NAME:	VEDANT TUSHAR DAPOLIKAR
UID:	2021700016
BRANCH:	CS-DS

EXPERIMENT- 2

- **AIM:** Experiment on finding the running time of an merge sort and quick sort algorithm.

- **ALGORITHM:**

  ❖ FOR MERGE SORT:

```
step 1: start
step 2: declare array and left, right, mid variable
step 3: perform merge function.
   if left > right
      return
   mid= (left+right)/2
   mergesort(array, left, mid)
   mergesort(array, mid+1, right)
   merge(array, left, mid, right)
Step 4: for mergesort(),
   ● Find the middle point to divide the array into two halves:
   ● Call mergesort for first half,then for second half
   ● Merge the two sorted halves

step 5: Stop
```

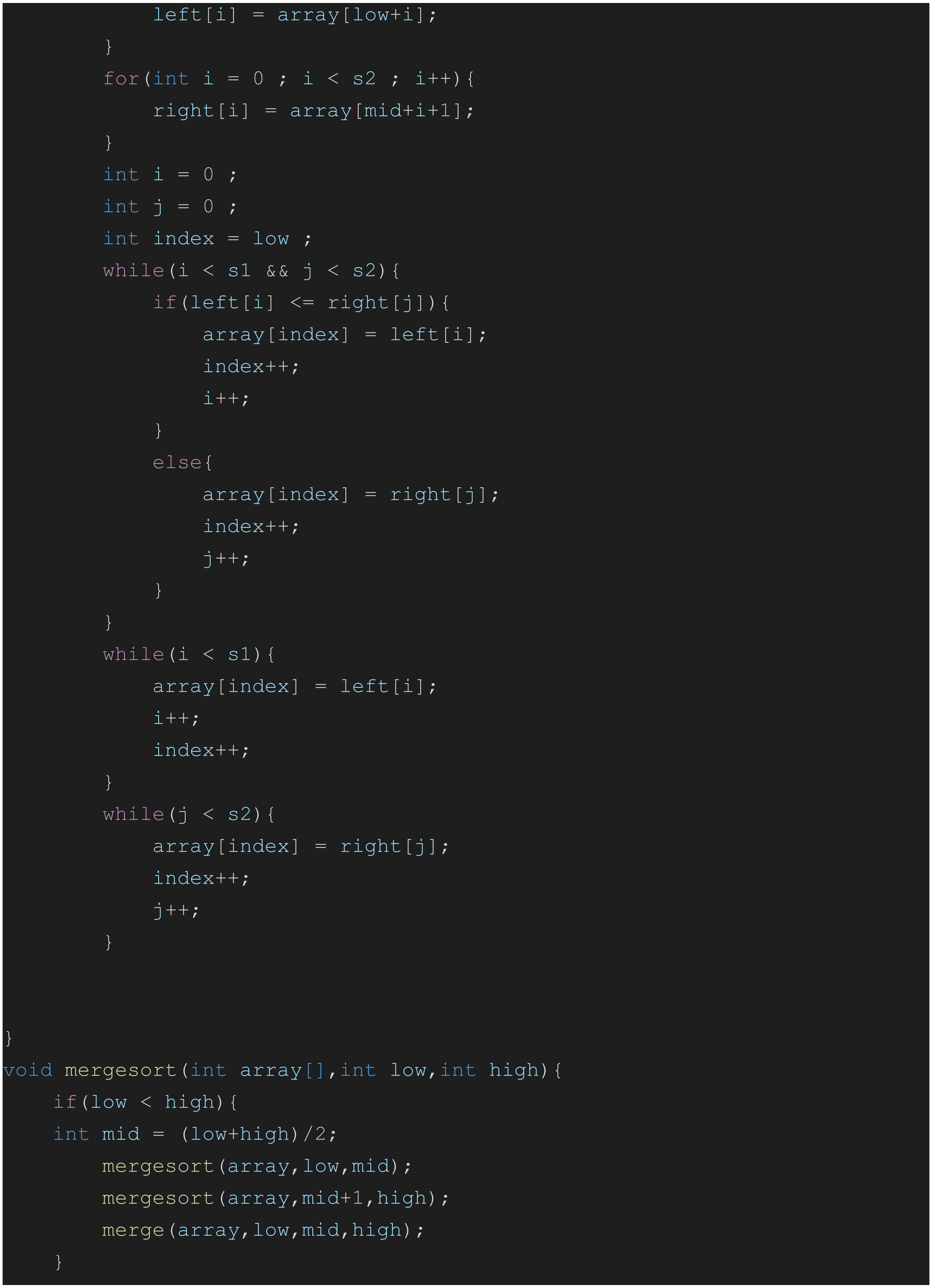
- FOR INSERTION SORT:

Quicksort is a sorting algorithm based on the divide and conquer approach where

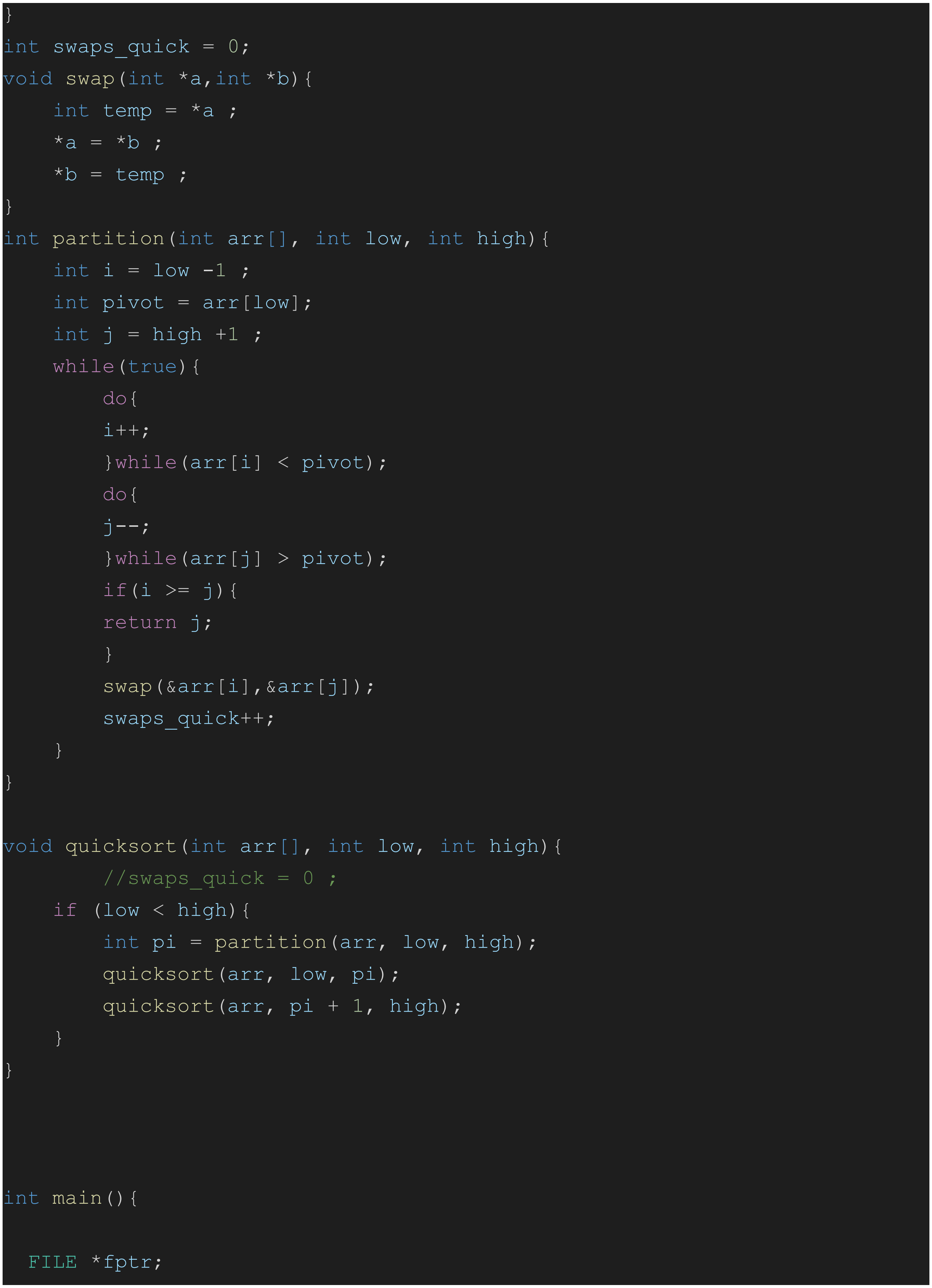
1. An array is divided into subarrays by selecting a pivot element
2. The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.
3. If the element is greater than the pivot element, a second pointer is set for that element.
4. Now, pivot is compared with other elements. If an element smaller than the pivot element is reached, the smaller element is swapped with the greater element found earlier.
5. Again, the process is repeated to set the next greater element as the second pointer. And, swap it with another smaller element.
6. At this point, elements are already sorted. Finally, elements are combined to form a sorted array.
7. End

