DISCRETE SHOW AND TELL

PASSWORD STRENGTH CHECKER

MADE BY AMMAR ANAS

MEMEBERS

Ammar Anas BSAI24056

Muhammad Rayyan Malik
BSAI24066

Muhammad Anas Fayyaz
BSAI24072

Qasim Bin Shahzad
BSAI24070

Syed Muhammad Fawad Hassan Rizvi
BSAI24051

THE IMPORTANCE OF DISCRETE MATHS

Role In Computing:

- Provides the building blocks for algorithms, data structures, and logical reasoning.
- Used in areas like cybersecurity, software engineering, and artificial intelligence.

Applications:

- **Cryptography:** Secures online communication through encryption.
- **Graph Theory:** Models networks (e.g., social, computer, transport).
- **Set Theory:** Organizes and structures data effectively.
- **Logic and Proofs:** Validates the correctness of software systems.

USE IN PASSWORD SECURITY

Core Role in Strength Evaluation:

- Analyzes and validates password rules through mathematical principles.
- Ensures passwords are resistant to guessing and cracking techniques.

Concepts Applied:

- **Combinatorics:** Determines possible password combinations**.**
- **Set Theory:** Enforces rules like character diversity.
- **Probability:** Calculates the likelihood of guessing.
- **Graph Theory:** Detects patterns like consecutive sequences.
- **Algorithm Complexity:** Ensures efficient real-time checking.

PASSWORD CHECKER

STEP 1

User enters the Password

STEP 2

The program evaluates the password based on:

LENGTH

DIVERSITY OF CHARACTERS

REPITION AND PATTERNS

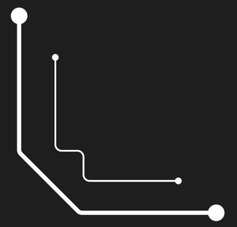
FEW SPECIAL BONUS POINTS

STEP 3

Outputs the strength of the Password

KEY FEATURES

- Interactive
- User Friendly
- Provides Detailed Feedback



```
PASSWORD STRENGTH CHECKER MADE BY AMMAR ANAS

Enter a password to check its strength.
Password can be max 30 characters.
Password cannot include any spaces.

Press 'D' to toggle point breakdown.

Press 'Enter' to continue to password entry.
Press 'Q' to exit.
```

```
Enter your Password: *********
```

```
Enter your Password: *********

Password Strength: Weak

Password: AmmarAnas

Point Breakdown:

Length Points: 2
Lowercase Points: 2
Uppercase Points: 1
Digit Points: 0
Special Character Points: 0
Repeats Penalty: -1
Sequence Penalty: 0
Bonus Points: 0
Total Points: 4
```

POINTS BREAKDOWN

```
Length:
  - 8-12 characters: + 2 points
  - 13-20 characters: + 4 points
  - 21-30 characters: + 6 points
Character Diversity:
  - Lowercase letters:
      1-3: +1 point
      4 or more: +2 points
  - Uppercase letters:
      1-3: +1 point
      4 or more: +2 points
  - Digits:
      1-2: +1 point
      3-5: +2 points
      6 or more: +3 points
  - Special characters:
      1-3: +1 point
      4-6: +2 points
      7 or more: +3 points
Bonus:
  - 3 character groups: +2 points
  - 4 character groups: +4 points
```

```
Password Strength Based on Points:
0 to 4 points: Very Weak
5 to 9 points: Weak
10 to 14 points: Moderate
15 to 19 points: Strong
20 to 24 points: Very Strong
25+ points: Excellent
```

```
Penalties:
  - Repeated characters:
      1-2 repeats: -1 point
      3-4 repeats: -2 points
      5 or more repeats: -3 points
  - Consecutive sequences:
      1 sequence of 3 characters: -1 point
      2 or more sequences: -2 points
```

COMBINATORICS

COMBINATORICS

- Combinatorics studies the total number of possible arrangements or combinations in a set.
- In password security, combinatorics determines the size of the password search space.

