1: Scalability Issue

```
#include <stdio.h>

#define MAXUSERS 500000

int main() {
    int users;

    printf("Enter the number of  users: ");
    scanf("%d", &users);

    if (users > MAXUSERS) {
        printf("Platform Crash: Too many users!\n", MAXUSERS);
    } else {
        printf("Platform is running smoothly with %d  users.\n", users);
    }

    return 0;
}
```

2: Recommendation Algorithm Failure

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define FAILURE_PROBABILITY 0.02

int main() {
    int failed_recommendations = 0;

    srand(time(0)); // Seed the random number generator

    for (int i = 0; i < 100; i++) {
        if ((rand() / (double)RAND_MAX) < FAILURE_PROBABILITY) {
            failed_recommendations++;
        }
    }

    printf("Total Recommendations: %d\n", TOTAL_RECOMMENDATIONS);
    printf("Failed Recommendations: %d\n", failed_recommendations);
```

```
    return 0;
}
```

3: Inventory Optimization

```
#include <stdio.h>
#include <limits.h>

#define WAREHOUSES 10
#define MAX_ITEMS 1000

int min(int a, int b) {
    return a < b ? a : b;
}

void optimizeInventory(int capacities[], int items[], int n) {
    int dp[WAREHOUSES + 1][MAX_ITEMS + 1];

    // Initialize DP table
    for (int i = 0; i <= WAREHOUSES; i++) {
        for (int j = 0; j <= MAX_ITEMS; j++) {
            if (i == 0) {
                dp[i][j] = INT_MAX;
            } else if (j == 0) {
                dp[i][j] = 0;
            } else {
                dp[i][j] = dp[i - 1][j];
                if (capacities[i - 1] <= j) {
                    dp[i][j] = min(dp[i][j], items[i - 1] + dp[i][j - capacities[i - 1]]);
                }
            }
        }
    }

    // Optimal allocation
    int result = dp[WAREHOUSES][MAX_ITEMS];
    if (result == INT_MAX) {
        printf("No feasible allocation\n");
    } else {
        printf("Optimal inventory allocation: %d items\n", result);
    }
```

```
}

int main() {
    int capacities[WAREHOUSES] = {100, 200, 150, 300, 250, 350, 400, 450, 500, 600};
    int items[WAREHOUSES] = {10, 20, 15, 30, 25, 35, 40, 45, 50, 60};

    optimizeInventory(capacities, items, WAREHOUSES);

    return 0;
}
```

6: Order Fulfillment Optimization

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define STAGES 5

void simulateOrderFulfillment(int stages[], int optimized_stages[]) {
    int total_time = 0, optimized_time = 0;

    printf("Stage Times: \n");
    for (int i = 0; i < STAGES; i++) {
        printf("Stage %d: %d seconds\n", i + 1, stages[i]);
        total_time += stages[i];
    }

    printf("\nOptimized Stage Times: \n");
    for (int i = 0; i < STAGES; i++) {
        optimized_stages[i] = stages[i] - (rand() % (stages[i] / 2));
        printf("Stage %d: %d seconds\n", i + 1, optimized_stages[i]);
        optimized_time += optimized_stages[i];
    }

    printf("\n Total Time: %d seconds\n", total_time);
    printf("Optimized Time: %d seconds\n", optimized_time);
}

int main()
 {
```

```
    srand(time(0));

    int stages[STAGES] = {60, 120, 90, 180, 300};

    int optimized_stages[STAGES];

    simulateOrderFulfillment(stages, optimized_stages);

    return 0;
}
```