- **PROGRAM:**

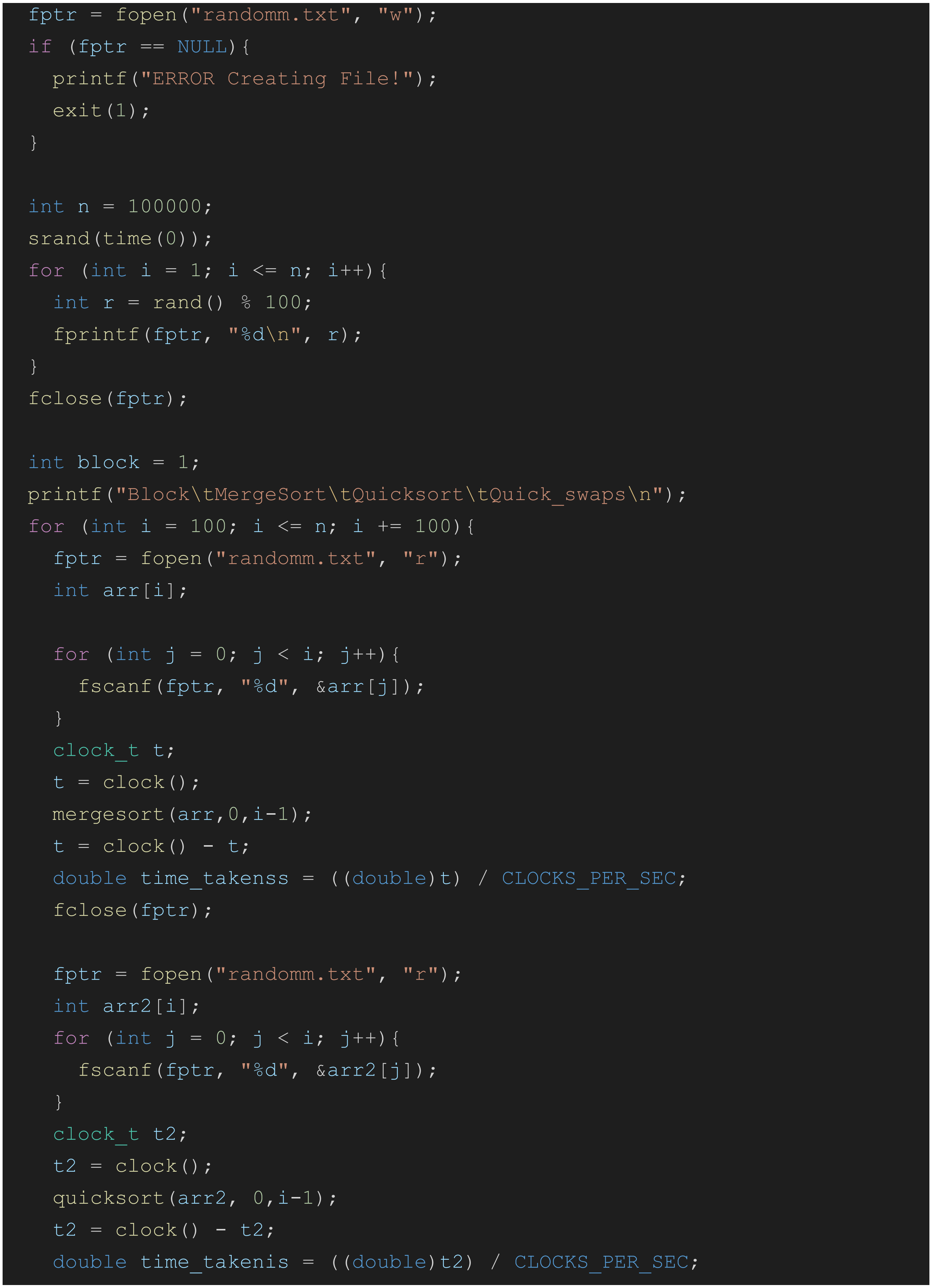
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include<stdbool.h>
void merge(int array[], int low, int mid,int high){
        int s1 = mid - low +1 ;
        int s2 = high - mid ;
        int left[s1];
        int right[s2];
        for(int i = 0 ; i < s1 ; i++){
```



