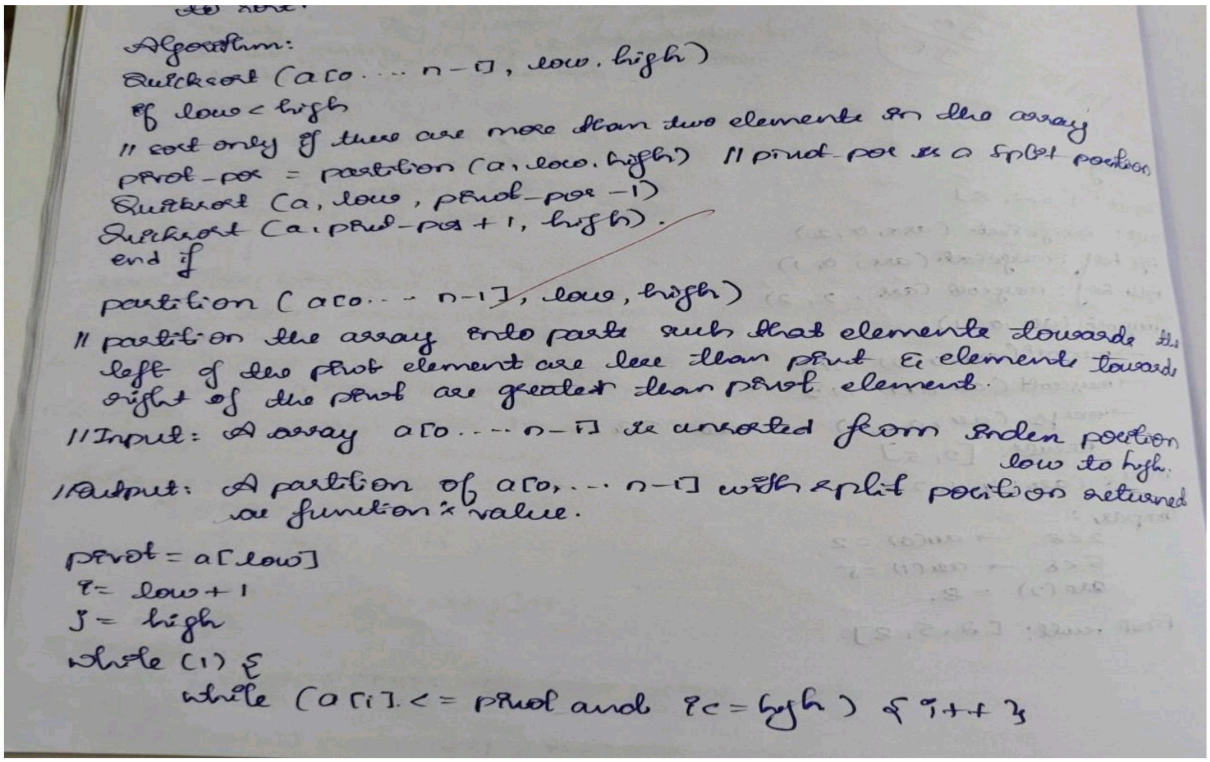
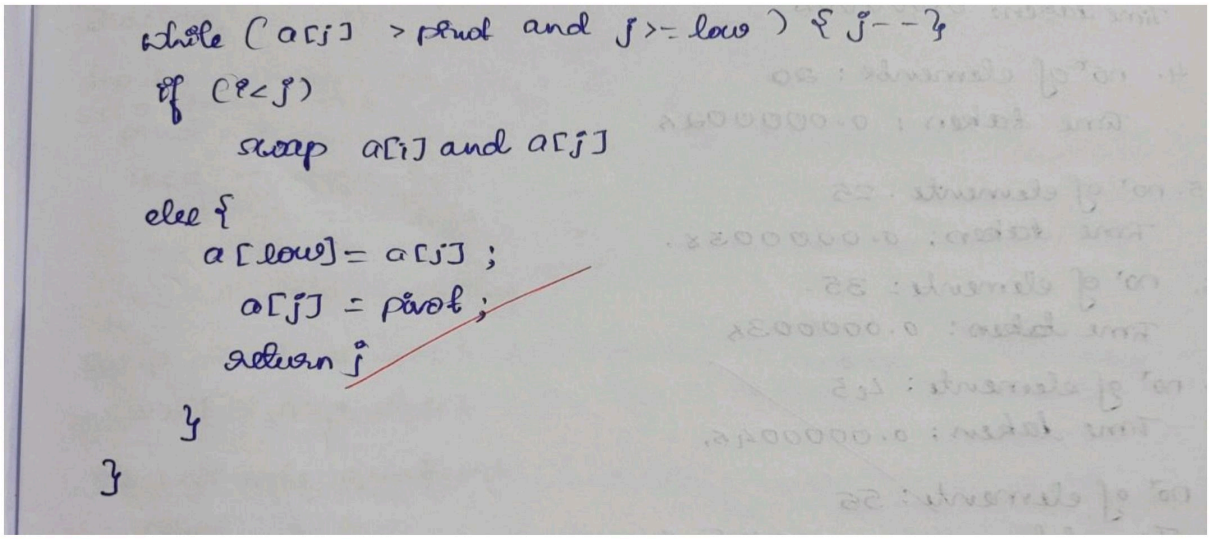
PROGRAM 2

