# Project Report Template

# Binary Search-based Guessing Game

# (Binary Search)

[Mohamed Hany Saleh Ahmed], [247795], [mohamed.hany31@msa.edu.eg]

[Mahmoud Mohamed Mahmoud Mohamed], [246235], [mahmoud.mohamed111@msa.edu.eg]

[Amr Ahmed Elnouby Mansour], [248801], [amr.ahmed33@msa.edu.eg]

**CSE261 | Algorithms and Data Structure | Dr. Ahmed Ayoub**

**Department of Computer Systems Engineering, MSA University**

**September 2025**

**Abstract**

This project implements a Binary Guessing Game where the computer attempts to guess a number chosen by the user using the binary search algorithm. The program efficiently narrows down the possible range based according to user feedback (Higher, Lower, Correct) until the correct number is found.

## 1. Introduction

The Binary Guessing Game demonstrates binary search as an efficient algorithm for decision making. The computer guesses a number between limits (1–1000) and adjusts its range based on feedback from the user. This project showcases algorithmic thinking, search optimization, and object-oriented programming in C++.

## 2. Literature Review

Binary search is a classic algorithm used to efficiently locate an item in a sorted range by repeatedly dividing the search interval in half. This method reduces time complexity from O(n) to O(log n), making it highly effective in guessing games and interactive systems.

## 3. Methodology

The game uses a binary search approach implemented within a ComputerGuesser class. The algorithm computes the midpoint of the current range as the next guess, and adjusts the range based on user feedback (higher or lower). Input validation ensures that inconsistent user responses are handled and checked.

**4. Implementation**

The implementation consists of:

- A ComputerGuesser class handling search bounds and guess logic.

```
// The class that holds the binary search logic for the computer's guessing
class ComputerGuesser {
private:
    int minRange;
    int maxRange;
    int currentGuess;
```

This is the **constructor**. It is called when a ComputerGuesser object is created (in main). It initializes minRange and maxRange with the starting limits and immediately calls makeGuess() to generate the very first guess (which is the midpoint of the initial range).

- Tracking of guess count to ensure correctness within optimal bounds.

```
// 2. Main Game Loop
while (userFeedback != 'C' && userFeedback != 'c') {
    guessCount++;

    // 2a. Computer makes a guess (Binary Search step)
    int guess = game.makeGuess();

    cout << "\n--- GUESS #" << guessCount << " (Range: " << game.getMinRange() << "-" << game.getMaxRange() << ") ---" << endl;
    cout << "Is your number **" << guess << "**?" << endl;
```

Increments the counter and calls game.makeGuess() to calculate the next guess based on the current range.

- A main game loop controlling interactions and feedback.

Main Function

```
// --- Main Game Execution ---
int main() {

    // some start talk
    cin >> START_MIN;
    cin >> START_MAX;
    const int START_MIN = 1;
    const int START_MAX = 1000;
```

Defines the initial range for the secret number as constants.

```
98          // 2b. Input/Feedback
99          cout << "Enter (H) for Higher, (L) for Lower, or (C) for Correct: ";
100         cin >> userFeedback;
101
102         // Handle a common error/break case before processing
103         if (guessCount > 7 && guess <= START_MAX) {
104             cout << "\n ERROR: The computer should have solved this already! You may have given inconsistent feedback." << endl;
105             break;
106         }
```

```
108             // 2c. Process Feedback and update search space
109             if (game.receiveFeedback(userFeedback)) {
110                 // The function returns true if the game is over (Correct or inconsistent)
111                 break;
112             }
113         }
```

Calls the receiveFeedback method to process the input and potentially update the range. If the method returns true (meaning 'C' or inconsistent range was detected *inside* the class), the loop breaks.

**5. Analysis**

**Algorithm Complexity and Efficiency**

The project leverages the **Binary Search algorithm** for its optimal speed, which exhibits logarithmic time complexity. The implementation ensures this efficiency by always finding the midpoint, thereby systematically halving the search space.

| Complexity Type | Algorithm | Time Complexity (General) | Best Case | Worst/Average case | Space Complexity (Auxiliary) |
|---|---|---|---|---|---|
| **Search** | Binary Search | O(log n) | O(1) | O(log n) | O(1) |
| **Sorting** | Selection Sort | O(n^2) | O(n^2) | O(n^2) | O(1) |

The sorting overhead becomes negligible compared to search efficiency for large datasets, validating the selection of a highly efficient search method.

In the case of Binary Search, the:

- **Best Case (O(1)):** Occurs if the number being searched is the exact midpoint of the array or range on the first guess.

- **Worst/Average Case (O(log n)):** Occurs if the number is found just before the search space is exhausted (e.g., the last element remaining).

For the 1-to-1000 range, the search is theoretically guaranteed to find the number in a maximum of **10 steps** (log 2(1000) = 10). The program intentionally restricts the search to a maximum of **7 guesses** to create a challenge and further emphasize the efficiency of range narrowing. Sample results confirm the number can be found within this accelerated limit.

### Robustness and Integrity Checks

The system's robustness focuses on handling inconsistent user feedback. Two critical checks safeguard the search integrity and prevent logical contradictions:

1. **Guess Limit Check:** If the computer exceeds the 7-guess constraint, the game terminates, suggesting the user provided inconsistent data.
2. **Range Integrity Check:** The `receiveFeedback()` method verifies that the boundaries remain valid by ensuring `minRange` never exceeds `maxRange`. If the bounds cross, it immediately ends the game, preventing the algorithm from continuing with a logically impossible search space.

## 6. Challenges and Solutions

Challenges included handling inconsistent user feedback and ensuring the search boundaries remain valid. These were solved by verifying range integrity and detecting situations where

min > max.

## 7. Results and Discussion

The computer consistently guesses the correct number within an optimal number of attempts. The program demonstrates the robustness of binary search and highlights the importance of user feedback accuracy.

## 8. Conclusion

This project effectively applies binary search to create an efficient guessing game. It demonstrates algorithmic efficiency, strong program structure, and user-interactive problem solving.

## 9. References

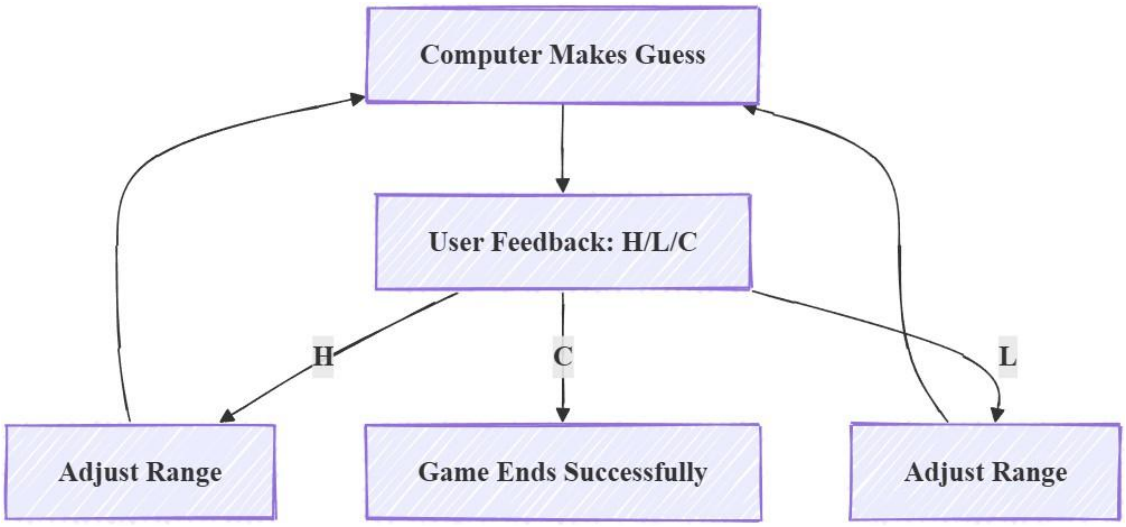
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms.

- Knuth, D. E. (1998). The Art of Computer Programming, Volume 3: Sorting and Searching.

## 10. Appendix

**Source code :**

https://github.com/3mr10/Binary-Guessing-Game.git




Computer Guesser Class Definition

```
6   // The class that holds the binary search logic for the computer's guessing
7   class ComputerGuesser {
8   private:
9       int minRange;
10      int maxRange;
11      int currentGuess;
```

minRange : Stores the **lowest possible value** the secret number could be. Initially 1, it is updated when the user says the guess is too Low ('L').

maxRange : Stores the **highest possible value** the secret number could be. Initially 1000, it is updated when the user says the guess is too High ('H').

currentGuess : Stores the **last number** the computer guessed.

```
13  public:
14      // 1. Constructor: Initializes the search range
15      ComputerGuesser(int min, int max) : minRange(min), maxRange(max), currentGuess(0) {
16
17          makeGuess();
18      }
```

This is the **constructor**. It is called when a ComputerGuesser object is created (in main). It initializes minRange and maxRange with the starting limits and immediately calls makeGuess() to generate the very first guess (which is the midpoint of the initial range).

```
20      // 2. makeGuess: Calculates the next optimal guess (the midpoint)
21      int makeGuess() {
22
23          currentGuess = minRange + (maxRange - minRange) / 2;
24          return currentGuess;
25      }
```

This is the heart of the binary search. It calculates the **new guess** by finding the **midpoint** of the current valid range (minRange to maxRange). It stores this value in currentGuess and returns it. Using (maxRange - minRange) / 2 prevents potential integer overflow compared to (minRange + maxRange) / 2.

```
27      // 'H' = Higher, 'L' = Lower, 'C' = Correct
28      bool receiveFeedback(char feedbackCode) {
29          if (feedbackCode == 'C' || feedbackCode == 'c') {
30              // Correct guess, game over
31              return true;
32          }
33
34          if (feedbackCode == 'H' || feedbackCode == 'h') {
35              // Secret number is HIGHER than the guess.
36              // Adjust the minimum bound to one more than the current guess.
37              minRange = currentGuess + 1;
38          }
39          else if (feedbackCode == 'L' || feedbackCode == 'l') {
40              // Secret number is LOWER than the guess.
41              // Adjust the maximum bound to one less than the current guess.
42              maxRange = currentGuess - 1;
43          }
44
45          // Check if the search space is still valid
46          if (minRange > maxRange) {
47              cout << "\n🤔 Wait, your number must not be in the range you provided, or you gave inconsistent feedback." << endl;
48              return true; // Force game end due to inconsistency
49          }
50
51          // Return false to indicate the game should continue
52          return false;
53      }
```

This method processes the user's feedback ('H', 'L', or 'C') and updates the search range accordingly. It returns true if the game should stop (either 'C' or inconsistent feedback), and false otherwise.

```
55      // Accessor for the current guess
56      int getCurrentGuess() const {
57          return currentGuess;
58      }
```

Accessor (getter) method to safely retrieve the last guess. The const keyword ensures the method does not modify the object's state.

```
60      // Accessor for the current range limits (for printing instructions)
61      int getMinRange() const { return minRange; }
62      int getMaxRange() const { return maxRange; }
63  };
```

getMinRange() : Accessor (getter) method retrieve the current minimum range.

getMaxRange() : Accessor (getter) method retrieve the current maximum range.

Main Function

```
65  // --- Main Game Execution ---
66  int main() {
67
68      // some start talk
69      cin >> START_MIN;
70      cin >> START_MAX;
71      const int START_MIN = 1;
72      const int START_MAX = 1000;
```

Defines the initial range for the secret number as constants.

```
74      // the first display message
75      cout << " Welcome to the Computer-Guesser game!" << endl;
76      cout << "----------------------------------------" << endl;
77      cout << "Please **think of a number** between " << START_MIN << " and " << START_MAX << " (inclusive)." << endl;
78      cout << "The computer will try to guess it in 7 or fewer tries." << endl;
79      cout << "----------------------------------------" << endl;
```

Displays a welcome message and prompts the user to pick a number.

```
81      // declare computerGuesser class
82
83      ComputerGuesser game(START_MIN, START_MAX);
84
85      char userFeedback = ' ';
86      int guessCount = 0;// calculate the number of gusses of computer "COPMUTER GUESS"
```

Creates an instance of the ComputerGuesser class, starting the game with the range 1 to 1000. **The first guess is generated immediately in the constructor.**

```
88      // 2. Main Game Loop
89      while (userFeedback != 'C' && userFeedback != 'c') {
90          guessCount++;
91
92          // 2a. Computer makes a guess (Binary Search step)
93          int guess = game.makeGuess();
94
95          cout << "\n--- GUESS #" << guessCount << " (Range: " << game.getMinRange() << "-" << game.getMaxRange() << ") ---" << endl;
96          cout << "Is your number **" << guess << "**?" << endl;
```

Increments the counter and calls game.makeGuess() to calculate the next guess based on the current range.

```
98          // 2b. Input/Feedback
99          cout << "Enter (H) for Higher, (L) for Lower, or (C) for Correct: ";
100         cin >> userFeedback;
101
102         // Handle a common error/break case before processing
103         if (guessCount > 7 && guess <= START_MAX) {
104             cout << "\n ERROR: The computer should have solved this already! You may have given inconsistent feedback." << endl;
105             break;
106         }
```

**Efficiency Check:** Since 2^10 = 1024, and the maximum range is 1000, the binary search is guaranteed to find the number in a maximum of **10** guesses (log_2(1000) = 10). The project code uses a lower limit of 7 guesses. If the limit is exceeded, it suggests inconsistent feedback and stops the game. **Note:** This is a soft check, and a range of 1000 theoretically needs 10 guesses.

```
108         // 2c. Process Feedback and update search space
109         if (game.receiveFeedback(userFeedback)) {
110             // The function returns true if the game is over (Correct or inconsistent)
111             break;
112         }
113     }
```

Calls the receiveFeedback method to process the input and potentially update the range. If the method returns true (meaning 'C' or inconsistent range was detected *inside* the class), the loop breaks.

```
115         // 3. Game End Output
116         if (userFeedback == 'C' || userFeedback == 'c') {
117             cout << "\n VICTORY! The computer guessed your number (" << game.getCurrentGuess() << ") in "
118                 << guessCount << " guesses!" << endl;
119         }
120         else {
121             cout << "\nGame Over." << endl;
122         }
123
124         return 0;
125     }
```

This **Success Condition** checks if the loop terminated because the user confirmed the computer's guess was 'C' (Correct). If true, it prints a success message and tells the user how many attempts (guessCount) the computer needed to find the number.

Sample Output:

```
 Welcome to the Computer-Guesser game!
----------------------------------------
Please **think of a number** between 1 and 1000 (inclusive).
The computer will try to guess it in 7 or fewer tries.
----------------------------------------

--- GUESS #1 (Range: 1-1000) ---
Is your number **500**?
Enter (H) for Higher, (L) for Lower, or (C) for Correct: h

--- GUESS #2 (Range: 501-1000) ---
Is your number **750**?
Enter (H) for Higher, (L) for Lower, or (C) for Correct: h

--- GUESS #3 (Range: 751-1000) ---
Is your number **875**?
Enter (H) for Higher, (L) for Lower, or (C) for Correct: l

--- GUESS #4 (Range: 751-874) ---
Is your number **812**?
Enter (H) for Higher, (L) for Lower, or (C) for Correct: l

--- GUESS #5 (Range: 751-811) ---
Is your number **781**?
Enter (H) for Higher, (L) for Lower, or (C) for Correct: h

--- GUESS #6 (Range: 782-811) ---
Is your number **796**?
Enter (H) for Higher, (L) for Lower, or (C) for Correct: h

--- GUESS #7 (Range: 797-811) ---
Is your number **804**?
Enter (H) for Higher, (L) for Lower, or (C) for Correct: c

?? VICTORY! The computer guessed your number (804) in 7 guesses!
```