Why it matters:

- Larger combinations mean higher entropy, making passwords harder to guess.

- **The formula:**
  Total Combinations = $(\mathbf{N})^{L}$

- **N:** Number of possible characters.
- **L :** Password length.

COMBINATORICS

```
Length:
  - 8-12 characters: + 2 points
  - 13-20 characters: + 4 points
  - 21-30 characters: + 6 points
```

```
// Length Points
if (length >= 8 && length <= 12)
{
    lengthPoints = 2;
    points += 2;
}
else if (length >= 13 && length <= 20)
{
    lengthPoints = 4;
    points += 4;
}
else if (length >= 21)
{
    lengthPoints = 6;
    points += 6;
}
else
{
    lengthPoints = 0;
}
```

COMBINATORICS

Lets calculate the total combinations for a password length of 4 using 62 characters (uppercase, lowercase, digits):

$$62^4 = 14{,}776{,}336 \text{ combinations}$$

AND

Lets calculate the total combinations for a password length of 8 using 62 characters (uppercase, lowercase, digits):

$$62^8 = 218{,}340{,}105{,}584{,}896 \text{ combinations}$$

SET THEORY

SET THEORY

- Set theory organizes elements into groups (sets) and examines their relationships.

Why it matters:

- Passwords must include diverse characters to resist brute-force attacks.

- **Examples of sets:**

  U = uppercase letters (A-Z)
  L = lowercase letters (a-z)
  D = digits (0-9)
  S = special charecters (!,@,#,etc)

SET THEORY

Password strength is evaluated by ensuring it includes at least one character from each set:

uppercase, lowercase, digits, and special symbols.

This diversity strengthens the password and enhances security.

```
// Character Diversity Points
int lowercase = 0, uppercase = 0, digits = 0, special = 0;
for (int i = 0; i < length; i++) {
    if (islower(password[i])) lowercase++;
    else if (isupper(password[i])) uppercase++;
    else if (isdigit(password[i])) digits++;
    else special++;
}

if (lowercase >= 1 && lowercase <= 3) {
    lowercasePoints = 1;
    points += 1;
}
else if (lowercase > 3) {
    lowercasePoints = 2;
    points += 2;
}
else {
    lowercasePoints = 0;
}

if (uppercase >= 1 && uppercase <= 3) {
    uppercasePoints = 1;
    points += 1;
}
else if (uppercase > 3) {
    uppercasePoints = 2;
    points += 2;
}
else {
    uppercasePoints = 0;
}
```

```
Character Diversity:
  - Lowercase letters:
      1-3: +1 point
      4 or more: +2 points
  - Uppercase letters:
      1-3: +1 point
      4 or more: +2 points
  - Digits:
      1-2: +1 point
      3-5: +2 points
      6 or more: +3 points
  - Special characters:
      1-3: +1 point
      4-6: +2 points
      7 or more: +3 points
```

PROBABILITY

PROBABILITY

- Probability measures the likelihood of an event occurring.
- In password security, it calculates the chances of an attacker guessing the password.

Why it matters:

- Predictable patterns (e.g., "Password999") are easy to guess.
- By penalizing such patterns, we reduce the likelihood of weak passwords being used.

PROBABILITY

Repetition is penalized, considering probability, to encourage users to create stronger, less predictable passwords that are harder to guess.

```
// Repetition Penalty Points
int repeats = 0;
for (int i = 1; i < length; i++) {
    if (password[i] == password[i - 1]) repeats++;
}
if (repeats >= 1 && repeats <= 2) {
    repeatsPenalty = -1;
    points -= 1;
}
else if (repeats >= 3 && repeats <= 4) {
    repeatsPenalty = -2;
    points -= 2;
}
else if (repeats > 4) {
    repeatsPenalty = -3;
    points -= 3;
}
else {
    repeatsPenalty = 0;
}
```

```
Penalties:
  - Repeated characters:
      1-2 repeats: -1 point
      3-4 repeats: -2 points
      5 or more repeats: -3 points
  - Consecutive sequences:
      1 sequence of 3 characters: -1 point
      2 or more sequences: -2 points
```

GRAPH THEORY

GRAPH THEORY

- A graph consists of nodes (or vertices) and edges connecting them.
- Graph theory studies these structures to model relationships and patterns.
- In password checking, graph theory can identify patterns like consecutive sequences of characters.

