EXPERIMENT NO:05

```
import random
import time

def quicksort_deterministic(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quicksort_deterministic(arr, low, pi - 1)
        quicksort_deterministic(arr, pi + 1, high)

def partition(arr, low, high):
    pivot = arr[low]  # Choosing the first element as pivot
    i = low + 1
    for j in range(low + 1, high + 1):
        if arr[j] < pivot:
            arr[i], arr[j] = arr[j], arr[i]
            i += 1
    arr[low], arr[i - 1] = arr[i - 1], arr[low]
    return i - 1

def quicksort_randomized(arr, low, high):
    if low < high:
        pi = randomized_partition(arr, low, high)
        quicksort_randomized(arr, low, pi - 1)
        quicksort_randomized(arr, pi + 1, high)

def randomized_partition(arr, low, high):
    random_index = random.randint(low, high)
    arr[low], arr[random_index] = arr[random_index], arr[low]  # Swap with the first element
```

```
    return partition(arr, low, high)

def analyze_sorting_time(sort_func, arr):
    start_time = time.time()
    sort_func(arr.copy(), 0, len(arr) - 1)  # Copy to keep original array intact
    end_time = time.time()
    return end_time - start_time

# Example usage
if __name__ == "__main__":
    # Generate a random list of integers
    array_size = 1000
    arr = [random.randint(1, 10000) for _ in range(array_size)]

    # Analyze deterministic quicksort
    time_deterministic = analyze_sorting_time(quicksort_deterministic, arr)
    print(f"Time taken by Deterministic Quick Sort: {time_deterministic:.6f} seconds")

    # Analyze randomized quicksort
    time_randomized = analyze_sorting_time(quicksort_randomized, arr)
    print(f"Time taken by Randomized Quick Sort: {time_randomized:.6f} seconds")
```

OUTPUT:-

Time taken by Deterministic Quick Sort: X.XXXXXX seconds

Time taken by Randomized Quick Sort: Y.YYYYYY seconds