#include <bits/stdc++.h>

using namespace std;

void insertionSort(int arr[], int n)

{

int i, key, j;

for (i = 1; i < n; i++)

{

key = arr[i];

j = i -1;

while (j >= 0 && arr[j] > key)

{

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

j = j - 1;

}

arr[j + 1] = key;

}

}

////////////////////////////////////////

void merge(int arr[], int l, int m, int r);

// Utility function to find minimum of two integers

int min(int x, int y) { return (x<y)? x :y; }

/\* Iterative mergesort function to sort arr[0...n-1] \*/

void mergeSort(int arr[], int n)

{

int curr\_size; // For current size of subarrays to be merged

// curr\_size varies from 1 to n/2

int left\_start; // For picking starting index of left subarray

// to be merged

// Merge subarrays in bottom up manner. First merge subarrays of

// size 1 to create sorted subarrays of size 2, then merge subarrays

// of size 2 to create sorted subarrays of size 4, and so on.

for (curr\_size=1; curr\_size<=n-1; curr\_size = 2\*curr\_size)

{

// Pick starting point of different subarrays of current size

for (left\_start=0; left\_start<n-1; left\_start += 2\*curr\_size)

{

// Find ending point of left subarray. mid+1 is starting

// point of right

int mid = min(left\_start + curr\_size - 1, n-1);

int right\_end = min(left\_start + 2\*curr\_size - 1, n-1);

// Merge Subarrays arr[left\_start...mid] & arr[mid+1...right\_end]

merge(arr, left\_start, mid, right\_end);

}

}

}

/\* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] \*/

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp 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 temp 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 there are any \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there are any \*/

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

/////////////////////////////////////////////////

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

/\* This function is same in both iterative and recursive\*/

int partition(int arr[], int l, int h)

{

int x = arr[h];

int i = (l - 1);

for (int j = l; j <= h - 1; j++) {

if (arr[j] <= x) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[h]);

return (i + 1);

}

/\* A[] --> Array to be sorted,

l --> Starting index,

h --> Ending index \*/

void quickSort(int arr[], int l, int h)

{

// Create an auxiliary stack

int stack[h - l + 1];

// initialize top of stack

int top = -1;

// push initial values of l and h to stack

stack[++top] = l;

stack[++top] = h;

// Keep popping from stack while is not empty

while (top >= 0) {

// Pop h and l

h = stack[top--];

l = stack[top--];

// Set pivot element at its correct position

// in sorted array

int p = partition(arr, l, h);

// If there are elements on left side of pivot,

// then push left side to stack

if (p - 1 > l) {

stack[++top] = l;

stack[++top] = p - 1;

}

// If there are elements on right side of pivot,

// then push right side to stack

if (p + 1 < h) {

stack[++top] = p + 1;

stack[++top] = h;

}

}

}

///////////////////////////////////////////

void CountingSort(int input\_array[], int s, int r)

{

int\* output\_array = new int[s];

int\* count\_array = new int[r];

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

count\_array[i] = 0;

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

++count\_array[input\_array[i]];

for (int i = 1; i < r; i++)

count\_array[i] = count\_array[i] + count\_array[i - 1];

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

output\_array[--count\_array[input\_array[i]]] = input\_array[i];

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

input\_array[i] = output\_array[i];

}

/////////////////////////////////

int main() {

int n = 0,N;

cout <<"Enter the Number of Array:\nPlease Enter 5 Array.\n";

cin>>N;

for (size\_t i = 1; i <= 5; i++)

{

cout << "Enter the size of array "<<i<<" : ";

cin >> n;

int\* arr = new int[n];

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

{

arr[i] = rand() ;

}

double start\_s3=clock();

insertionSort(arr, n);

double stop\_s3=clock();

cout << "time insertSort: " <<fixed<< (stop\_s3-start\_s3)/double(CLOCKS\_PER\_SEC)<< endl;

cout<<"------------------------------------------------\n";

//mergtime

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

{

arr[i] = rand() ;

}

double start\_s4=clock();

mergeSort(arr,n);

double stop\_s4=clock();

cout << "time mergeSort: " << fixed<<(stop\_s4-start\_s4)/double(CLOCKS\_PER\_SEC) << endl;

cout<<"------------------------------------------------\n";

//quicktime

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

{

arr[i] = rand() ;

}

double start\_s2=clock() ;

quickSort(arr, 0,n);

double stop\_s2=clock();

cout << "time quickSort: " << fixed<<(stop\_s2-start\_s2)/double(CLOCKS\_PER\_SEC)<< endl;

cout<<"------------------------------------------------\n";

//countingtime

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

{

arr[i] = rand() ;

}

double start\_s1=clock() ;

CountingSort(arr, 0, n);

double stop\_s1=clock();

cout << "time countSort: " <<fixed<< (stop\_s1-start\_s1)/double(CLOCKS\_PER\_SEC) << endl;

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

return 0;

}