Why it matters:

- Consecutive patterns (e.g., “abc”, “1234”) make passwords predictable and vulnerable to attacks.
- By detecting and penalizing such sequences, your program encourages stronger passwords.

GRAPH THEORY

Graph theory helps identify patterns and relationships between characters, such as consecutive sequences in a password (e.g., “abc” or “123”).

By representing characters as nodes and their connections as edges, it becomes easier to detect predictable patterns that weaken password strength.

```java
// Sequence Penalty Points
int sequences = 0;
for (int i = 2; i < length; i++) {
    if ((password[i] == password[i - 1] + 1) && (password[i - 1] == password[i - 2] + 1)) {
        sequences++;
    }
}
if (sequences == 1) {
    sequencePenalty = -1;
    points -= 1;
}
else if (sequences > 1) {
    sequencePenalty = -2;
    points -= 2;
}
else {
    sequencePenalty = 0;
}
```

```
Penalties:
  - Repeated characters:
      1-2 repeats: -1 point
      3-4 repeats: -2 points
      5 or more repeats: -3 points
  - Consecutive sequences:
      1 sequence of 3 characters: -1 point
      2 or more sequences: -2 points
```

GRAPH THEORY

INCASE OF ABC

INCASE OF 123

ALGORITHM COMPLEXITY

ALGORITHM COMPLEXITY

- Algorithm complexity refers to the efficiency of an algorithm, typically measured in terms of time complexity and space complexity.
- Time complexity (e.g., **O(n)**) measures how the execution time of an algorithm grows relative to the size of the input.
- In your password strength checker, time complexity is important because you want the program to evaluate passwords quickly, even if they are long.

Why it matters:

- Efficient Algorithms ensure the password strength checker can handle longer passwords and multiple checks in real-time.
- In the worst case, an algorithm with high time complexity (e.g., $\mathbf{O(n^2)}$) could slow down the user experience.
- Your checker evaluates password criteria (length, character diversity, penalties) in linear time **O(n)**, ensuring fast and responsive performance.

