**AOA Lab 2**

Name: Vivian Vijay Ludrick

Roll No: 9914

Branch: SE Comps A

Batch: C

**Merge Sort(**Stable**):**

CODE:

#include <iostream>

// merge the two list

void mergelist(int arr[], int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1; // Size of left subarray

int n2 = r - m; // Size of right subarray

// Create temporary arrays

int L[n1], R[n2];

// Copy data to temporary arrays L[] and R[]

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

// Merge the temporary arrays back into arr[l..r]

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

// Copy the remaining elements of L[], if any

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

// Copy the remaining elements of R[], if any

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// l is for left index and r is right index of the sub-array of arr to be sorted

void mergeSort(int arr[], int l, int r) {

if (l < r) {

// Same as (l+r)/2, but avoids overflow for large l and r

int m = l + (r - l) / 2;

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Merge the sorted halves

mergelist(arr, l, m, r);

}

}

// Function to print an array

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++)

std::cout << arr[i] << " ";

std::cout << std::endl;

}

int main() {

// predefined array

int arr[] = {12, 11, 13, 5, 6, 6, 7};

int n = 7;

// print the given array

std::cout << "Original array: \n";

printArray(arr, n);

// apply the mergesort on the array

mergeSort(arr, 0, n - 1);

// print the sorted array

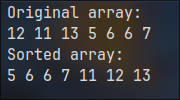
std::cout << "Sorted array: \n";

printArray(arr, n);

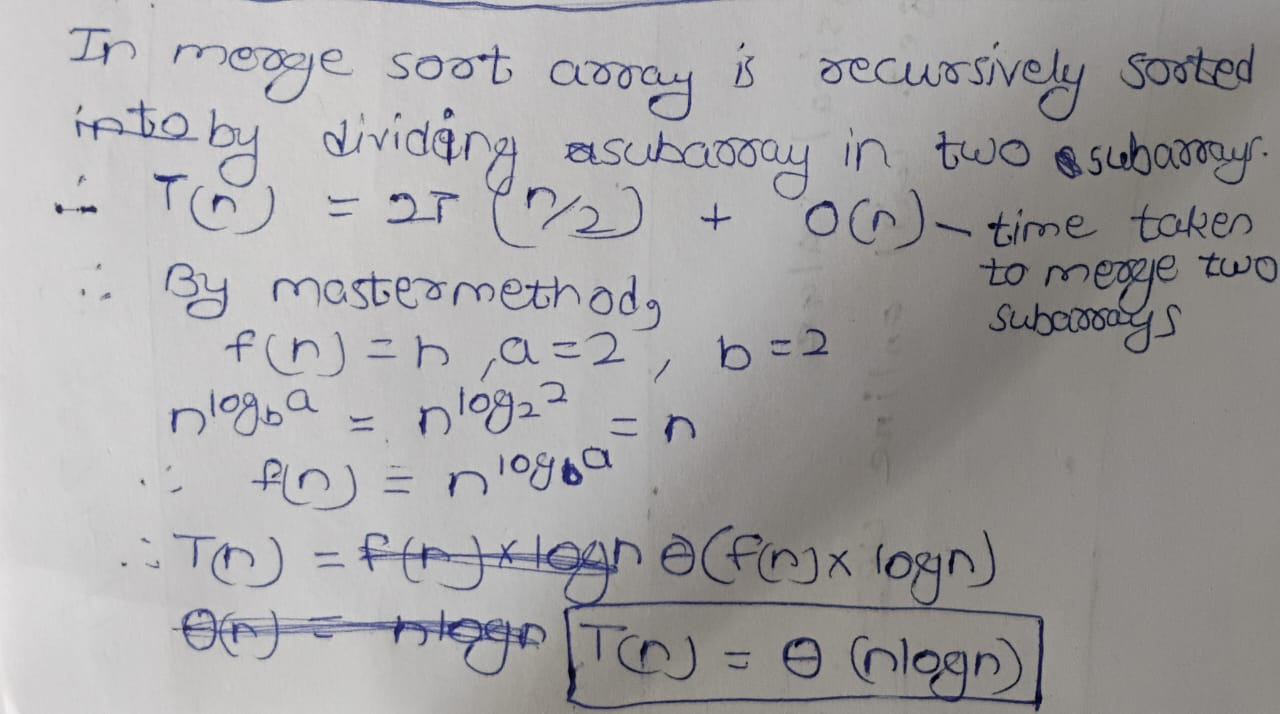
return 0;

}

OUTPUT:



COMPLEXITY:



**Quick Sort(**Fast**):**

CODE:

#include <iostream>

int partition(int arr[], int low, int high) {

// making the last element as the pivot element

int pivot = arr[high];

// get the index where the pivot needs to be placed

int i = low;

for (int j = low; j < high; j++) {

if (arr[j] < pivot) {

// if the current element is greater than the pivot then increment the

// counter and swap it with the previous element

i++;

std::swap(arr[i - 1], arr[j]);