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

ALGORITHM:



```
Algorithm:
Quicksort (a[0 .... n-1], low, high)
if low < high
// sort only if there are more than two elements in the array
pivot-pos = partition (a, low, high) // pivot-pos is a split position
Quicksort (a, low, pivot-pos -1)
Quicksort (a, pivot-pos + 1, high).
end if
partition (a[0 .... n-1], low, high)
// partition the array into parts such that elements towards the
left of the pivot element are less than pivot & elements towards
right of the pivot are greatest than pivot element.
//Input: An array a[0 .... n-1] is unsorted from index position
                                                    low to high.
//Output: A partition of a[0, ... n-1] with split position returned
        as function's value.
pivot = a[low]
i = low + 1
j = high
while (1) {
    while (a[i] <= pivot and i <= high ) { i++ }
```



```
while (a[j] > pivot and j >= low ) { j-- }
if (i < j)
    swap a[i] and a[j]
else {
    a[low] = a[j] ;
    a[j] = pivot ;
    return j
  }
}
```

CODE

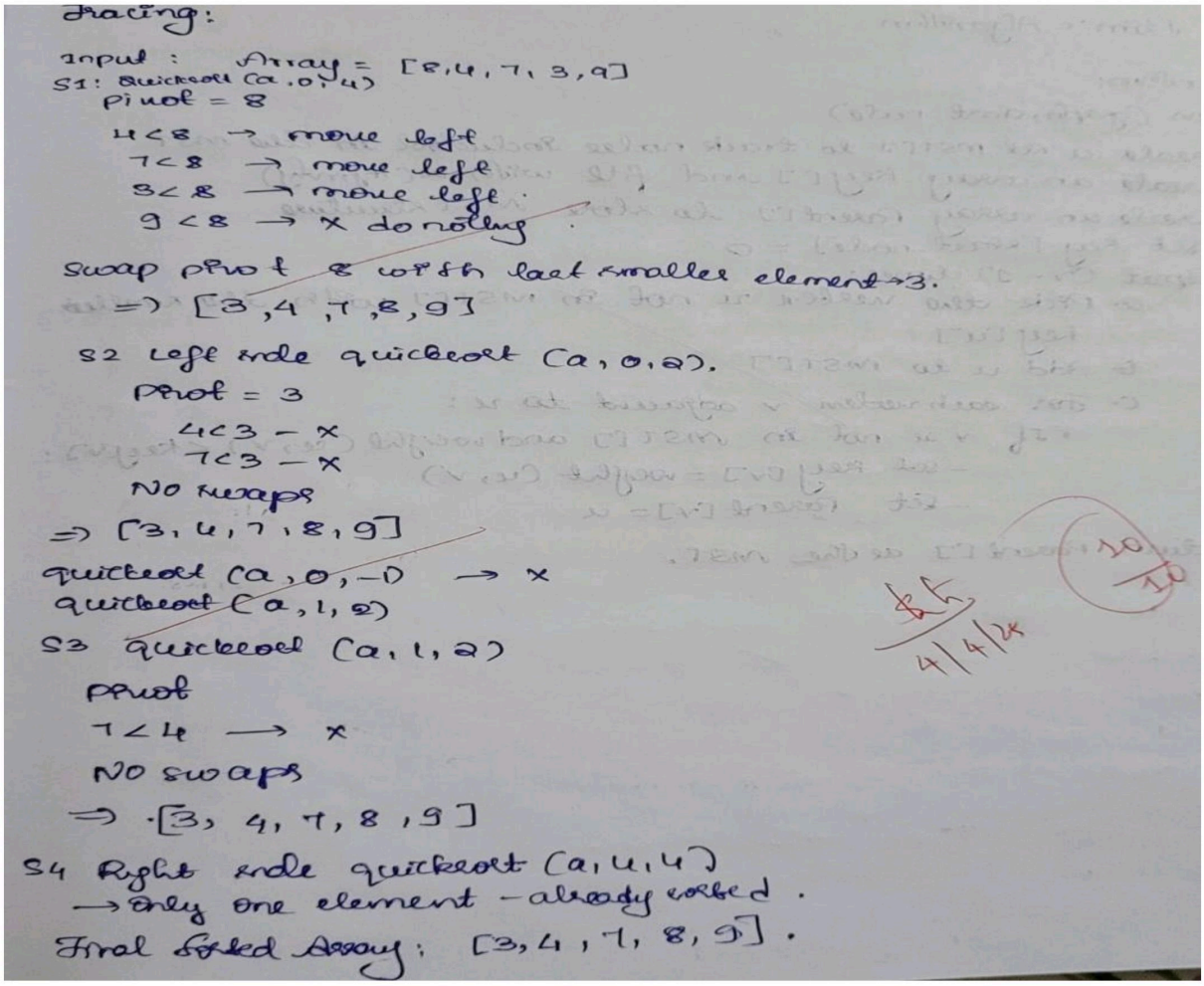
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to swap two elements
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Partition function
int partition(int arr[], int low, int high) {
    int pivot = arr[high];  // last element as pivot
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

