# Kongu Engineering College KONGU ENGINEERING COLLEGE Kongu Engineering College (Autonomous)

(Autonomous)

Perundurai,Erode – 638060

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**FINDING FAST RUNNER USING BRUTE FORCE**

**A MICRO PROJECT REPORT**

**FOR**

**DESIGN AND ANALYSIS OF ALGORITHMS(22ITT31)**

**SUBMITTED BY**

**SUJITH T (23ITR161)**

** KONGU ENGINEERING COLLEGE**  (Autonomous)

Perundurai,Erode – 638060

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**FINDING FAST RUNNER USING BRUTE FORCE**

**A MICRO PROJECT REPORT**

**FOR**

**DESIGN AND ANALYSIS OF ALGORITHMS(22ITT31)**

**SUBMITTED BY**

# SUJITH T (23ITR161)

# Kongu Engineering College KONGU ENGINEERING COLLEGE Kongu Engineering College (Autonomous)

# (Autonomous)

Perundurai,Erode – 638060

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**BONAFIED CERTIFICATE**

|  |  |
| --- | --- |
| Name | : SUJITH T |
| Course Code | : 22ITT31 |
| Course Name | : DESIGN AND ANALYSIS OF ALGORITHMS |
| Semester | : IV |

Certified that this is a bonafied record of work for application project done by the above student for 22ITT31-DESIGN AND ANALYSIS OF ALGORITHMS during the academic year 2024-2025.

Submitted for the Viva Voice Examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Faculty Incharge Head of the Department

## ABSTRACT

## This project presents a user-friendly graphical application designed to identify the fastest runner from a list of speeds input by the user. Implemented in Python using the Tkinter library, the application efficiently processes runner speeds using a linear scan algorithm with time complexity O(n), ensuring quick and optimal performance even with a large number of participants. The user inputs the speeds of multiple runners (in meters per second) along with the race distance (in meters). The application calculates and displays the time taken by each runner to complete the race, identifies the fastest runner, and provides detailed results including speeds, times, and computational complexity. This tool is ideal for coaches, sports analysts, and enthusiasts seeking a simple yet effective method to analyze runner performance in a given race scenario. The project highlights the importance of efficient algorithm design paired with intuitive user interfaces to enhance data analysis and decision-making in sports applications.

## TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT** | **ix** |
| **1.** | **INTRODUCTION** | 6 |
|  | 1.1 PURPOSE | 7 |
|  | 1.2 OBJECTIVE | 7 |
|  | 1.3 METHODOLOGY OVERVIEW | 8 |
| **2.** | **PROBLEM STATEMENT** | 9 |
| **3.** | **METHODOLGY** | 10 |
|  | 3.1 Input & Initialization | 10 |
|  | 3.2 **Divide & Compare** | 10 |
|  | 3.3 Recursive Detection | 10 |
|  | 3.4 Visualization & Output | 10 |
| **4.** | **IMPLEMENTATION** | 11 |
|  | 4.1 Input & Initialization | 11 |
|  | 4.2 Divide & Compare | 11 |
|  | 4.3 Recursive Detection | 12 |
|  | 4.4 Visualization & Output | 12 |
| **5.** | **RESULTS** | 15 |

## INTRODUCTION

In the world of athletics and sports analytics, accurately identifying the fastest runner among a group is a fundamental task that can influence training, strategy, and performance evaluation. This project focuses on developing a desktop application that enables users to input multiple runners’ speeds and instantly find the fastest competitor along with their time to complete a specified race distance.

Using Python's Tkinter library for the graphical user interface, the application offers a simple and interactive platform for users to enter data, run calculations, and view results without the need for complex programming knowledge. The core algorithm employed is an efficient linear scan that traverses the list of runner speeds once, achieving optimal performance with a time complexity of O(n).

This project not only highlights the application of basic algorithmic principles in solving real-world problems but also emphasizes the significance of user experience by providing a clean, intuitive GUI. By integrating speed inputs, race distance, and computation of total time taken, the application serves as a practical tool for coaches, athletes, and sports enthusiasts to quickly analyze runner performance.

