ALGORITHM LABORATORY

ASSIGNMENT-3a

PROBLEM STATEMENT: Update Quick sort algorithm implemented in Assignment 3 such that every time a random element is chosen as the pivot.

1. *Quick Sort (randomised element as pivot):*

ALGORITHM (QUICK SORT (randomised element as pivot)):

a. Choose a Pivot Randomly
   i. Select a pivot element randomly from the array
   ii. Swap it with the last element to use it as the pivot
b. Partition the Array
   i. Rearrange elements such that:
      1. Elements smaller than the pivot go to the left
      2. Elements larger than the pivot go to the right
   ii. The pivot is now in its correct sorted position
c. Recursively Apply Quick Sort
   i. Apply Quick Sort to the left and right subarrays (excluding the pivot)
d. Repeat Until Sorted
   i. Continue the process until all subarrays are of size 1 or empty

PROGRAM CODE:

```
#include <bits/stdc++.h>
using namespace std;
using namespace std::chrono;

int partition(vector<int>& arr, int low, int high) {
    int random_index = low + rand()%(high-low+1);
    swap(arr[low],arr[random_index]);
    int pivot = arr[low];
    int i = low + 1;
```

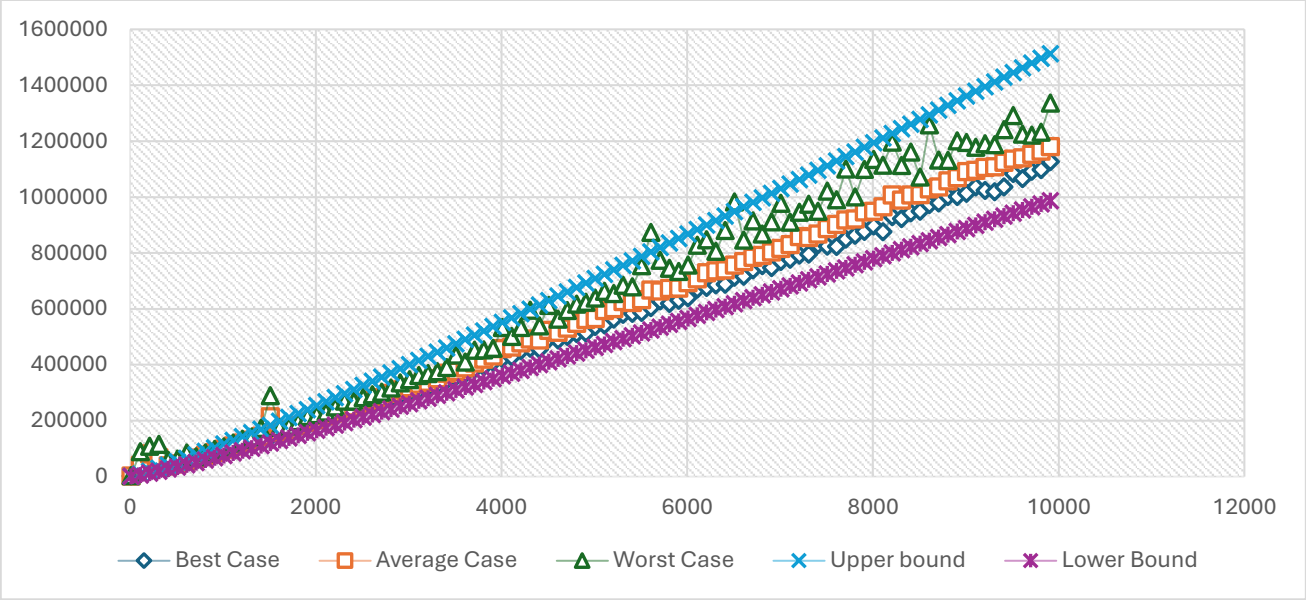
```
    for (int j = low + 1; j <= high; j++) {
        if (arr[j] < pivot) {
            swap(arr[i], arr[j]);
            i++;
        }
    }
    swap(arr[low], arr[i - 1]);
    return i - 1;
}

void quickSort(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    srand(time(NULL));
    for (int k = 10;k<=10000;k+=100) {
        double best_case_average_time = INT_MAX;
        double worst_case_average_time = INT_MIN;
        double random_case_average_time = 0;
        int num_trials = 100;
        if (k == 1000000) num_trials = 10;

        for (int i = 0; i < num_trials; i++) {
            vector<int> arr(k);
            for (int j = 0; j < k; j++)
                arr[j] = rand() % (5 * k);

            auto start = high_resolution_clock::now();
            quickSort(arr, 0, k - 1);
            auto end = high_resolution_clock::now();
            random_case_average_time += duration_cast<nanoseconds>(end -
start).count();
            best_case_average_time = min((double)duration_cast<nanoseconds>(end -
start).count(),best_case_average_time);
            worst_case_average_time = max((double)duration_cast<nanoseconds>(end -
start).count(),worst_case_average_time);
        }

        random_case_average_time /= num_trials;

        cout <<k<<","<< best_case_average_time << ","<< random_case_average_time
<< "," << worst_case_average_time << endl;
    }
    return 0;
}
```

