

NAME:	Shubham Solanki
UID:	2022301015
SUBJECT	Design and Analysis of Algorithms
EXPERIMENT NO:	2(B)
AIM:	Understanding more concepts regarding quick sort algorithm
Algorithm:	<p>Quick Sort Algorithm:</p> <ol style="list-style-type: none"> 1. Pick a pivot element from the array (usually the first or last element). 2. Partition the array around the pivot element by rearranging the elements so that all elements smaller than the pivot come before it, and all elements larger than the pivot come after it. 3. Recursively apply steps 1 and 2 to the sub-array of elements smaller than the pivot, and the sub-array of elements larger than the pivot. 4. The base case of the recursion is when the sub-array has fewer than two elements, in which case it is already sorted.
Code:	<pre> #include <stdio.h> #include <stdlib.h> #include <time.h> long SWAP = 0; void merge(int arr[], int p, int q, int r) { int i, j, k; int n1 = q - p + 1; </pre>

```

int n2 = r - q;
int L[n1], R[n2];
for (i = 0; i < n1; i++)
    L[i] = arr[p + i];
for (j = 0; j < n2; j++)
    R[j] = arr[q + 1 + j];
i = 0;
j = 0;
k = p;
while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = R[j];
        j++;
    }
    k++;
}
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
}
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        int m = l + (r - l) / 2;

```

```

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int quicksort(int a[], int start, int end) {
    int pivot = a[end];
    //int pivot = a[start];
    //int random = start + rand() % (end - start);
    //int pivot = a[random];
    //int mid = start + (end - start)/2;
    //int pivot = a[mid];
    int i = (start - 1);
    for (int j = start; j <= end - 1; j++) {
        if (a[j] < pivot) {
            i++;
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
            SWAP++;
        }
    }
    int t = a[i + 1];
    a[i + 1] = a[end];
    a[end] = t;
    SWAP++;
    return (i + 1);
}

double quick(int a[], int start, int end) {
    if (start < end) {
        int p = quicksort(a, start, end);
        quick(a, start, p - 1);
        quick(a, p + 1, end);
    }
}

```

```

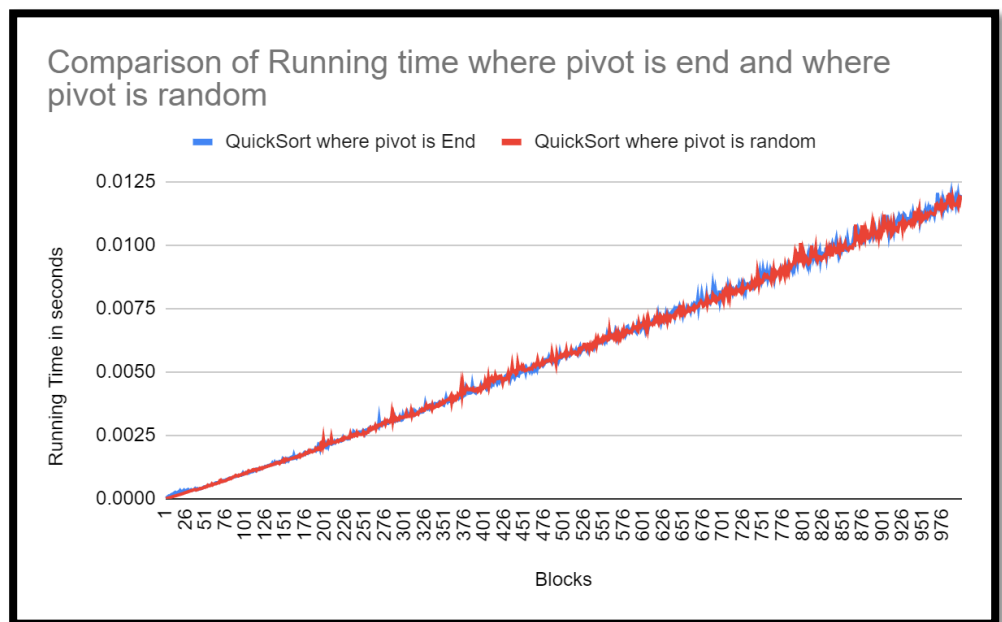
int main() {
    double qust, mest;
    srand(time(0));
    FILE * fp, * file;
    fp = fopen("random.txt", "w");
    for (int i = 0; i < 100000; i++) {
        fprintf(fp, "%d\n", rand() % 900001 + 100000);
    }
    int upper_limit = 100;
    fclose(fp);
    file = fopen("outputEnd.txt", "w");
    fprintf(file,
"Block\tMerSort\tQuickSort\tSwaps\n");
    for (int i = 0; i < 1000; i++) {
        fp = fopen("random.txt", "r");
        int arr1[upper_limit], arr2[upper_limit],
temp_num;
        for (int j = 0; j < upper_limit; j++) {
            fscanf(fp, "%d", & temp_num);
            arr1[j] = temp_num;
            arr2[j] = temp_num;
        }
        fclose(fp);
        clock_t t;
        t = clock();
        mergeSort(arr2, 0, upper_limit - 1);
        t = clock() - t;
        mest = ((double) t) / CLOCKS_PER_SEC;
        clock_t t1;
        t1 = clock();
        qust = quick(arr1, 0, upper_limit - 1);
        t1 = clock() - t1;
        qust = ((double) t1) / CLOCKS_PER_SEC;
        fprintf(file, "%d\t%lf\t%lf\t%ld\n", i + 1,
mest, qust, SWAP);
    }
}