```
            left[i] = array[low+i];
        }
        for(int i = 0 ; i < s2 ; i++){
            right[i] = array[mid+i+1];
        }
        int i = 0 ;
        int j = 0 ;
        int index = low ;
        while(i < s1 && j < s2){
            if(left[i] <= right[j]){
                array[index] = left[i];
                index++;
                i++;
            }
            else{
                array[index] = right[j];
                index++;
                j++;
            }
        }
        while(i < s1){
            array[index] = left[i];
            i++;
            index++;
        }
        while(j < s2){
            array[index] = right[j];
            index++;
            j++;
        }


}
void mergesort(int array[],int low,int high){
    if(low < high){
    int mid = (low+high)/2;
        mergesort(array,low,mid);
        mergesort(array,mid+1,high);
        merge(array,low,mid,high);
    }
```

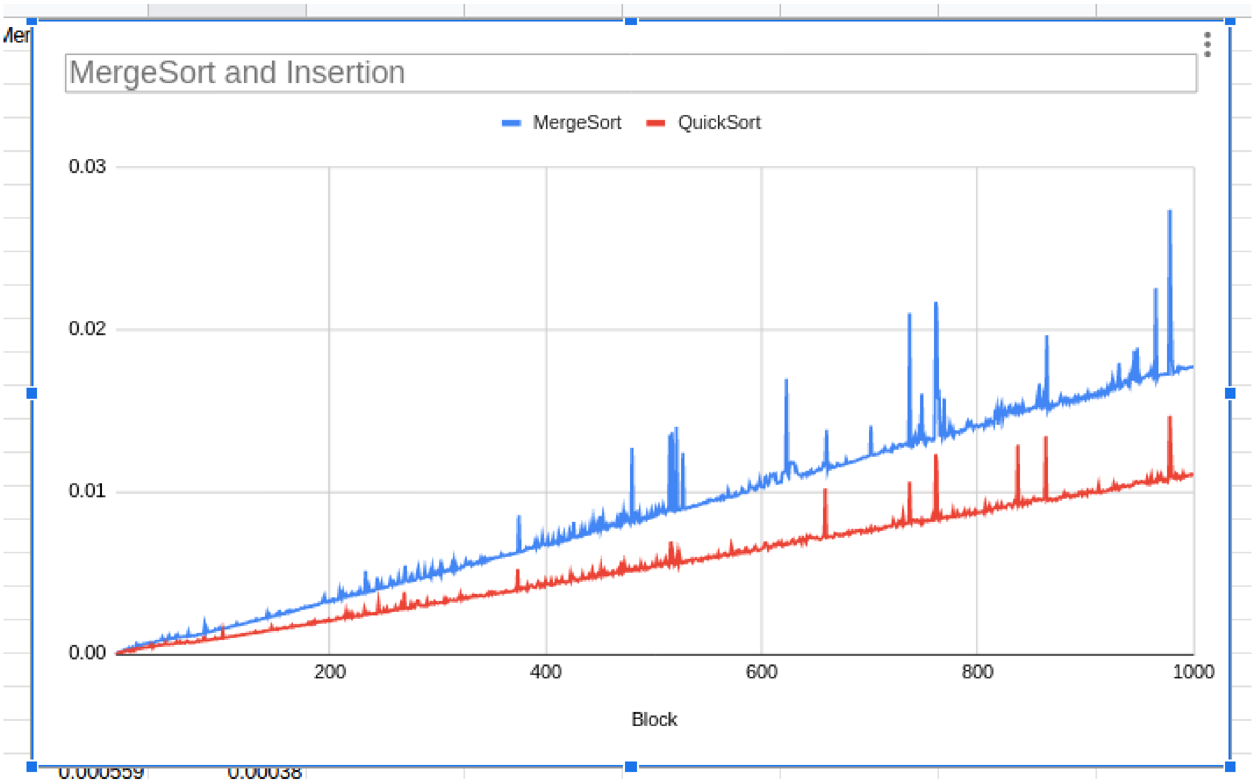