TIME COMPLEXITY:

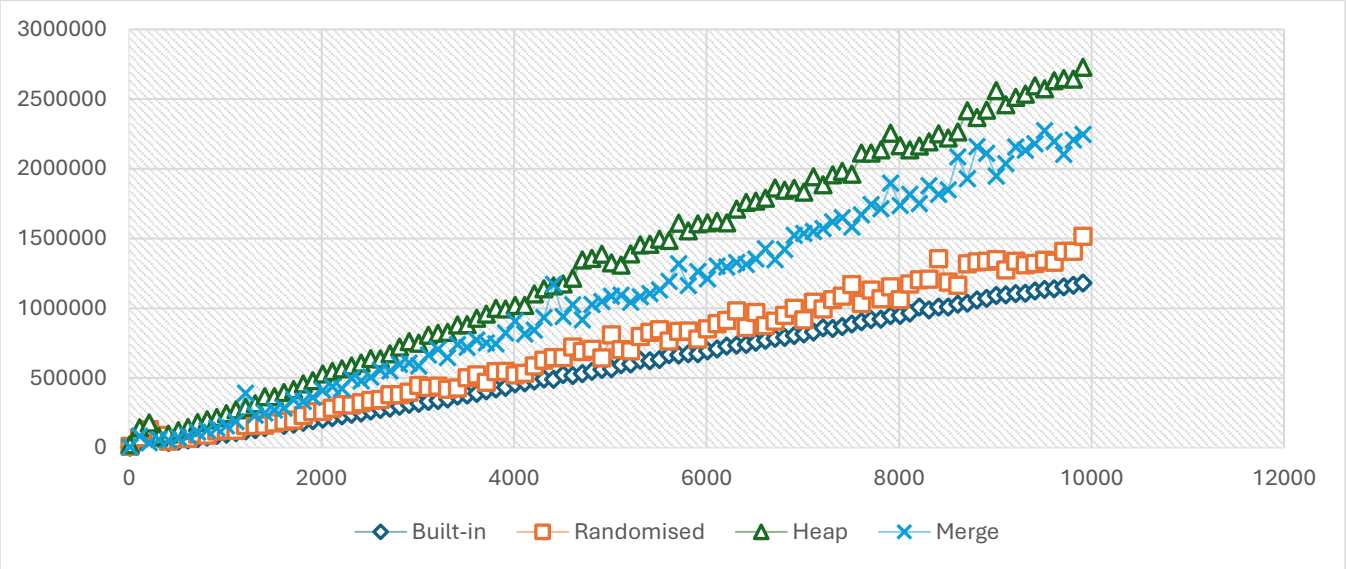
- Best Case: nlog(n)
- Average Case: nlog(n)
- Worst Case: nlog(n)