}

}

// once the position to swap is found then swap the pivot to the position we

// get

std::swap(arr[i], arr[high]);

return i;

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

// getting the location of the pivot element

int pi = partition(arr, low, high);

// dividing the the unsorted sub arrays as before and after pivot element

// and again applying the quicksort on them

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

// predefined array

int arr[] = {12, 11, 13, 5, 6, 6, 7};

int arr\_size = 6;

// print the given array

std::cout << "Given array is \n";

for (int i = 0; i < arr\_size; i++)

std::cout << arr[i] << " ";

std::cout << std::endl;

// apply quicksort on the given array

quickSort(arr, 0, arr\_size - 1);

// print the sorted array

std::cout << "Sorted array is \n";

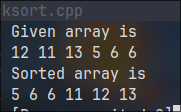
for (int i = 0; i < arr\_size; i++)

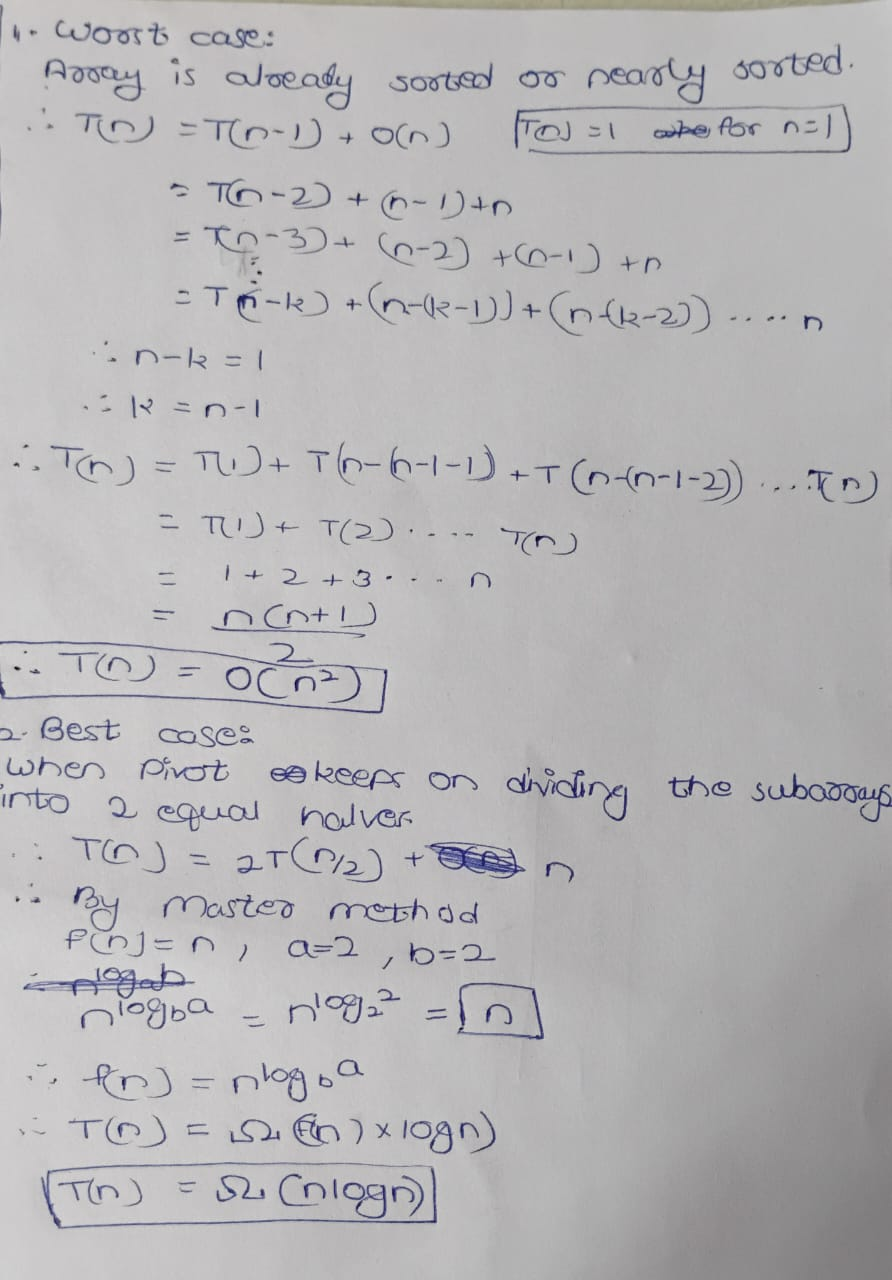
std::cout << arr[i] << " ";

return 0;

}

OUTPUT:



COMPLEXITY:

What changes will you do to make quick sort a randomized quick sort?