// QuickSort function
void quickSort(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() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    clock_t start = clock();

    quickSort(arr, 0, n - 1);

    clock_t end = clock();
    double time_taken = (double)(end - start) / CLOCKS_PER_SEC;

    printf("\nSorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    printf("\n\nTime taken: %f seconds\n", time_taken);
    return 0;
}
```

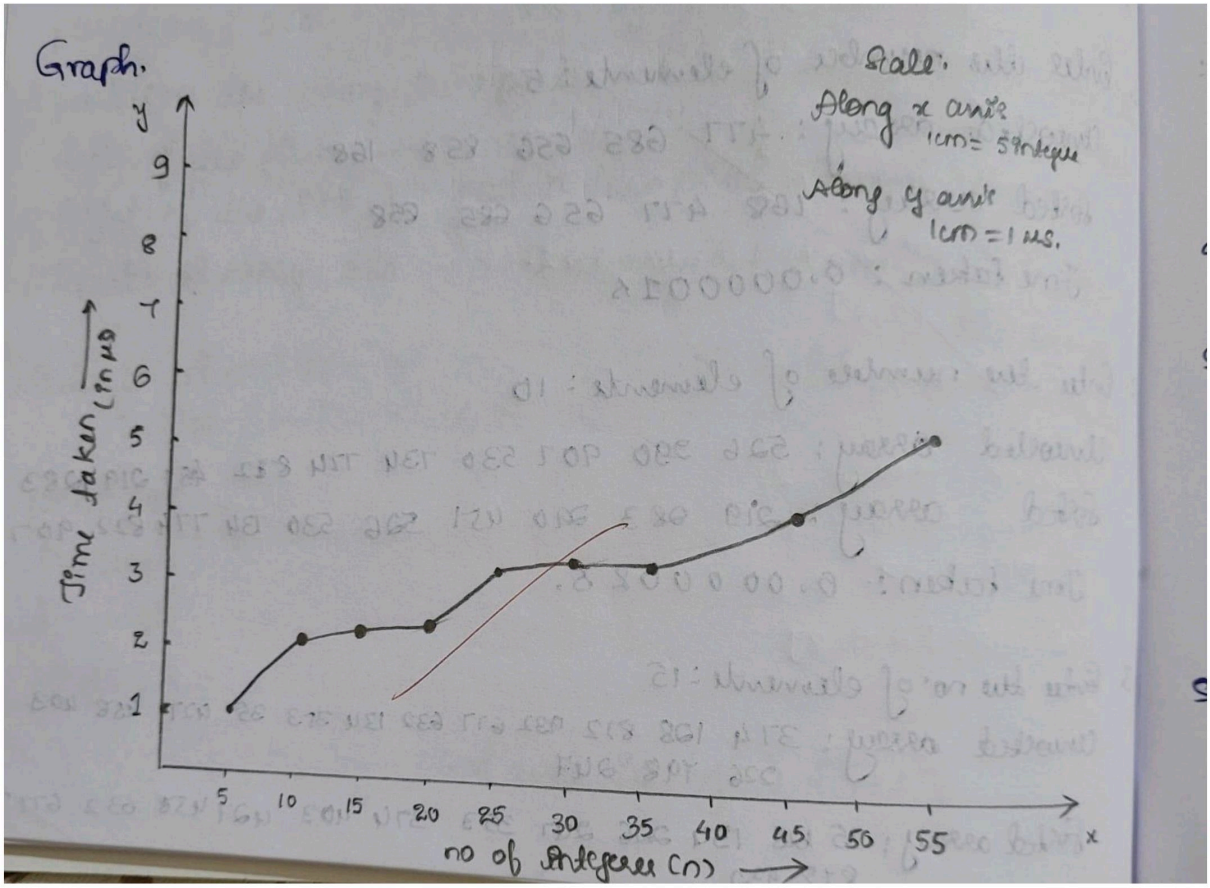
OUTPUT:

