

NUMBER GUESSING GAME

Problem Statement:

This mini project involves creating a number guessing game. The user specifies a range $[X, Y]$, and the system randomly picks a number within this range. The goal is for the user to guess this number. To make the guessing process efficient, the system will employ binary search to guide the user toward the correct number, reducing the number of guesses and optimizing time complexity. The system will provide feedback on each guess, indicating whether the correct number is higher or lower. This approach ensures that the game is both challenging and efficient, as the binary search algorithm systematically narrows down the possible range of numbers, leading the user to the correct answer in the minimum number.

Algorithm:

An algorithm is a step-by-step procedure used for solving a problem or performing a task. In the context of the number guessing game described above, the algorithm involves the following steps:

- 1) Input Range: The user inputs two integers X and Y , defining the range $[X, Y]$.
- 2) Random Number Selection: The system randomly selects a number, N , within this range.
- 3) Guessing Process:
 - The user makes a guess.
 - The system provides feedback:
 - "Too high" if the guess is greater than N .
 - "Too low" if the guess is less than N .
 - "Correct" if the guess is equal to N .
 - The guessing process continues until the user guesses the correct number.
- 4) Binary Search Approach: The system should guide the user to use binary search:
 - Start with the middle of the range.
 - Adjust the range based on whether the guess is too high or too low.

Pseudo Code:

```
function guessNumber(X, Y):
```

```
    # Step 1: System selects a random number N in the range [X, Y]
```

```
    N = random(X, Y)
```

```
    # Step 2: Initialize the low and high pointers for binary search
```

```
    low = X
```

```
    high = Y
```

```
    attempts = 0
```

```
    while True:
```

```
        # Step 3: User makes a guess
```

```
        guess = (low + high) // 2 # User's guess using binary search
```

```
        attempts += 1
```

```
        # Step 4: System provides feedback
```

```
        if guess < N:
```

```
            print("Too low")
```

```
            low = guess + 1 # Adjust the range to [guess + 1, high]
```

```
        elif guess > N:
```

```
            print("Too high")
```

```
            high = guess - 1 # Adjust the range to [low, guess - 1]
```

```
        else:
```

```
            print("Correct! You guessed the number in", attempts, "attempts.")
```

```
            break
```

Analysis:**Time Complexity:**

The binary search approach ensures that each guess cuts the remaining range in half. Thus, the time complexity is $O(\log_2(Y - X + 1))$. This is the optimal approach to minimize the number of guesses.

Space Complexity:

The space complexity is $O(1)$ since we are only using a constant amount of additional space for variables like low, high, guess, and attempts.

Python Code Implementation:

```
import random
```

```
def guess_number_game():
```

```
    # Step 1: User inputs the range
```

```
    X = int(input("Enter the starting range (X): "))
```

```
    Y = int(input("Enter the ending range (Y): "))
```

```
    if X > Y:
```

```
        print("Invalid range. Starting range (X) should be less than or equal to ending range (Y).")
```

```
        return
```

```
    # Step 2: System selects a random number in the range [X, Y]
```

```
    N = random.randint(X, Y)
```

```
    low = X
```

```
    high = Y
```

```
    attempts = 0
```

```
print(f'Guess the number between {X} and {Y}.')
```

```
while True:
```

```
    # Step 3: User makes a guess (using binary search technique)
```

```
    guess = int(input("guess:"))
```

```
    attempts += 1
```

```
    print(f'Your guess: {guess}')
```

```
    # Step 4: System provides feedback
```

```
    if guess < N:
```

```
        print(f'Too low, guess next number between {guess+1} and {high}')
```

```
        low = guess + 1
```

```
    elif guess > N:
```

```
        print(f'Too high, guess next number between {low} and {guess-1}')
```

```
        high = guess - 1
```

