Devam Desai
IIT2022035
DAA Assignment 5

Q1. **Randomised QuickSort**
Code Used for Analysis:

```
void solve(int n){
    // cout << n << endl;
    swap_ct=0;
    vector<int> arr;
    int maxi = 1e5;

    for (int i = 0;i < n;i++){
        arr.push_back(rand()%(maxi));
    }
    int ans=0;

    auto start = high_resolution_clock::now();
    quicksort(arr,0,n-1);
    auto stop = high_resolution_clock::now();
    // cout<<endl;
    // printArray(arr,n);
    // auto start1 = high_resolution_clock::now();

            // ans=countInversions(arr,n);
    // auto stop1 = high_resolution_clock::now();
    // cout<<ct<<endl;
    auto duration = duration_cast<microseconds>(stop - start);
    // auto duration1=duration_cast<microseconds>(stop1 - start1);
    cout << swap_ct << endl;
}
```

Code:

```
void swap(vint &arr, int i, int j) {
    swap_ct++;
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

int partitionLeft(vint &arr, int low, int high) {
    int pivot = arr[high];
    int i = low;
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            swap(arr, i, j);
            i++;
        }
    }
    swap(arr, i, high);
    return i;
}
```

```
int partitionRight(vint &arr, int low, int high) {
    srand(time(NULL));

//CHOOSING PIVOT ELEMENT HERE

    int r = low + rand() % (high - low);
    swap(arr, r, high);
    return partitionLeft(arr, low, high);
}

void quicksort(vint &arr, int low, int high) {
    if (low < high) {
        int p = partitionRight(arr, low, high);
        quicksort(arr, low, p - 1);
        quicksort(arr, p + 1, high);
    }
}
```

Apriori Analysis:

Analysis of the time complexity of the randomized quicksort algorithm in terms of its best, average, and worst-case scenarios.

1. *Best-case scenario:*
   - The best-case scenario occurs when the pivot selected in each partitioning step happens to be the median of the array.
   - In this ideal situation, the array is perfectly divided into two equal halves at each recursive call.
   - The recurrence relation becomes $T(n) = 2T(n/2) + \Theta(n)$, where $T(n)$ is the time complexity for an array of size n.
   - The solution to this recurrence relation is $T(n) = O(n \log n)$.

2. *Average-case scenario:*
   - In the average case, the pivot is selected randomly, and on average, it has a good chance of partitioning the array in a balanced way.
   - The expected time complexity for the average case is $O(n \log n)$.
   - This result is derived by considering the expected size of subproblems at each level of recursion.

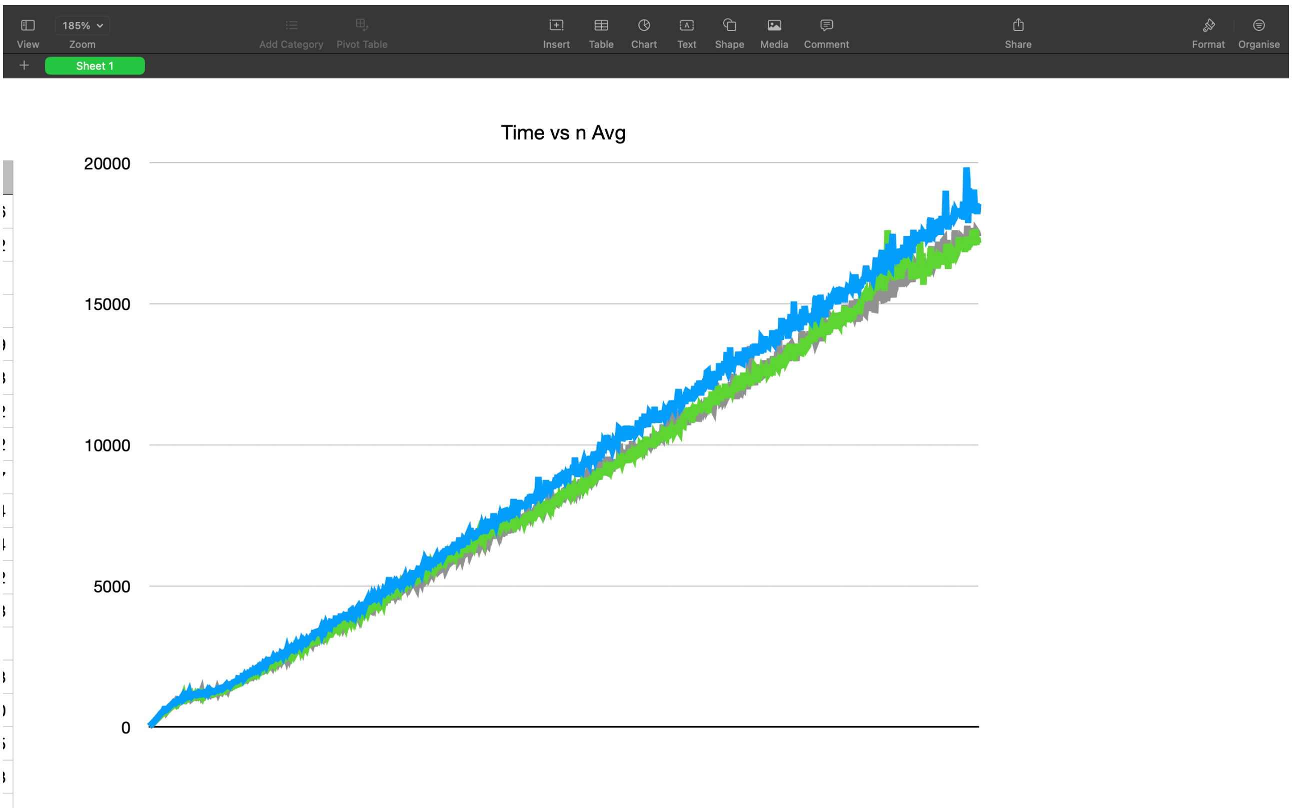
3. *Worst-case scenario:*
   - The worst-case scenario occurs when the pivot selection is always biased, leading to unbalanced partitioning.
   - The recurrence relation becomes $T(n) = T(n-1) + \Theta(n)$, where $T(n)$ is the time complexity for an array of size n.
   - The solution to this recurrence relation is $T(n) = O(n^2)$.
   - However, the probability of encountering the worst-case scenario is very low due to the random selection of the pivot.