OUTPUT and PLOT:

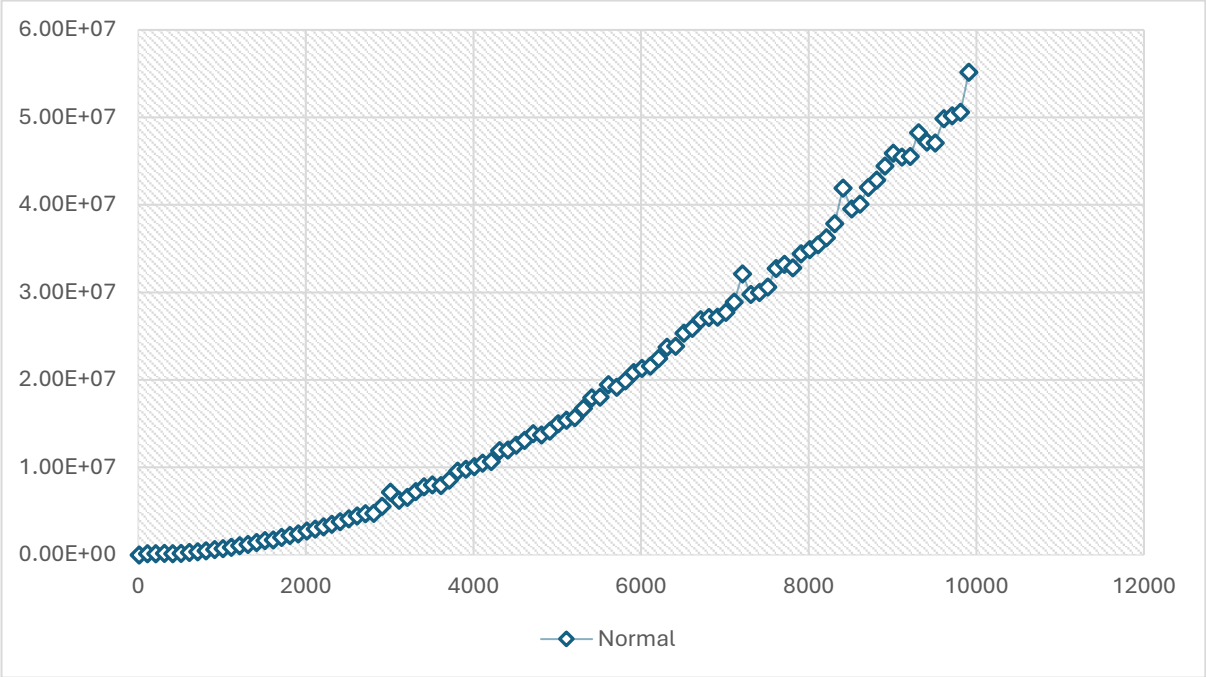
1. Randomised quick sort:

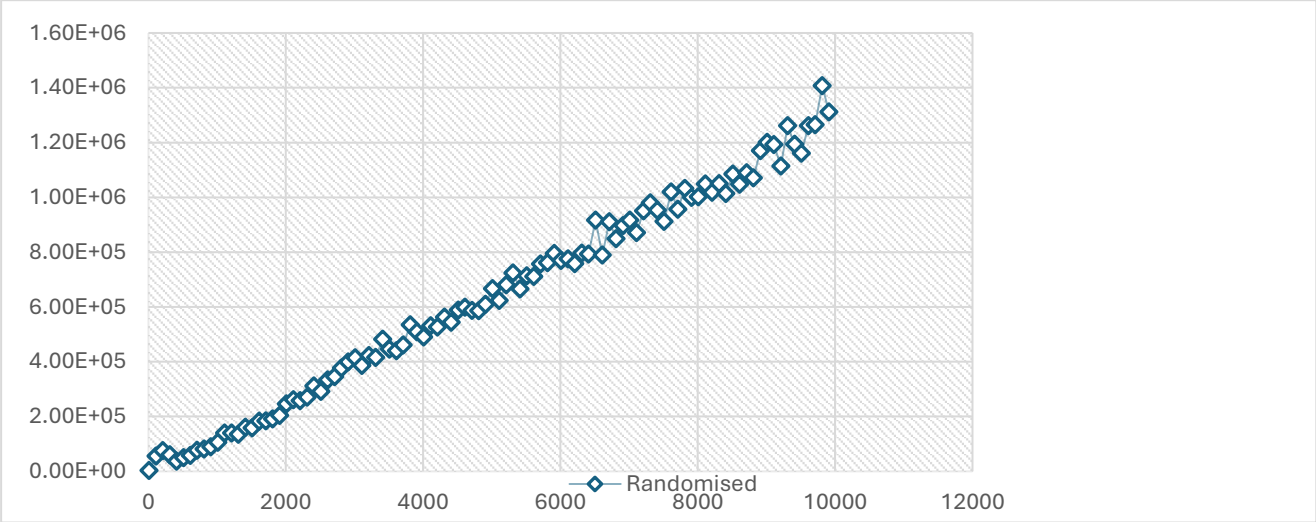


2. All sorts:

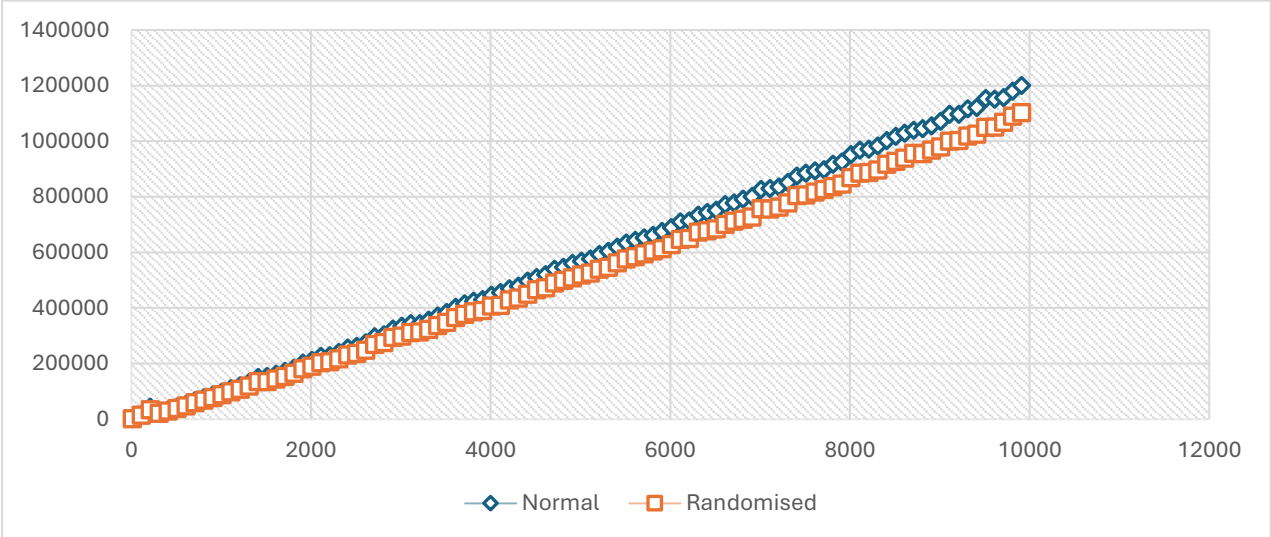


3. Worst Case (normal vs randomised quick sort):





4. Average case (normal vs randomised quick sort):



5. Best case (normal vs randomised quick sort):