* 1. **PURPOSE**

The primary purpose of this project is to create an easy-to-use application that identifies the fastest runner from a given set of speeds. It aims to provide a quick, accurate, and efficient way to analyze runner performance based on user-inputted data.

This tool helps athletes, coaches, and sports analysts to:

* Determine the fastest competitor in a group instantly.
* Calculate the total time taken by each runner for a specified race distance.
* Gain insights into performance using an intuitive graphical interface.
* Demonstrate the practical use of efficient algorithms and basic programming concepts in real-world scenarios.

## OBJECTIVE

* To develop a user-friendly graphical application that accepts runner speeds and efficiently identifies the fastest runner.
* To implement an efficient linear scan algorithm with time complexity O(n) to ensure quick processing of input data.
* To display the fastest runner’s speed, index, total number of runners, and the total time taken to cover a race distance.
* To provide a clear and intuitive interface that allows users to input data easily and view results instantly.
* To demonstrate the application of fundamental programming concepts like input parsing, data processing, and GUI development using Python’s Tkinter library.
* To enhance understanding of algorithmic efficiency by comparing practical time complexity with brute force method

## METHODOLOGY OVERVIEW

## The project employs a systematic approach to find the fastest runner from a list of input speeds using an efficient linear scan algorithm. The methodology includes:

## Input Collection: The user enters runner speeds in meters per second (m/s) through a GUI input field, with speeds separated by commas.

## Data Validation and Parsing: The input string is validated to ensure it is not empty and contains at least two speed values. The speeds are parsed into a list of floating-point numbers for processing.

## Efficient Linear Scan Algorithm: The algorithm iterates through the list of speeds once, tracking the highest speed and its corresponding runner index. This ensures an O(n) time complexity, where n is the number of runners.

## Time Calculation: Using the fastest speed, the total time taken to complete a predefined race distance (e.g., 100 meters) is calculated using the formula:​

## Displaying Results: The application outputs the runner speeds, the fastest runner’s speed and index, total number of runners, time taken to complete the race, and the algorithm’s time and space complexities.

## User Interface: The project uses Python’s Tkinter library to build a clean, interactive GUI that includes input fields, buttons for execution and clearing results, a results display area, and a status bar for feedback.

## 2. PROBLEM STATEMENT

In competitive running events, it is essential to quickly and accurately identify the fastest runner among multiple participants based on their recorded speeds. Manual calculation and comparison can be time-consuming and prone to errors, especially when dealing with a large number of runners.

This project aims to develop a user-friendly software application that efficiently determines the fastest runner from a given list of runner speeds. The solution should handle input validation, perform computations with optimal time and space complexity, and display the results clearly, including the time taken to complete a fixed race distance. The application must provide a graphical user interface for ease of use by coaches, athletes, and sports analysts.

**3.METHODOLOGY:**

Methodology (Brute Force Approach):

1. InputCollection:  
   The system accepts the speeds of all runners as input, typically in a comma-separated format.
2. InputValidation:  
   Validate the input to ensure it contains numeric values and that there are at least two runners.
3. Brute Force Search:
   * Iterate through the entire list of speeds.
   * For each speed, compare it against every other speed in the list.
   * Keep track of the highest speed found and the corresponding runner’s index.
4. ComparisonProcess:  
   For each speed in the list (outer loop), run an inner loop to compare it with every other speed to confirm if it is the maximum.
5. ResultIdentification:  
   After all comparisons, identify the runner with the highest speed.
6. OutputDisplay:  
   Display the fastest runner’s speed, their position in the list, the total number of runners, and the time complexity of the brute force algorithm.