In summary, the randomized quicksort algorithm has an average-case time complexity of $O(n \log n)$, which is its most significant and commonly analyzed performance measure. The worst-case time complexity is theoretically $O(n^2)$, but the probability of this occurring is

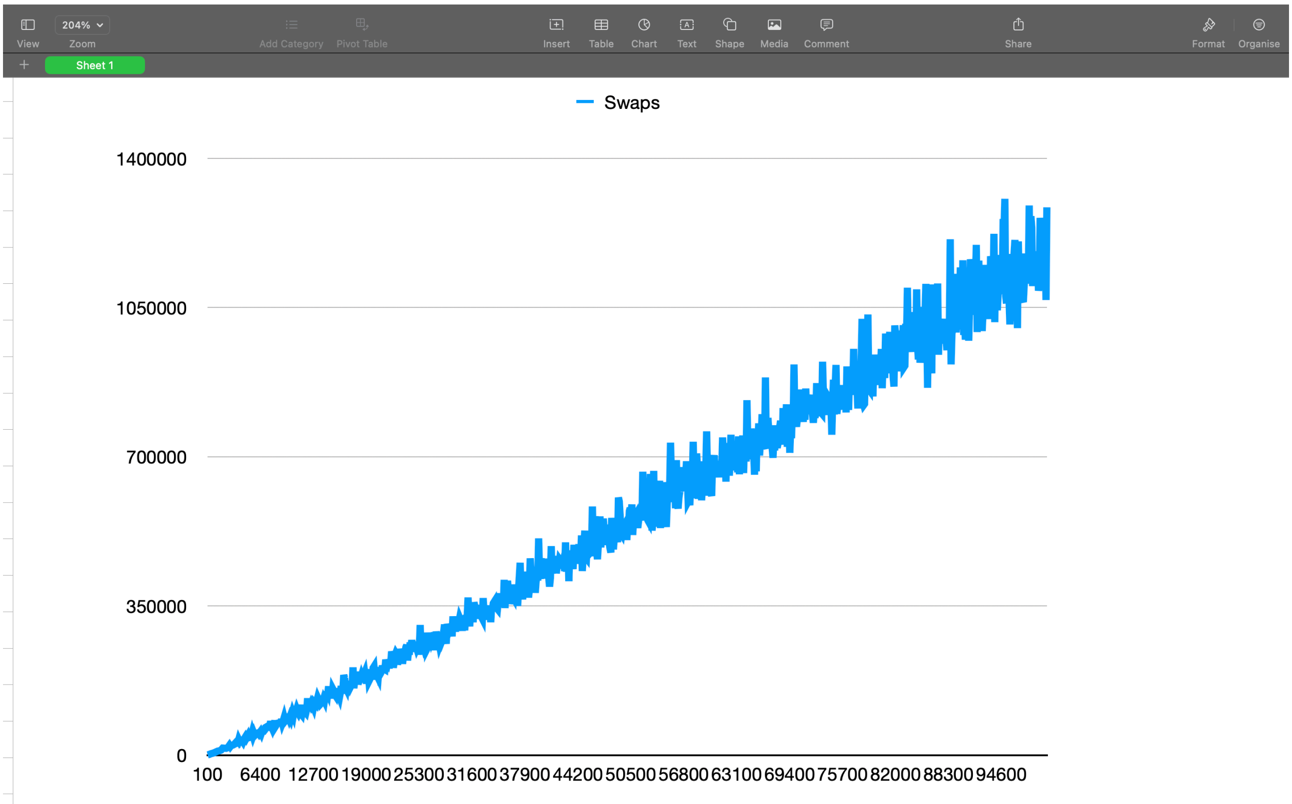
low due to the random pivot selection. The best-case time complexity is O(n log n) when the pivot selection is always optimal.
Hence Randomized Quicksort performs better in the worst case than any other fixed pivot selection algorithm