```

```
fflush(stdout);
upper_limit += 100;
}
return 0;
}
```

Graphs and Observation:

Running time comparison for Different Pivot Positions

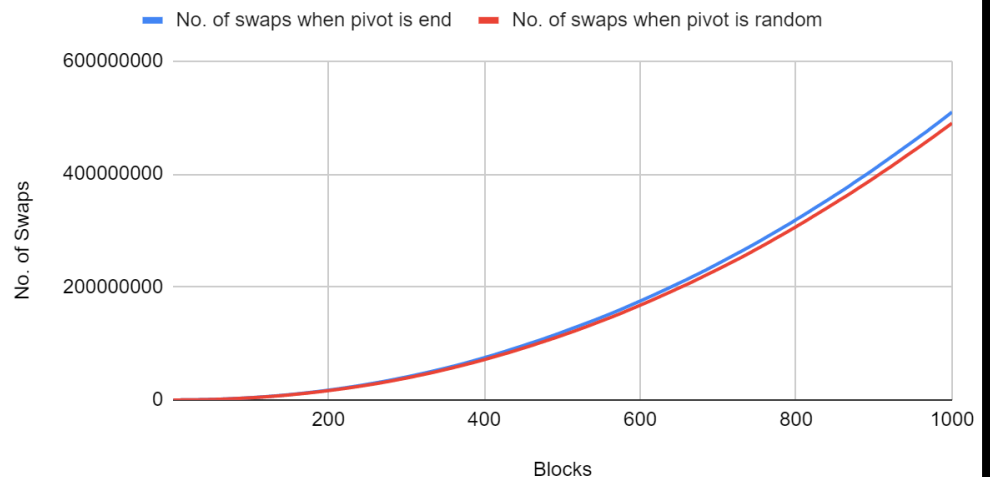


- Here, we can see that the time complexity of quick sort is nearly the same even when varied pivot points are taken into account.
- We can see that a quick sort when the pivot is at a random position takes longer than when the pivot is in an end position at the conclusion of execution.

- Despite the fact that both executions are finished in 0.1 seconds.

Number of swaps considering different pivot positions

Comparison of No. of swaps required when pivot is end and where pivot is random



- We can see from this that fewer swaps are needed for a quick sort when the pivot is in a random position as opposed to when it is at the end.
- The average number of swaps is over 500,000,000.
- Throughout the entire execution, the number of swaps keeps rising.

Conclusion:

Thus, we have provided observations for different pivots for quick sort algorithms.