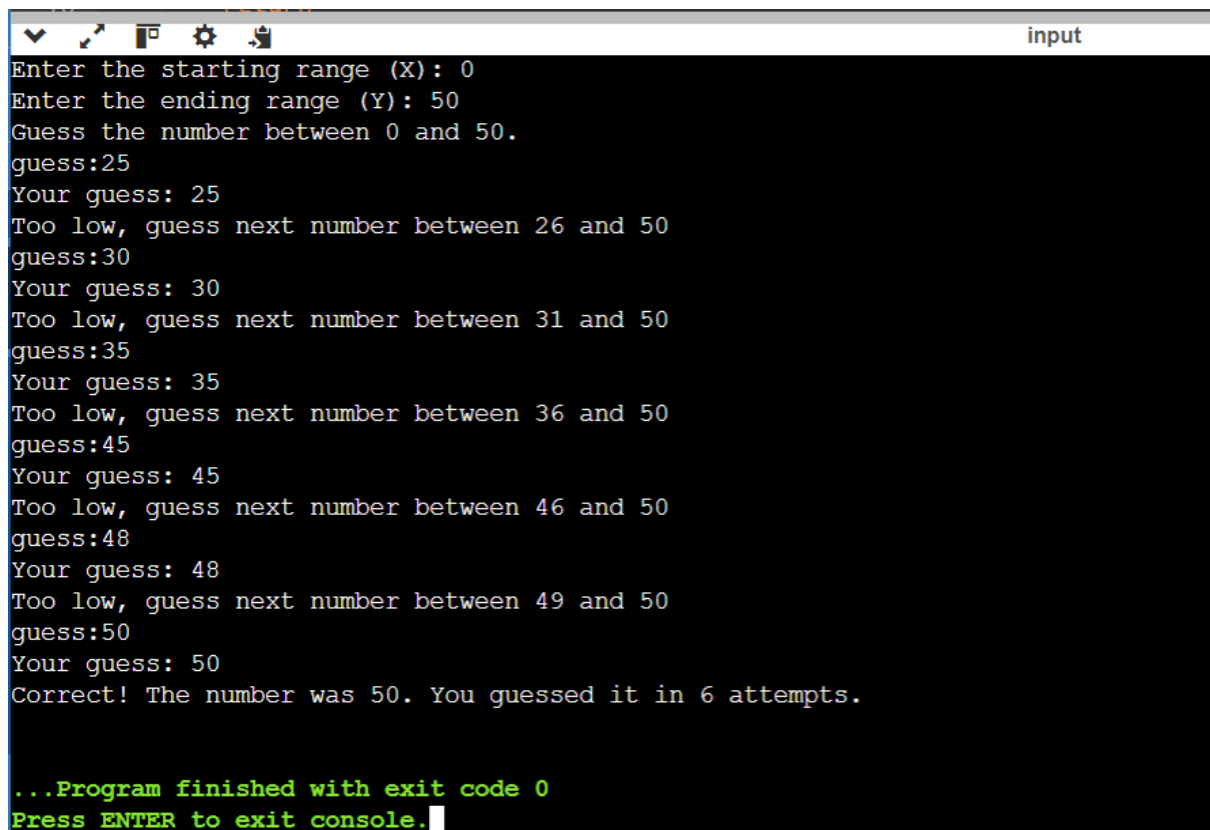
```
    else:
```

```
        print(f'Correct! The number was {N}. You guessed it in {attempts} attempts.')
```

```
        break
```

```
# Run the game
```

```
guess_number_game()
```

Output:A screenshot of a terminal window with a dark background and light-colored text. The window has a title bar at the top with several icons on the left and the word "input" on the right. The text in the terminal shows a sequence of prompts and user input for a number guessing game. The game starts with a range of 0 to 50. The user makes six guesses: 25, 30, 35, 45, 48, and 50. Each guess except the last is marked as "Too low" with a new range provided. The final guess of 50 is correct, and the program ends with a message indicating it finished with exit code 0 and a prompt to press ENTER to exit the console.

```
Enter the starting range (X): 0
Enter the ending range (Y): 50
Guess the number between 0 and 50.
guess:25
Your guess: 25
Too low, guess next number between 26 and 50
guess:30
Your guess: 30
Too low, guess next number between 31 and 50
guess:35
Your guess: 35
Too low, guess next number between 36 and 50
guess:45
Your guess: 45
Too low, guess next number between 46 and 50
guess:48
Your guess: 48
Too low, guess next number between 49 and 50
guess:50
Your guess: 50
Correct! The number was 50. You guessed it in 6 attempts.

...Program finished with exit code 0
Press ENTER to exit console.
```