```
}
int swaps_quick = 0;
void swap(int *a,int *b){
    int temp = *a ;
    *a = *b ;
    *b = temp ;
}
int partition(int arr[], int low, int high){
    int i = low -1 ;
    int pivot = arr[low];
    int j = high +1 ;
    while(true){
        do{
        i++;
        }while(arr[i] < pivot);
        do{
        j--;
        }while(arr[j] > pivot);
        if(i >= j){
        return j;
        }
        swap(&arr[i],&arr[j]);
        swaps_quick++;
    }
}

void quicksort(int arr[], int low, int high){
        //swaps_quick = 0 ;
    if (low < high){
        int pi = partition(arr, low, high);
        quicksort(arr, low, pi);
        quicksort(arr, pi + 1, high);
    }
}



int main(){

  FILE *fptr;
```

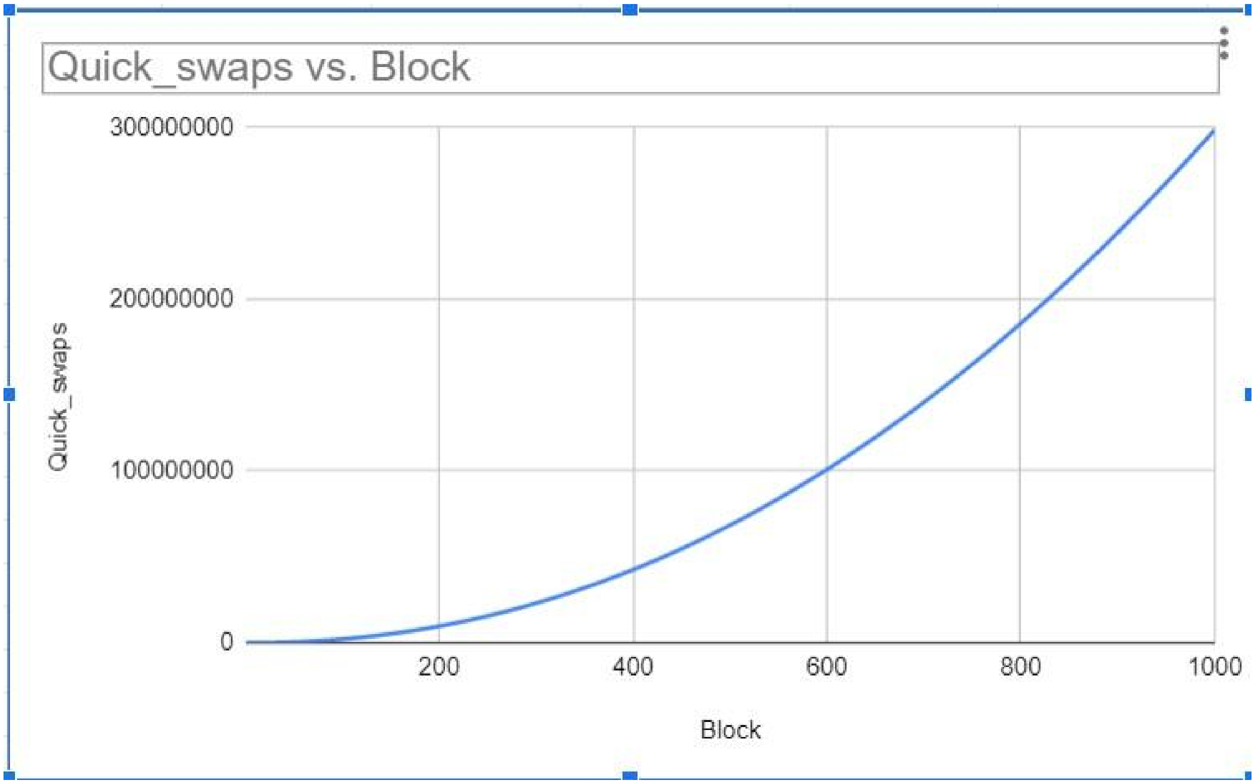


```
  fptr = fopen("randomm.txt", "w");
  if (fptr == NULL){
    printf("ERROR Creating File!");
    exit(1);
  }

  int n = 100000;
  srand(time(0));
  for (int i = 1; i <= n; i++){
    int r = rand() % 100;
    fprintf(fptr, "%d\n", r);
  }
  fclose(fptr);

  int block = 1;
  printf("Block\tMergeSort\tQuicksort\tQuick_swaps\n");
  for (int i = 100; i <= n; i += 100){