ALGORITHM COMPLEXITY

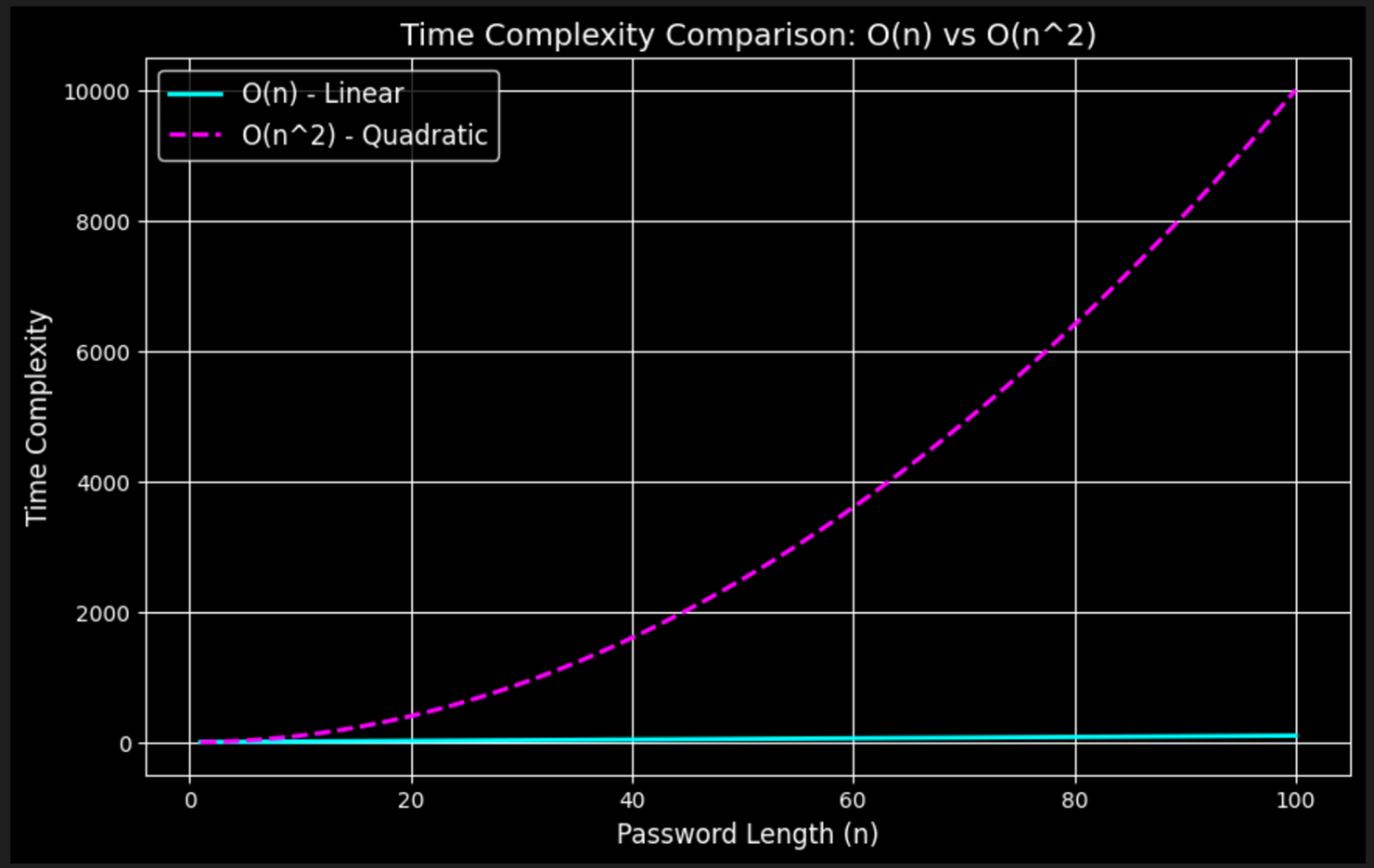
```
// Length Points
if (length >= 8 && length <= 12)
{
    lengthPoints = 2;
    points += 2;
}
```

Has **O(1)** time complexity

```
// Repetition Penalty Points
int repeats = 0;
for (int i = 1; i < length; i++) {
    if (password[i] == password[i - 1]) repeats++;
}
```