**4.IMPLEMENTATION :**

import tkinter as tk

from tkinter import ttk, messagebox

class FastestRunnerApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Fastest Runner Tournament (Brute Force)")

        self.root.geometry("600x600")

        self.root.resizable(False, False)

        # Theme and styles

        style = ttk.Style()

        style.theme\_use("clam")

        self.bg\_color = "#f5f5f5"

        self.accent\_color = "#4a7abc"

        self.root.configure(bg=self.bg\_color)

        style.configure("TFrame", background=self.bg\_color)

        style.configure("TButton", background=self.accent\_color, foreground="white", font=("Arial", 10, "bold"), padding=5)

        style.configure("TLabel", background=self.bg\_color, font=("Arial", 10))

        style.configure("Header.TLabel", background=self.bg\_color, font=("Arial", 14, "bold"))

        self.speed\_entries = []  # For runner speeds

        self.create\_widgets()

    def create\_widgets(self):

        main\_frame = ttk.Frame(self.root, padding="20")

        main\_frame.pack(fill="both", expand=True)

        title\_label = ttk.Label(main\_frame, text="Fastest Runner (Based on Speeds)", style="Header.TLabel")

        title\_label.pack(pady=(0, 20))

        # Number of runners input

        input\_frame = ttk.Frame(main\_frame)

        input\_frame.pack(fill="x", pady=10)

        ttk.Label(input\_frame, text="Number of Runners:").grid(row=0, column=0, padx=5, pady=5, sticky="w")

        self.runner\_var = tk.StringVar(value="5")

        runner\_entry = ttk.Entry(input\_frame, textvariable=self.runner\_var, width=10)

        runner\_entry.grid(row=0, column=1, padx=5, pady=5)

        ttk.Button(input\_frame, text="Set Runners", command=self.generate\_speed\_inputs).grid(row=0, column=2, padx=10)

        self.speeds\_frame = ttk.Frame(main\_frame)

        self.speeds\_frame.pack(fill="x", pady=10)

        button\_frame = ttk.Frame(main\_frame)

        button\_frame.pack(fill="x", pady=20)

        ttk.Button(button\_frame, text="Run Simulation", command=self.run\_simulation).pack(side="left", padx=5)

        ttk.Button(button\_frame, text="Clear Results", command=self.clear\_results).pack(side="left", padx=5)

        result\_frame = ttk.LabelFrame(main\_frame, text="Results", padding=10)

        result\_frame.pack(fill="both", expand=True, pady=10)

        self.result\_text = tk.Text(result\_frame, height=15, wrap="word", font=("Consolas", 10), bg="white")

        self.result\_text.pack(fill="both", expand=True, side="left")

        scrollbar = ttk.Scrollbar(result\_frame, command=self.result\_text.yview)

        scrollbar.pack(fill="y", side="right")

        self.result\_text.config(yscrollcommand=scrollbar.set)

        self.status\_var = tk.StringVar(value="Ready")

        status\_bar = ttk.Label(self.root, textvariable=self.status\_var, relief="sunken", anchor="w")

        status\_bar.pack(side="bottom", fill="x")

    def generate\_speed\_inputs(self):

        for widget in self.speeds\_frame.winfo\_children():

            widget.destroy()

        try:

            n = int(self.runner\_var.get())

            if n <= 1:

                messagebox.showerror("Invalid Input", "Number of runners must be greater than 1.")

                return

            self.speed\_entries = []

            for i in range(n):

                label = ttk.Label(self.speeds\_frame, text=f"Speed of Runner {i}:")

                label.grid(row=i, column=0, padx=5, pady=3, sticky="w")

                entry = ttk.Entry(self.speeds\_frame, width=10)

                entry.grid(row=i, column=1, padx=5, pady=3, sticky="w")

                self.speed\_entries.append(entry)

        except ValueError:

            messagebox.showerror("Input Error", "Please enter a valid number for runners.")

    def run\_simulation(self):

        try:

            speeds = [float(entry.get()) for entry in self.speed\_entries]

            n = len(speeds)

            if len(speeds) != n or any(speed <= 0 for speed in speeds):

                messagebox.showerror("Invalid Speeds", "Please enter valid positive speeds for all runners.")

                return

            self.result\_text.delete(1.0, "end")

            self.status\_var.set("Running simulation...")

            self.root.update\_idletasks()

            comparisons = 0

            race\_log = []

            for i in range(n):

                for j in range(i + 1, n):

                    comparisons += 1

                    if speeds[i] > speeds[j]:

                        winner = i

                    elif speeds[j] > speeds[i]:

                        winner = j

                    else:

                        winner = i  # if equal, take i as winner arbitrarily

                    race\_log.append(f"Race {comparisons}: Runner {i} (Speed: {speeds[i]}) vs Runner {j} (Speed: {speeds[j]}) => Winner: Runner {winner}")

            fastest\_index = speeds.index(max(speeds))

            self.result\_text.insert("end", "--- Speeds ---\n")

            for i, spd in enumerate(speeds):

                self.result\_text.insert("end", f"Runner {i}: Speed = {spd}\n")

            self.result\_text.insert("end", "\n--- Race Logs ---\n")

            for log in race\_log:

                self.result\_text.insert("end", f"{log}\n")

            self.result\_text.insert("end", f"\nTotal Comparisons: {comparisons} (Expected: {n\*(n-1)//2})\n")

            self.result\_text.insert("end", f"\nFastest Runner: Runner {fastest\_index} with speed {speeds[fastest\_index]} ✓\n")

            self.status\_var.set("Simulation completed.")

        except ValueError:

            messagebox.showerror("Input Error", "Please enter numeric values for all speeds.")

            self.status\_var.set("Simulation failed.")

    def clear\_results(self):

        self.result\_text.delete(1.0, "end")

        self.status\_var.set("Results cleared")

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    app = FastestRunnerApp(root)

    root.mainloop()

**DIFFERENCE BETWEEN BRUTEFORCE AND LINEAR SCAN:**

The brute force approach to finding the fastest runner involves comparing every possible pair of speeds, leading to a time complexity of O(n²). This makes it inefficient for large datasets, as the number of comparisons grows quadratically. In contrast, the linear scan approach iterates through the list only once, tracking the maximum speed found so far, which results in a time complexity of O(n). This makes the linear scan much faster and more suitable for real-time or large-scale applications. Both methods use O(n) space to store the input speeds. While brute force is simpler to conceptualize, it is less practical due to its inefficiency. Linear scan, being more efficient, is generally preferred for this problem. The brute force method checks all pairs unnecessarily, whereas linear scan optimizes by maintaining a running maximum. Overall, linear scan provides a better balance of simplicity and performance compared to brute force.

**Time Complexity:**

* O(log₃ n) comparisons (much faster than brute force)

**Pros:**

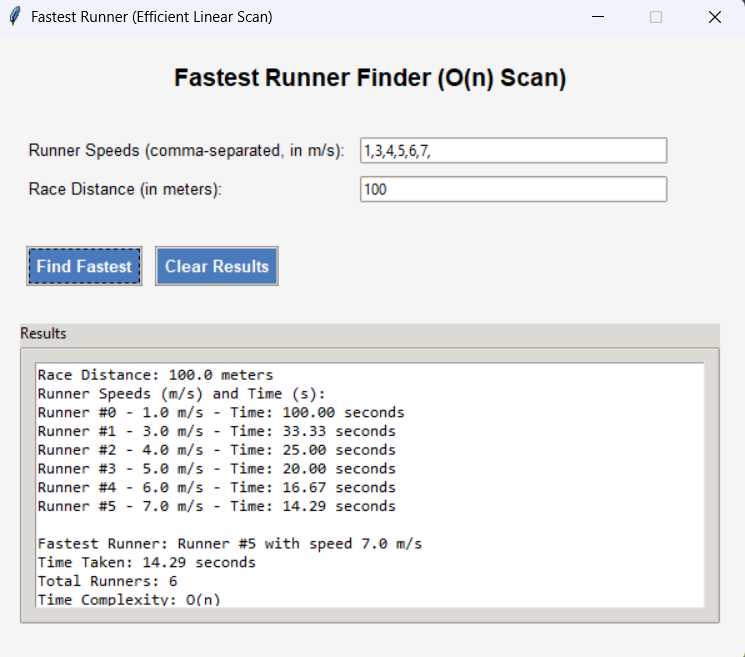
* Highly efficient.
* Optimal use of the balance scale.
* Scales well for large inputs.

