

PART 1

PROOFS IN THE PRACTICAL SENSE

- WHEN ALGORITHMS FAIL PEOPLE: HEALTHCARE.GOV LAUNCH
- TODAY'S MISSION: PROVING OUR BINARY SEARCH

PART 2

CORRECTNESS PROOF

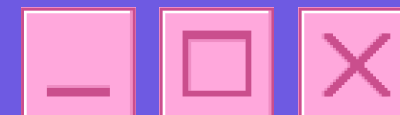
- THE “WHY” BEHIND THE CODE
- PROVING CORRECTING STEP BY STEP
- GROUP ACTIVITY: PROVE THE SCENARIO

PART 3

RUNTIME PROOF

- RECURRENCE RELATION
- CIVIC IMPACT OF EFFICIENCY
- GROUP ACTIVITY: PROVE THE SCENARIO

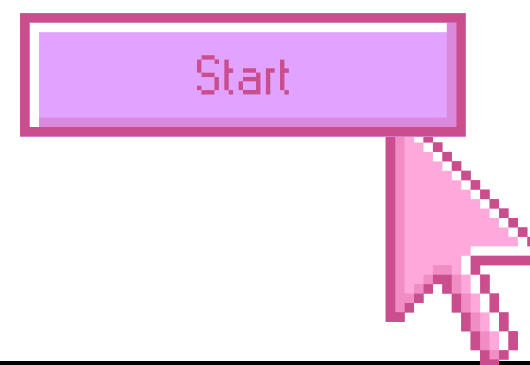
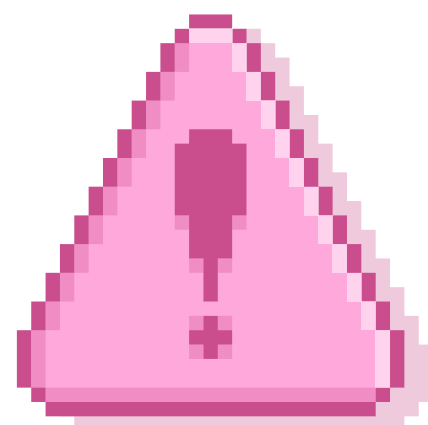




Welcome to

HEALTHCARE.GOV

WHY PROOFS MATTER



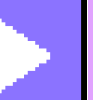
On October 1st, 2013, the Healthcare.gov launch failed due to inefficient algorithms, lack of testing, insufficient server capacity, and poor project management. Millions were hoping to enroll, but only 6 people were successful in receiving the correct plan. Users experienced problems with "wrong" insurance enrollments, like mistakes in application data and files being sent to the incorrect insurer. These issues stemmed from errors where enrollment information was duplicated or sent to the wrong insurance company.

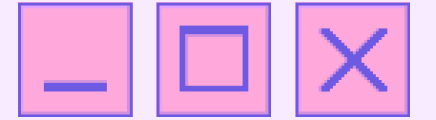
Event → Healthcare.gov launch 2013

Algorithms → correct bugs

Result → Thousands to millions of people got wrong insurance plans or no insurance

Impact → People couldn't access affordable healthcare they paid for





The Goal: Help millions of Americans find and enroll in health insurance.

The Reality: On day one, only 6 people succeeded. What Went Wrong?

Enrollment data was sent to the wrong insurance companies. People were matched with incorrect or no insurance plans. The system couldn't handle the scale.

- A bug in sorting = wrong order in resource allocation
- A flaw in search = people can't find emergency services
- In civic tech, algorithm errors hurt real people



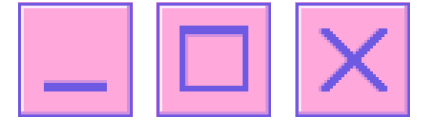


CORRECTNESS & RUNTIME

1. Formal Proof of Correctness
Step-by-Step proof for Binary Search and Merge Sort
2. Solving Recurrence Relations
A detailed look into Three Methods (Recursion Tree, Repeated Substitution, and the Master Theorem for both $T(n) = T(n/2) + c$ and $T(n) = 2T(n/2) + cn$)
3. Asymptotic Conclusion
Rigorously concluding $O(\log n)$ and $O(n \log n)$



Binary Search Skeleton Code

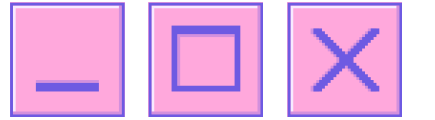
[Home](#)[Part 2](#)[Contact](#)

```
def binary_search(arr, target):
    # Initialize search boundaries
    low = ____
    high = ____

    # Continue searching while the search space is
    valid
    while ____ <= ____:
        # Find the middle index
        mid = ____
        # Check if we found the target
        if arr[mid] == ____:
            return ____
        # If target is in the right half
        elif arr[____] < ____:
            low = ____
        # If target is in the left half
        else:
            high = ____
    # Target not found
    return ____
```



Binary Search Algorithm Solution

[Home](#)[Part 2](#)[Contact](#)

```
def binary_search(arr, target):  
    low = 0  
    high = len(arr) - 1  
  
    while low <= high:  
        mid = (low + high) // 2  
        if arr[mid] == target:  
            return mid  
        elif arr[mid] < target:  
            low = mid + 1  
        else:  
            high = mid - 1  
    return -1
```