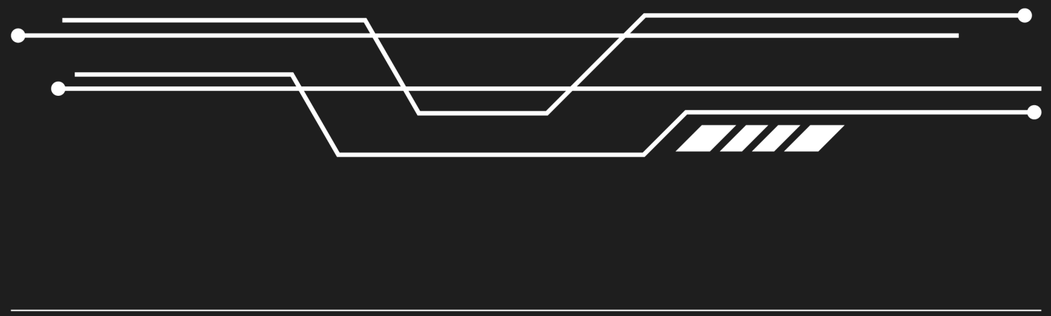
Has **O(n)** time complexity

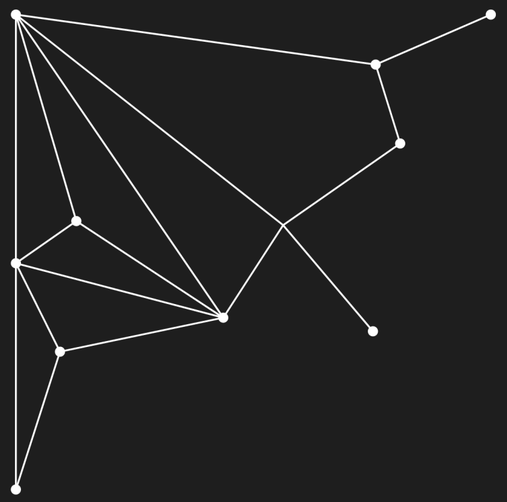
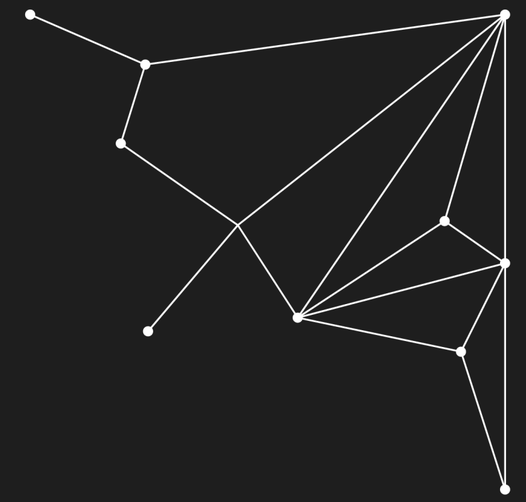
ALGORITHM COMPLEXITY



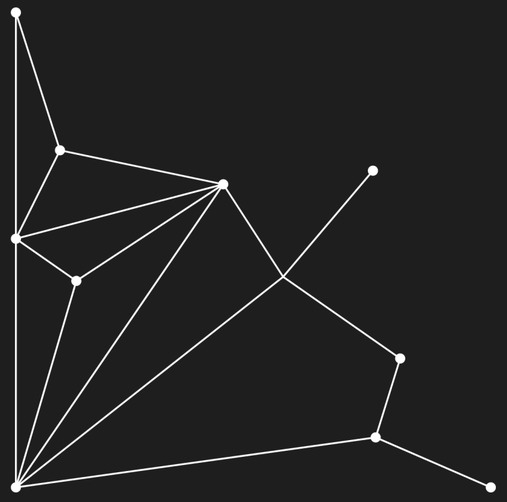
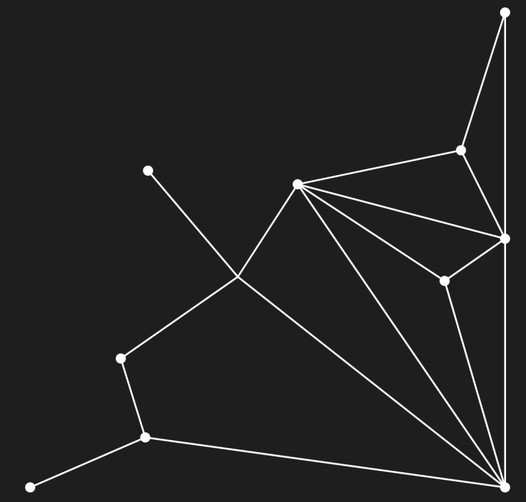
Here we can observe the difference between time complexities of an **O(n)** algorithm and an **O(n^2)** algorithm

CONCLUSION



QUESTIONS ?

AMMAR ITU

AMMAR ITU

THANK YOU FOR YOUR TIME