```python
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm


def median_of_three(arr, idx1, idx2, idx3):
    """Find the median of three elements by index"""
    a, b, c = arr[idx1], arr[idx2], arr[idx3]
    if (a <= b <= c) or (c <= b <= a):
        return idx2
    elif (b <= a <= c) or (c <= a <= b):
        return idx1
    else:
        return idx3


def get_pivot_position_median_of_three(n):
    """
    Simulates selecting three random elements from an array of size n
    and returns the position of the median-of-three in the sorted array.
    """
    # Create a sorted array [0, 1, 2, ..., n-1]
    arr = np.arange(n)

    # Randomly select three distinct indices
    indices = np.random.choice(n, size=3, replace=False)

    # Find the median of the three elements
    median_idx = median_of_three(arr, indices[0], indices[1], indices[2])

    # Return the position (which is the value itself in a sorted array)
    return arr[median_idx]


def is_acceptable_split(position, n, a):
    """Check if position gives at worst an a-to-(1-a) split"""
    return a * n <= position <= (1 - a) * n


def estimate_probability(n, a, num_trials=100000):
    """
    Estimate the probability of getting at worst an a-to-(1-a) split
    using median-of-three pivot selection
    """
    acceptable_splits = 0

    for _ in range(num_trials):
        pivot_position = get_pivot_position_median_of_three(n)
        if is_acceptable_split(pivot_position, n, a):
            acceptable_splits += 1

    return acceptable_splits / num_trials

```

```python

def theoretical_probability_correct(a, n):
    """
    Calculate the theoretical probability based on proper integration of the sum.
    For large n, the probability that the median of three randomly chosen elements
    falls between an and (1-a)n can be calculated as:

    Sum_{m=ceil(an)}^{floor((1-a)n)} ((m-1)(n-m)) / binom(n,3)

    Which can be approximated by integration for large n.
    """
    # For numerical stability with large n
    if n > 100:
        # Integrate (x-1)(n-x) from an to (1-a)n and divide by binom(n,3)
        # First calculate the indefinite integral: ∫(x-1)(n-x)dx = nx^2/2 - x^3/3 - nx + x^2/2
        def integral(x):
            return (n * x ** 2) / 2 - x ** 3 / 3 - n * x + x ** 2 / 2

        result = (integral((1 - a) * n) - integral(a * n)) / (n * (n - 1) * (n - 2) / 6)
        return max(0, min(1, result))  # Ensure result is between 0 and 1
    else:
        # Direct calculation for small n
        total = 0
        for m in range(int(np.ceil(a * n)), int(np.floor((1 - a) * n)) + 1):
            total += (m - 1) * (n - m)
        return total / (n * (n - 1) * (n - 2) / 6)


def compare_empirical_vs_theoretical():
    """Compare empirical results with theoretical formula for various values of a"""
    n = 1000  # Size of array
    a_values = np.linspace(0.01, 0.49, 20)  # Values of a to test

    empirical_probs = []
    theoretical_probs = []

    print("Comparing empirical vs theoretical probabilities:")
    print("a\tEmpirical\tTheoretical\tDifference")
    print("-" * 70)

    for a in tqdm(a_values):
        empirical = estimate_probability(n, a, num_trials=10000)
        theoretical = theoretical_probability_correct(a, n)

        empirical_probs.append(empirical)
        theoretical_probs.append(theoretical)

        print(f"{a:.2f}\t{empirical:.6f}\t{theoretical:.6f}\t{abs(empirical - theoretical):.6f}")

    # Plot the results
    plt.figure(figsize=(12, 7))
```

```
    plt.plot(a_values, empirical_probs, 'bo-', label='Empirical')
    plt.plot(a_values, theoretical_probs, 'g-', label='Theoretical (Correct)')
    plt.xlabel('a (imbalance parameter)')
    plt.ylabel('Probability of at worst a-to-(1-a) split')
    plt.title('Median-of-Three Pivot Selection: Probability of Balanced Splits')
    plt.legend()
    plt.grid(True)

    # Add annotations for specific a values of interest
    for a in [0.1, 0.2, 0.3, 0.4]:
        theoretical = theoretical_probability_correct(a, n)
        plt.annotate(f'a={a}: P≈{theoretical:.4f}',
                     xy=(a, theoretical),
                     xytext=(a + 0.03, theoretical + 0.05),
                     arrowprops=dict(arrowstyle='->'))

    plt.savefig('median_of_three_splits.png')
    print("\nGraph saved as 'median_of_three_splits.png'")
    plt.show()


def pivot_position_distribution(n=1000, num_trials=100000):
    """Generate and visualize the distribution of pivot positions"""
    positions = []

    for _ in tqdm(range(num_trials)):
        positions.append(get_pivot_position_median_of_three(n))

    plt.figure(figsize=(12, 7))

    # Plot histogram
    counts, bins, _ = plt.hist(positions, bins=50, density=True, alpha=0.7)

    # Plot theoretical distribution
    x = np.linspace(0, n, 1000)
    # The correct density function for median-of-three pivot positions
    y = 6 * (x / n) * (1 - x / n) / n
    plt.plot(x, y, 'r-', linewidth=2, label='Theoretical density: 6(x/n)(1-x/n)/n')

    plt.xlabel('Pivot Position')
    plt.ylabel('Probability Density')
    plt.title(f'Distribution of Median-of-Three Pivot Positions (n={n})')
    plt.legend()
    plt.grid(True)

    plt.savefig('pivot_distribution.png')
    print("\nPivot distribution graph saved as 'pivot_distribution.png'")
    plt.show()


def main():
    print("Median-of-Three Pivot Selection in Quicksort Simulation")
    print("=" * 60)
```

```
    print("\nThis script demonstrates the probability distribution of pivot positions")
    print("when using median-of-three selection in Quicksort.")
    print("\nComparing empirical results with theoretical formulas...")

    # Compare empirical results with theoretical formula
    compare_empirical_vs_theoretical()

    # Show the distribution of pivot positions
    print("\nGenerating distribution of pivot positions...")
    pivot_position_distribution(n=1000, num_trials=50000)

    # Focused analysis on specific a values
    print("\nDetailed probability analysis for specific values of a:")
    n = 1000
    for a in [0.1, 0.2, 0.3, 0.4]:
        empirical = estimate_probability(n, a, num_trials=50000)
        theoretical = theoretical_probability_correct(a, n)
        print(f"a = {a:.1f}:")
        print(f"  - Empirical probability: {empirical:.6f}")
        print(f"  - Correct theoretical formula: {theoretical:.6f}")
        print(f"  - Difference (empirical vs correct): {abs(empirical - theoretical):.6f}")

        # Show what this means
        if a == 0.1:
            print(f"  - Interpretation: ~{empirical * 100:.1f}% chance of getting at worst a 10-90 split")
        elif a == 0.2:
            print(f"  - Interpretation: ~{empirical * 100:.1f}% chance of getting at worst a 20-80 split")
        elif a == 0.3:
            print(f"  - Interpretation: ~{empirical * 100:.1f}% chance of getting at worst a 30-70 split")
        elif a == 0.4:
            print(f"  - Interpretation: ~{empirical * 100:.1f}% chance of getting at worst a 40-60 split")


if __name__ == "__main__":
    main()
```