ABSTRACT

One of the simplest two-player games is “Guess the number”. The first player thinks of a secret number in some known range while the second player attempts to guess the number. After each guess, the first player answers either “Higher”, “Lower” or “Correct!” depending on whether the secret number is higher, lower or equal to the guess. In this project, you will build a simple interactive program in Python where the computer will take the role of the first player while you play as the second player.

You will interact with your program using an input field and several buttons. For this project, we will ignore the canvas and print the computer's responses in the console. Building an initial version of your project that prints information in the console is a development strategy that you should use in later projects as well. Focusing on getting the logic of the program correct before trying to make it display the information in some “nice” way on the canvas usually saves lots of time since debugging logic errors in graphical output can be tricky.

One player thinks of a number from 1 to n, and the other player tries to guess at the number repeatedly. The first player tells the second if the guess is too high or too low, until the second player guesses correctly. A cost is associated with each verdict of “Too High” or “Too Low”.

The Number Guessing Game not only offers entertainment but also helps in developing logical thinking and problem-solving skills.

This feedback mechanism aids the player in narrowing down the possibilities. The game can be implemented in various programming languages and can range from basic text-based interfaces to more complex graphical user interfaces (GUI). Enhancements such as adjustable difficulty levels, scoring systems, and multiplayer modes can further enrich the gaming experience.

CHAPTER 1:

INTRODUCTION

- Analysis and Design of Algorithms (ADA) is a fundamental discipline within computer science that focuses on the study of efficient computational solutions to problems. Algorithms serve as the backbone of computing, enabling the manipulation and processing of data to achieve desired outcomes across various domains. The field of DAA encompasses the design, implementation, analysis, and optimization of algorithms to solve computational problems efficiently.
- The study of algorithms is a cornerstone of computer science, encompassing the design, analysis, and implementation of computational processes to solve problems efficiently. Algorithms are step-by-step procedures or formulas for solving problems, and their design and analysis are crucial for developing software that is both effective and efficient.
- The importance of algorithm analysis and design lies in the following:
    - Efficiency: Efficient algorithms make better use of resources, enabling applications to run faster and handle larger datasets.
    - Scalability: Well-designed algorithms can scale with the increasing size of input data, which is crucial in handling big data and high-performance computing tasks.
    - Optimization: Understanding the trade-offs between different algorithms helps in selecting the most appropriate one for a given context, leading to optimized performance.
    - Innovation: Algorithm design fosters innovation, allowing for the development of new and improved methods for solving complex problems.
- In summary, the analysis and design of algorithms are integral to advancing technology and solving real-world problems effectively. They provide the theoretical foundation and practical tools necessary for developing software that meets the demands of modern computing.

CHAPTER 2:

PROBLEM STATEMENT

- ✓ Build a Number guessing game, in which the user selects a range.
- ✓ Let’s say User selected a range, i.e., from A to B, where A and B belong to Integer.
- ✓ Some random integer will be selected by the system and the user has to guess that integer in the minimum number of guesses

CHAPTER 3:

IMPLEMENTATION

Analysis:

1) **Random Number Generation:**

   Typically implemented using a pseudo-random number generator (PRNG).

   For example, the random.randint(a, b) function can generate a random integer between a and b inclusive.

2) **User Input Handling:**

   Validating user input to ensure it is an integer within the specified range.

   Handling non-integer inputs gracefully to avoid crashes.

3) **Feedback Mechanism:**

   Simple conditional checks to compare the guessed number with the target number and provide appropriate feedback.

4) **Minimum Number of guesses:**

   The minimum number of guesses depends upon range. And the compiler must calculate the minimum number of guessing depends upon the range, on its own.

   For this, we have a formula:-

   Minimum number of guessing = log2 (Upper bound – lower bound + 1)

Algorithm:

Number Guessing Game works based on the **"Divide & Conquer"** algorithm.

Step - 1: User inputs the lower bound and upper bound of the range.

Step - 2: The compiler generates a random integer between the range and store it in a variable for future references.

Step - 3: For repetitive guessing, a while loop will be initialized.

Step - 4: If the user guessed a number which is greater than a randomly selected number, the user gets an output "Try Again! You guessed too high"

Step - 5: Else If the user guessed a number which is smaller than a randomly selected number, the user gets an output "Try Again! You guessed too small"

Step - 6: And if the user guessed in a minimum number of guesses, the user gets a "Congratulations!" Output.

Step - 7: Else if the user didn't guess the integer in the minimum number of guesses, he/she will get "Better Luck Next Time!" output.

Program:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    int lower, upper, x, guess, count = 0, flag = 0;
    int total_chances;
    // Taking Inputs
    printf("Enter Lower bound: ");
    scanf("%d", &lower);
    // Taking Inputs
    printf("Enter Upper bound: ");
    scanf("%d", &upper);
    // Seed the random number generator
    srand(time(0));
    // Generating random number between the lower and upper
    x = (rand() % (upper - lower + 1)) + lower;
    total_chances = (int)ceil(log(upper - lower + 1) / log(2));
    printf("\n\tYou've only %d chances to guess the " "integer!\n\n",total_chances);
    // for calculation of minimum number of guesses depends
    // upon range
    while (count < total_chances)
    {
        count++;
        // Taking guessing number as input
        printf("Guess a number: ");
        scanf("%d", &guess);
        // Condition testing
        if (x == guess)
        {
            printf("Congratulations you did it in %d try!\n",count);
            // Once guessed, loop will break
            flag = 1;
            break;
        }
        else if (x > guess)
        {
            printf("You guessed too small!\n");
        }
        else if (x < guess)
        {
            printf("You guessed too high!\n");
        }
    }
    // If Guessing is more than required guesses, shows this
    // output.
    if (!flag)
    {
        printf("\nThe number is %d\n", x);
        printf("\tBetter Luck Next time!\n");
    }
    return 0;
}
```

Time Complexity:

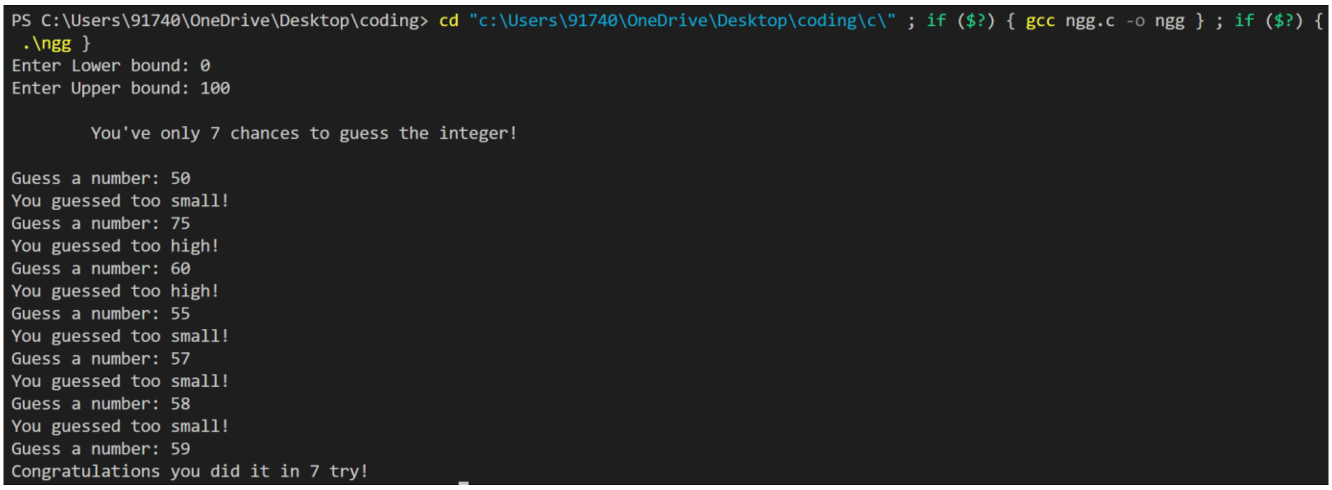
The time complexity of this code is **O(log2n)** where n is the difference between lower and upper bound of the range.

Space Complexity:

The space complexity of this code is **O(1)** as all the variables used in this code are of the same size and no extra space is required to store the values.

CHAPTER 4:

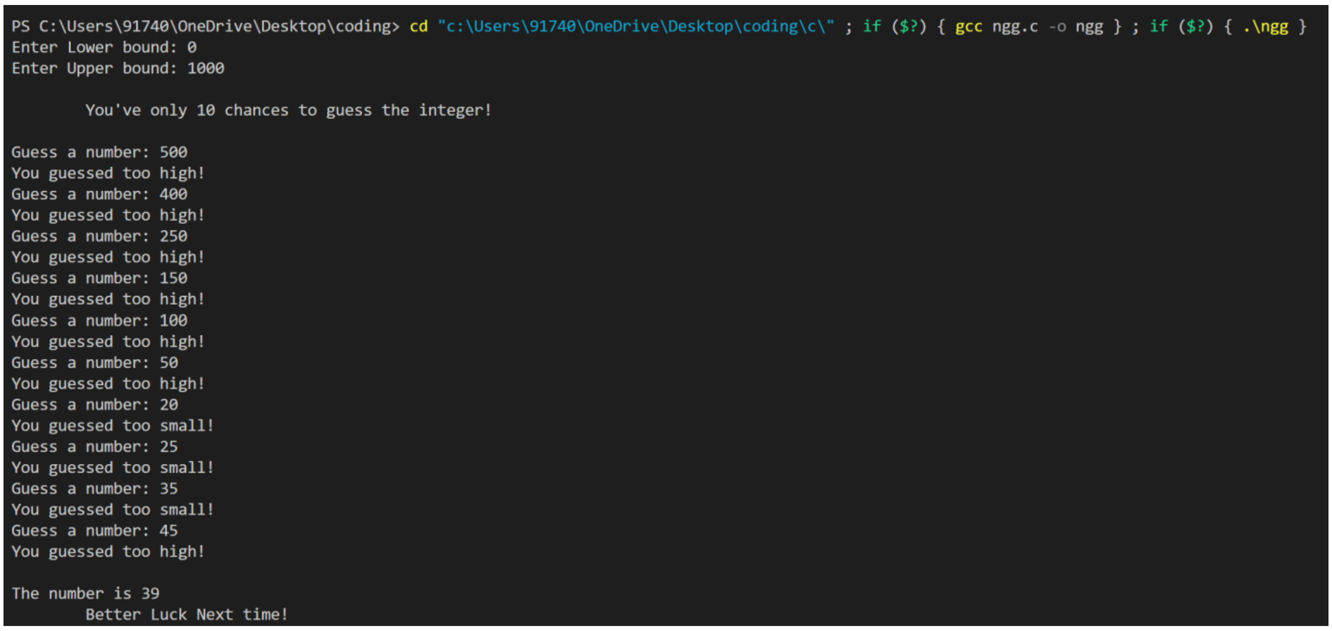
RESULTS