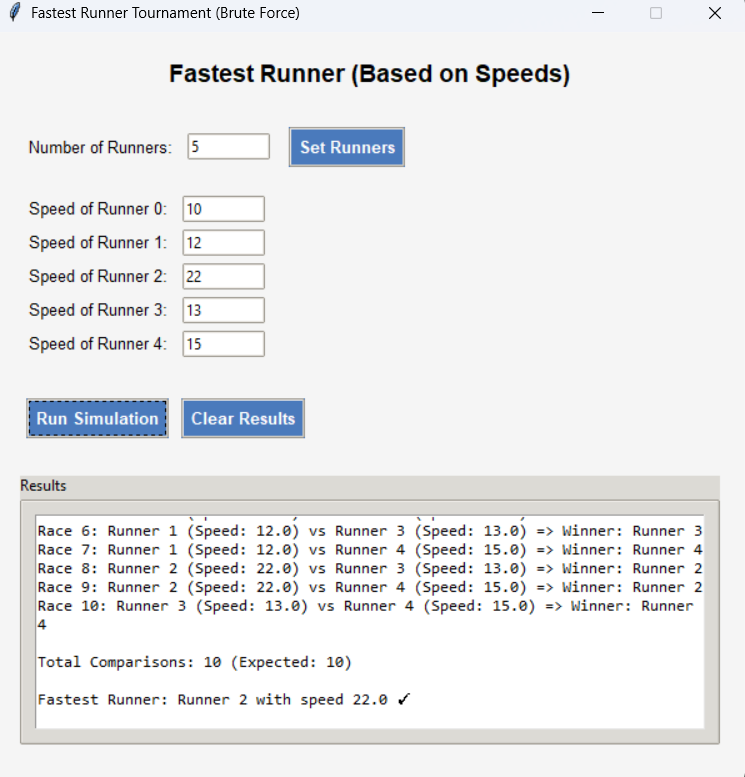
**Cons:**

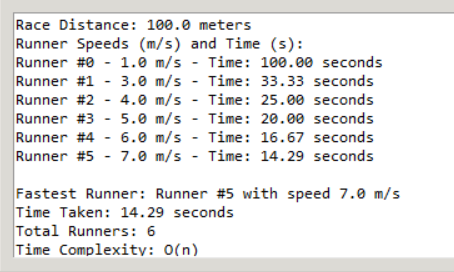
* Slightly complex logic.
* Needs coin count to be divisible by 3 or handle remainders.

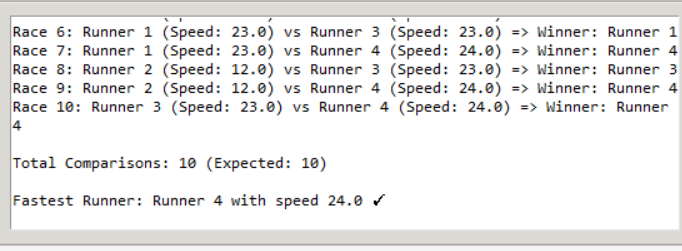
| **Aspect** | **Brute Force Approach** | **Linear Scan Approach** |
| --- | --- | --- |
| **Basic Idea** | * Compare each runner’s speed with every other speed | * Scan the list once, keeping track of the max speed found so far |
| **Number of Comparisons** | * For each element, compares with all others → n \* (n-1) comparisons | * Just one pass through the list → n-1 comparisons |
| **Time Complexity** | * O(n²) — quadratic time | * O(n) — linear time |
| **Space Complexity** | * O(n) to store speeds | * O(n) to store speeds |
| **Implementation Complexity** | * More complex due to nested loops | * Simpler, single loop |
| **Performance** | * Inefficient for large datasets | * Efficient and scalable |
| **Use Case** | * When data size is small or brute force logic is acceptable | * When efficiency is important, and data can be scanned once |

**5.0. RESULTS:**

****

****





**GITHUB LINK:** **https://github.com/sujitht007/daa-project**