Combine**: Merge subproblem solutions to get the final answer.

Example: Merge Sort

- Divide array in half
- Recursively sort each half
- Merge sorted halves

Observation

All subproblems are kept; no elements are discarded.

Prune & Search Strategy

Idea

Prune & Search works by:

1. **Select/guess**: Pick a candidate element (e.g., pivot).
2. **Prune**: Eliminate a fixed fraction of elements guaranteed not to contain the solution.
3. **Recurse**: Solve the smaller remaining problem.

Prune & Search Strategy

Idea

Prune & Search works by:

1. **Select/guess**: Pick a candidate element (e.g., pivot).
2. **Prune**: Eliminate a fixed fraction of elements guaranteed not to contain the solution.
3. **Recurse**: Solve the smaller remaining problem.

Key Difference

Unlike Divide & Conquer:

- Only a subset of the original problem is kept
- Progress guaranteed by eliminating a fraction of candidates each time

QuickSort: Divide & Conquer in Action

- Pick a pivot
- Partition array into

$$L = \text{elements } < \text{pivot}, \quad R = \text{elements } > \text{pivot}$$

- Recursively sort L and R

QuickSort: Divide & Conquer in Action

- Pick a pivot
- Partition array into

$$L = \text{elements } < \text{pivot}, \quad R = \text{elements } > \text{pivot}$$

- Recursively sort L and R

Observation

QuickSort keeps both sides — all elements are part of recursive subproblems. **Thus, it is Divide & Conquer.**

QuickSort: Divide & Conquer in Action

- Pick a pivot
- Partition array into

$$L = \text{elements } < \text{pivot}, \quad R = \text{elements } > \text{pivot}$$

- Recursively sort L and R

Observation

QuickSort keeps both sides — all elements are part of recursive subproblems. **Thus, it is Divide & Conquer.**

Contrast

Randomized Selection discards one side entirely (the side not containing k^{th} element), making it **Prune & Search**.

QuickSort Algorithm

Input: Array $A[1..n]$

1. **Base case:** If $n \leq 1$, return A.
2. Pick a **pivot** element p from A.
3. Partition A into:

$$L = \{x \in A : x < p\}, \quad R = \{x \in A : x > p\}$$

4. Recursively sort L and R.
5. Concatenate L, p, R to get the sorted array.

Partitioning in QuickSort

$$A = [9, 3, 7, 5, 2, 8, 6, 1, 4]$$

Partitioning in QuickSort

$$A = [9, 3, 7, 5, 2, 8, 6, 1, 4]$$

Pick pivot $p = 5$

Partitioning in QuickSort

$$A = [9, 3, 7, 5, 2, 8, 6, 1, 4]$$

Pick pivot $p = 5$

$$L = [3, 2, 1, 4], \quad R = [9, 7, 8, 6]$$

Partitioning in QuickSort

$$A = [9, 3, 7, 5, 2, 8, 6, 1, 4]$$

Pick pivot $p = 5$

$$L = [3, 2, 1, 4], \quad R = [9, 7, 8, 6]$$

Observation

Pivot ends up in its final sorted position; all elements $<$ pivot go left, $>$ pivot go right.

Recursion Tree of QuickSort

$$A = [9, 3, 7, 5, 2, 8, 6, 1, 4]$$

Recursion Tree of QuickSort

$$A = [9, 3, 7, 5, 2, 8, 6, 1, 4]$$

- Level 0: full array
- Level 1: partitions $L = [3, 2, 1, 4]$, pivot 5, $R = [9, 7, 8, 6]$
- Level 2: recursively sort L and R

Recursion Tree of QuickSort

$$A = [9, 3, 7, 5, 2, 8, 6, 1, 4]$$

- Level 0: full array
- Level 1: partitions $L = [3, 2, 1, 4]$, pivot 5, $R = [9, 7, 8, 6]$
- Level 2: recursively sort L and R

Observation

All elements are retained at each level — hence QuickSort is a **Divide & Conquer** algorithm.

QuickSort Recurrence and Complexity

- Let $T(n)$ be the time to sort n elements.
- Partitioning takes $\Theta(n)$
- Recursion on sizes k and $n-k-1$:

$$T(n) = T(k) + T(n-k-1) + cn$$

QuickSort Recurrence and Complexity

- Let $T(n)$ be the time to sort n elements.
- Partitioning takes $\Theta(n)$
- Recursion on sizes k and $n-k-1$:

$$T(n) = T(k) + T(n-k-1) + cn$$

Average Case

Balanced splits $\Rightarrow k \approx n/2$:

$$T(n) = 2T(n/2) + cn \implies T(n) = \Theta(n \log n)$$

QuickSort Recurrence and Complexity

- Let $T(n)$ be the time to sort n elements.
- Partitioning takes $\Theta(n)$
- Recursion on sizes k and $n-k-1$:

$$T(n) = T(k) + T(n-k-1) + cn$$

Average Case

Balanced splits $\Rightarrow k \approx n/2$:

$$T(n) = 2T(n/2) + cn \implies T(n) = \Theta(n \log n)$$

Worst Case

Extreme splits (pivot smallest/largest):

$$T(n) = T(0) + T(n-1) + cn \implies T(n) = \Theta(n^2)$$