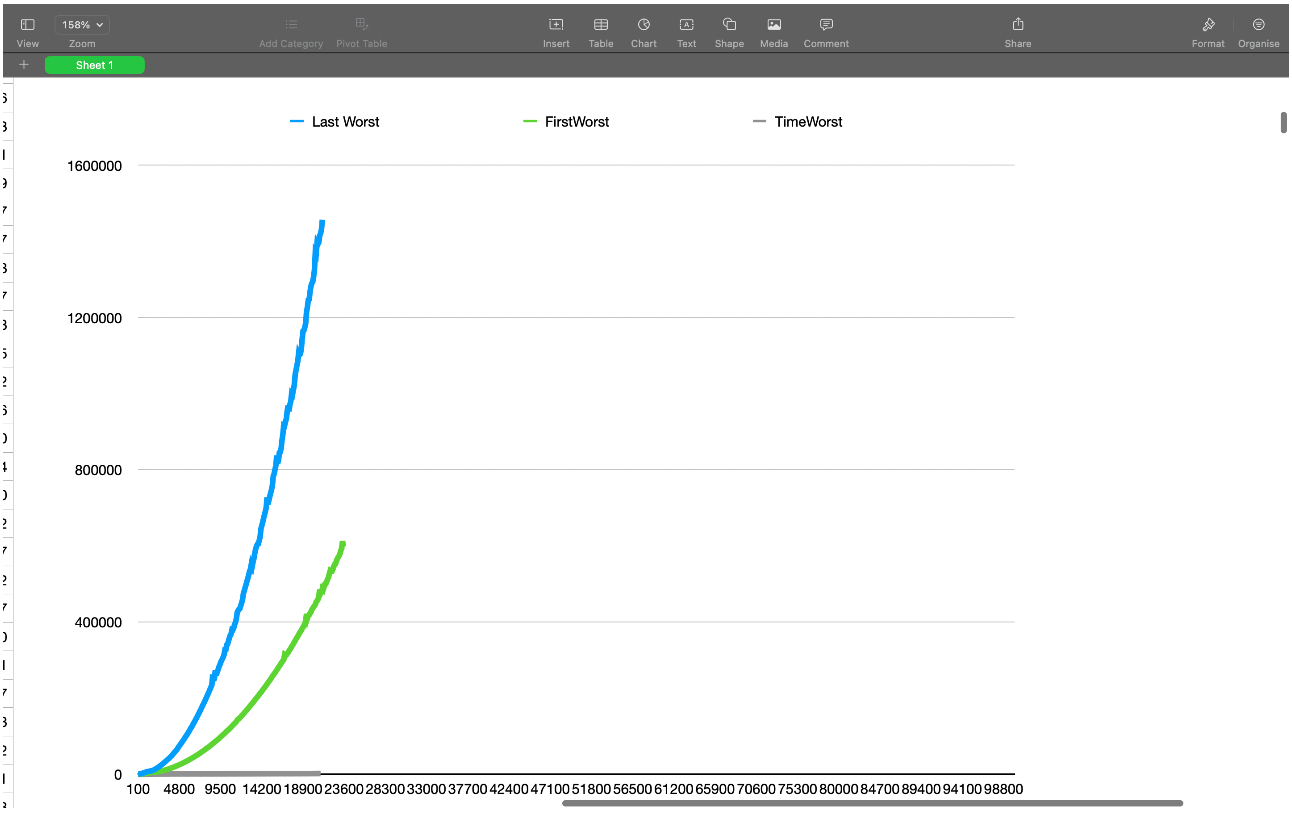
A Posteriori Analysis:

Graph of Average case of First Pivot, Last Pivot and Randomised Quicksort:



Graph of Swaps in Randomised QuickSort:





Graph of Worst Case of Last, First Pivot and Randomised Quicksort: Worst case calculated considering input as Sorted Array

Excel File for the Data is Attached.

Accuracy and Correctness: The algorithm gives the correct sorted output for each input array and size.

Profiling:
Excel File attached and Graphs given above.