    fptr = fopen("randomm.txt", "r");
    int arr[i];

    for (int j = 0; j < i; j++){
      fscanf(fptr, "%d", &arr[j]);
    }
    clock_t t;
    t = clock();
    mergesort(arr,0,i-1);
    t = clock() - t;
    double time_takenss = ((double)t) / CLOCKS_PER_SEC;
    fclose(fptr);

    fptr = fopen("randomm.txt", "r");
    int arr2[i];
    for (int j = 0; j < i; j++){
      fscanf(fptr, "%d", &arr2[j]);
    }
    clock_t t2;
    t2 = clock();
    quicksort(arr2, 0,i-1);
    t2 = clock() - t2;
    double time_takenis = ((double)t2) / CLOCKS_PER_SEC;
    printf("%d\t%f\t%f\t%d\n", block, time_takenss,
time_takenis,swaps_quick);

    fclose(fptr);
    block++;

  }
  return 0;
}
```

- **OUTPUT:**



FROM THIS GRAPH WE INTERPRET THAT QUICKSORT TAKES LESS TIME THAN MERGESORT ALGORITHM.
THUS, FOR BIGGER RANGE QUICKSORT IS MORE EFFICIENT .



- **CONCLUSION:**

WE HAVE USED TWO ALGORITHM TECHNIQUES i.e MERGESORT AND QUICKSORT TO SORT THE RANDOM NO.s . BOTH THE ALGORITHMS HAVE LESS TIME COMPLEXITY. I HAVE SEEN BEHAVIOR OF THE ALGORITHMS WITH TIME USING OF GRAPH . IT IS SEEN THAT QUICK HAS BETTER TIME COMPLEXITY THAN MERGE SORT.