```
PS C:\Users\91740\OneDrive\Desktop\coding> cd "c:\Users\91740\OneDrive\Desktop\coding\c\" ; if ($?) { gcc ngg.c -o ngg } ; if ($?) {
 .\ngg }
Enter Lower bound: 0
Enter Upper bound: 100

        You've only 7 chances to guess the integer!

Guess a number: 50
You guessed too small!
Guess a number: 75
You guessed too high!
Guess a number: 60
You guessed too high!
Guess a number: 55
You guessed too small!
Guess a number: 57
You guessed too small!
Guess a number: 58
You guessed too small!
Guess a number: 59
Congratulations you did it in 7 try!
```

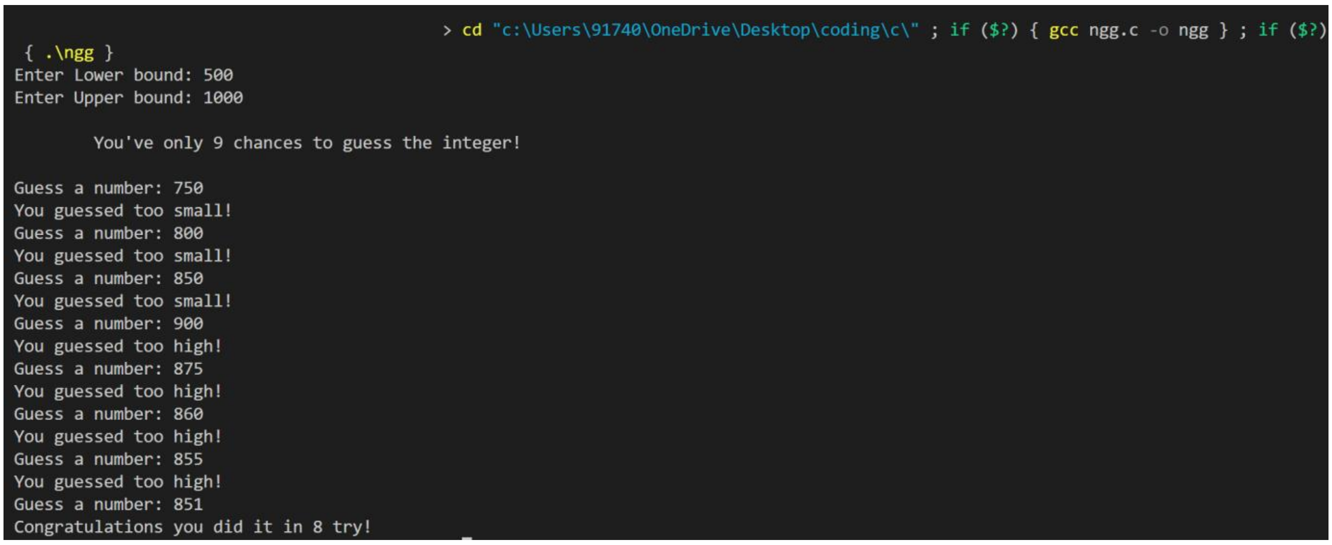


```
PS C:\Users\91740\OneDrive\Desktop\coding> cd "c:\Users\91740\OneDrive\Desktop\coding\c\" ; if ($?) { gcc ngg.c -o ngg } ; if ($?) { .\ngg }
Enter Lower bound: 0
Enter Upper bound: 1000

        You've only 10 chances to guess the integer!

Guess a number: 500
You guessed too high!
Guess a number: 400
You guessed too high!
Guess a number: 250
You guessed too high!
Guess a number: 150
You guessed too high!
Guess a number: 100
You guessed too high!
Guess a number: 50
You guessed too high!
Guess a number: 20
You guessed too small!
Guess a number: 25
You guessed too small!
Guess a number: 35
You guessed too small!
Guess a number: 45
You guessed too high!

The number is 39
        Better Luck Next time!
```



```
                                              > cd "c:\Users\91740\OneDrive\Desktop\coding\c\" ; if ($?) { gcc ngg.c -o ngg } ; if ($?)
 { .\ngg }
Enter Lower bound: 500
Enter Upper bound: 1000

        You've only 9 chances to guess the integer!

Guess a number: 750
You guessed too small!
Guess a number: 800
You guessed too small!
Guess a number: 850
You guessed too small!
Guess a number: 900
You guessed too high!
Guess a number: 875
You guessed too high!
Guess a number: 860
You guessed too high!
Guess a number: 855
You guessed too high!
Guess a number: 851
Congratulations you did it in 8 try!
```

CHAPTER 5:

CONCLUSION

- ➢ Although, we created a simple number guessing game in C we got to learn a lot of things while building this project.
    - Firstly, we learned about Binary Search and Divide & Conquer Upper and Lower Bound, followed by the use of while loop in C.
    - Finally, we got to brainstorm and as a result, we understood the thought process while building a project from scratch.
- ➢ The number guessing game in C is a simple game, where the computer generates a random number between upper bound and lower bound and the user has to guess that number in the minimum number of turns.
- ➢ After every try, the computer will tell us whether our guess was smaller, larger, or equal to the random computer-generated number and we can try again until we get the correct guess.
- ➢ The implementation of this game in C makes use of the following:
    - rand() function in C
    - Do while loop
    - If else statements
- ✓ The strategies to solve the game include:
    - Linear Search
    - Binary Search
    - Divide and Conquer

REFERENCES:

- ✓ GeeksforGeeks:https://www.geeksforgeeks.org/number-guessing-game-in-python/