```
Enter number of elements: 5
Enter 5 integers:
56 89 45 2 45

Sorted array:
2 45 45 56 89

Time taken: 0.000002 seconds
```

TRACING:

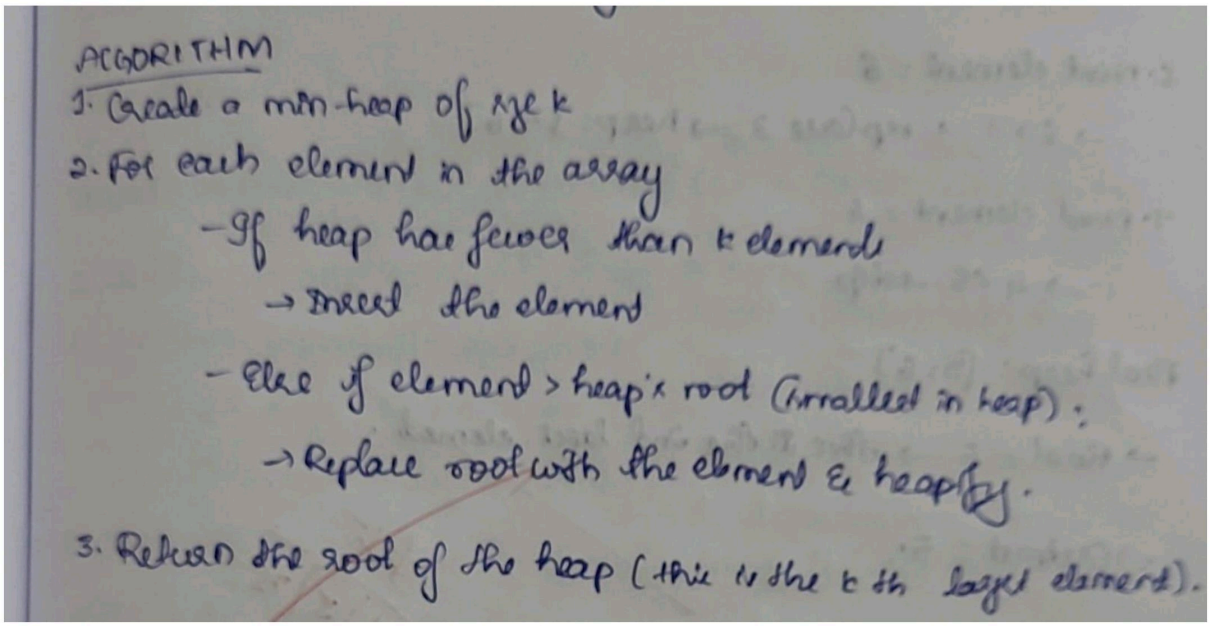


GRAPH:


Graph.
Scale:
Along x axis
1cm = 5 integers
Along y axis
1cm = 1 μs.
Time taken (in μs)
y
9
8
7
6
5
4
3
2
1
5
10
15
20
25
30
35
40
45
50
55
x
no of integers (n) →

LEETCODE 2:
K th LARGEST ELEMENT IN AN ARRAY

ALGORITHM:



CODE

```
#include <stdlib.h>


// Swap function
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}


// Partition for QuickSort
int partition(int* nums, int low, int high) {
    int pivot = nums[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (nums[j] <= pivot) {
```

```
            i++;
            swap(&nums[i], &nums[j]);
        }
    }
    swap(&nums[i + 1], &nums[high]);
    return i + 1;
}

// QuickSort
void quickSort(int* nums, int low, int high) {
    if (low < high) {
        int pi = partition(nums, low, high);
        quickSort(nums, low, pi - 1);
        quickSort(nums, pi + 1, high);
    }
}

int findKthLargest(int* nums, int numsSize, int k) {
    quickSort(nums, 0, numsSize - 1);
    return nums[numsSize - k]; // kth largest is (n-k)th smallest in sorted array
}
```

OUTPUT:

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =
[3,2,1,5,6,4]

k =
2

Output

5

Expected

5

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =
[3,2,3,1,2,4,5,5,6]

k =
4

Output

4

Expected

4

TRACING: