**Lab 08**

**Name: Nimra**

**Class: BESE-7B**

**Registration No. 199281**

Task 1

Implementation of bubble, selection , insertion and merge sort

#include <iostream>

using namespace std;

void display(int array[], int length){

for(int i = 0; i<length; i++){

cout<< array[i]<<", ";

system("Pause");

}

}

int\* bubble\_sort(int array[], int length){

for (int i = length - 1; i >= 1; i--){

for (int j = 0; j <= i - 1; j++){

if (array[j + 1] < array[j]){

swap(array[j], array[j + 1]);

}

}

}

return array;

}

int\* selection\_sort(int array[], int length){

for (int i = length - 1; i >= 0; i--){

int maximum = i;

for (int j = 0; j <= i - 1; j++){

if (array[maximum] < array[j]){

maximum = j;

}

swap(array[i], array[maximum]);

}

}

return array;

}

int\* insertion\_sort(int array[], int length){

for (int i = 1; i <= length - 1; i++){

int key = array[i];

int j = i - 1;

while (j >= 0 && array[j] > key){

array[j + 1] = array[j];

j--;

}

array[j + 1] = key;

}

return array;

}

void merge(int\*, int, int,int, int);

int\* merge\_sort(int \*array, int first, int last){

int mid;

if(first< last){

mid = (first + last) /2;

merge\_sort(array, first, mid);

merge\_sort(array, mid+1, last);

merge(array, first, mid, mid+1, last);

}

return array;

}

void merge(int a[], int first, int mid, int mid01, int last)

{

int length = last-first+1;

int\* temp = new int [length];

int left = first;

int right = mid01;

for (int i = 0; i < length; ++i) {

if (left > mid)

temp[i] = a[right++];

else if (right > last)

temp[i] = a[left++];

else if (a[left] <= a[right])

temp[i] = a[left++];

else

temp[i] = a[right++];

}

for (int i=0; i< length; ++i) {

a[first++] = temp[i];}

delete temp;

}

int main(int argc, char\* args[]) {

int Arr[] = { 8, -10, 20, 74, 999, 33, -99 };

int Arr1[] = { 60, 13, 20, 1, 16, -100, 234 };

int Arr2[] = { 12, 71, 55, 44, 33, 22, 11 };

int Arr3[] = { 10, -12, 5, 7, 1, 3, 4};

cout<<"Bubble sort applied"<<endl;

display(bubble\_sort(Arr, 7),7);

cout<<"Merge sort applied"<<endl;

display(merge\_sort(Arr1,0,6),7);

cout<<"Selection sort applied"<<endl;

display(selection\_sort(Arr2,7),7);

cout<<"Insertion sort applied"<<endl;

display(insertion\_sort(Arr3,7),7);

return 0;

}

Task 2:

#include <iostream>

#include<ctime>

using namespace std;

void display(int array[], int length){

for(int i = 0; i<length; i++){

cout<< array[i]<<", ";

system("Pause");

}

}

int\* bubble\_sort(int array[], int length){

for (int i = length - 1; i >= 1; i--){

for (int j = 0; j <= i - 1; j++){

if (array[j + 1] < array[j]){

swap(array[j], array[j + 1]);

}

}

}

return array;

}

int\* selection\_sort(int array[], int length){

for (int i = length - 1; i >= 0; i--){

int maximum = i;

for (int j = 0; j <= i - 1; j++){

if (array[maximum] < array[j]){

maximum = j;

}

swap(array[i], array[maximum]);

}

}

return array;

}

int\* insertion\_sort(int array[], int length){

for (int i = 1; i <= length - 1; i++){

int key = array[i];

int j = i - 1;

while (j >= 0 && array[j] > key){

array[j + 1] = array[j];

j--;

}

array[j + 1] = key;

}

return array;

}

int main(int argc, char\* args[]) {

srand(time(NULL));

int \*a1 = new int[100], \*a2 = new int[100], \*a3 = new int[100];

int \*b1 = new int[1000], \*b2 = new int[1000], \*b3 = new int[1000];

int \*c1 = new int[10000], \*c2 = new int[10000], \*c3 = new int[10000];

int \*d1 = new int[100000], \*d2 = new int[100000], \*d3 = new int[100000];

int \*e1 = new int[1000000], \*e2 = new int[1000000], \*e3 = new int[1000000];

for (long int i = 0; i < 100; i++)

{

int temp = 1 + rand() % 100;

a1[i] = temp;

}

for (long int i = 0; i < 100; i++)

{

int temp = 1 + rand() % 100;

a2[i] = temp;

}

for (long int i = 0; i < 100; i++)

{

int temp = 1 + rand() % 100;

a3[i] = temp;

}

for (long int i = 0; i < 1000; i++)

{

int temp = 1 + rand() % 100;

b1[i] = temp;

}

for (long int i = 0; i < 1000; i++)

{

int temp = 1 + rand() % 100;

b2[i] = temp;

}

for (long int i = 0; i < 1000; i++)

{

int temp = 1 + rand() % 100;

b3[i] = temp;

}

for (long int i = 0; i < 10000; i++)

{

int temp = 1 + rand() % 100;

c1[i] = temp;

}

for (long int i = 0; i < 10000; i++)

{

int temp = 1 + rand() % 100;

c2[i] = temp;

}

for (long int i = 0; i < 10000; i++)

{

int temp = 1 + rand() % 100;

c3[i] = temp;

}

for (long int i = 0; i < 100000; i++)

{

int temp = 1 + rand() % 100;

d1[i] = temp;

}

for (long int i = 0; i < 100000; i++)

{

int temp = 1 + rand() % 100;

d2[i] = temp;

}

for (long int i = 0; i < 100000; i++)

{

int temp = 1 + rand() % 100;

d3[i] = temp;

}

for (long int i = 0; i < 1000000; i++)

{

int temp = 1 + rand() % 100;

e1[i] = temp;

}

for (long int i = 0; i < 1000000; i++)

{

int temp = 1 + rand() % 100;

e2[i] = temp;

}

for (long int i = 0; i < 1000000; i++)

{

int temp = 1 + rand() % 100;

e3[i] = temp;

}

//calculating times

int ClockStart = 0, ClockStop = 0;

ClockStart = clock();

bubble\_sort(a1, 100);

ClockStop = clock();

cout << "Time taken for bubble sort for to sort 100 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

selection\_sort(a2, 100);

ClockStop = clock();

cout << "Time taken for selection sort for to sort 100 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

insertion\_sort(a3, 100);

ClockStop = clock();

cout << "Time taken for insertion sort for to sort 100 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

cout << endl;

ClockStart = clock();

bubble\_sort(b1, 1000);

ClockStop = clock();

cout << "Time taken for bubble sort for to sort 1000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

selection\_sort(b2, 1000);

ClockStop = clock();

cout << "Time taken for selection sort for to sort 1000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

insertion\_sort(b3, 1000);

ClockStop = clock();

cout << "Time taken for insertion sort for to sort 1000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

cout << endl;

ClockStart = clock();

bubble\_sort(c1, 10000);

ClockStop = clock();

cout << "Time taken for bubble sort for to sort 10000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

selection\_sort(c2, 10000);

ClockStop = clock();

cout << "Time taken for selection sort for to sort 10000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

insertion\_sort(c3, 10000);

ClockStop = clock();

cout << "Time taken for insertion sort for to sort 10000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

cout << endl;

ClockStart = clock();

bubble\_sort(d1, 100000);

ClockStop = clock();

cout << "Time taken for bubble sort for to sort 100000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

selection\_sort(d2, 100000);

ClockStop = clock();

cout << "Time taken for selection sort for to sort 100000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

insertion\_sort(d3, 100000);

ClockStop = clock();

cout << "Time taken for insertion sort for to sort 100000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

cout << endl;

ClockStart = clock();

bubble\_sort(e1, 1000000);

ClockStop = clock();

cout << "Time taken for bubble sort for to sort 1000000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

selection\_sort(e2, 1000000);

ClockStop = clock();

cout << "Time taken for selection sort for to sort 1000000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

insertion\_sort(e3, 1000000);

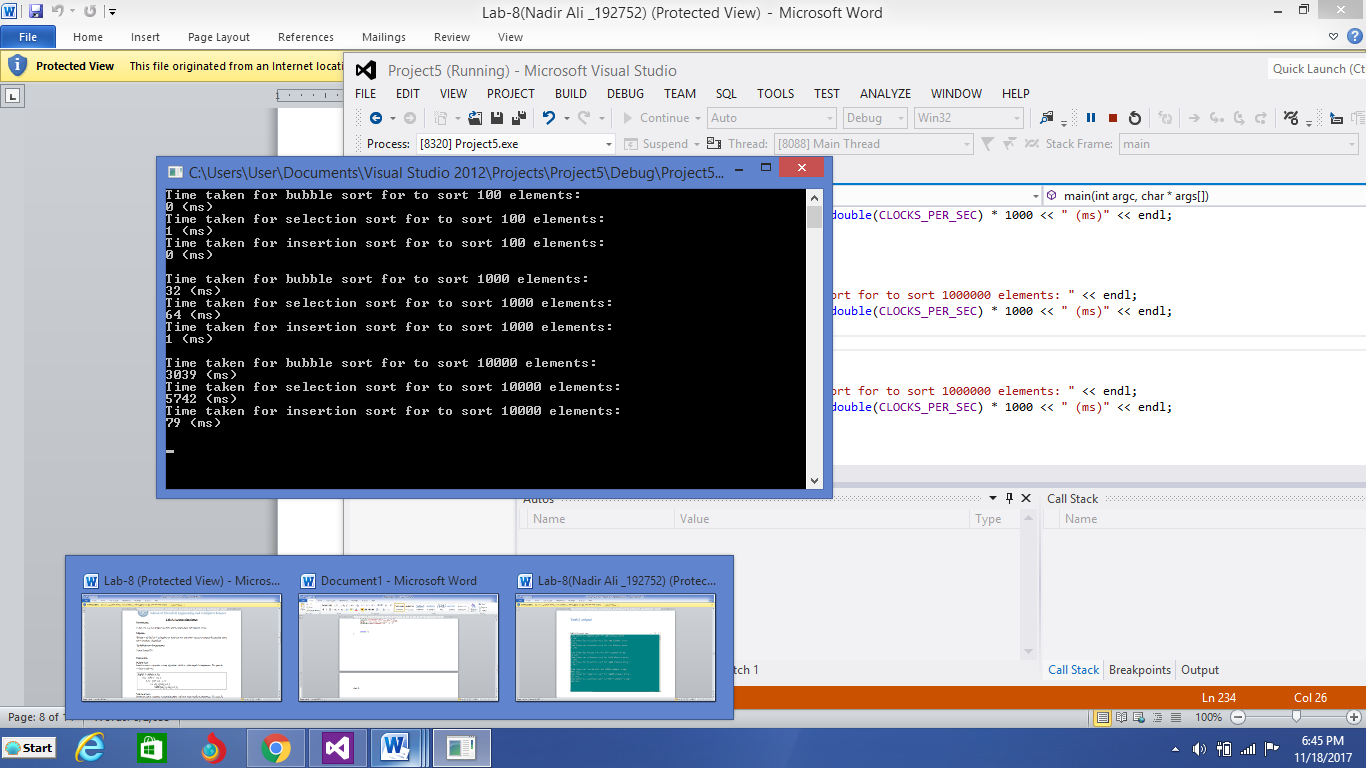
ClockStop = clock();

cout << "Time taken for insertion sort for to sort 1000000 elements: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

return 0;

}



The program was still running because it took a lot of time to execute for to sort the array of 1000000 elements, it couldn’t appear.

Order of best algorithm:

1-insertion

2-bubble

3-selection

Task 3:

#include <iostream>

#include<ctime>

using namespace std;

void display(int array[], int length){

for(int i = 0; i<length; i++){

cout<< array[i]<<", ";

system("Pause");

}

}

int\* bubble\_sort(int array[], int length){

for (int i = length - 1; i >= 1; i--){

for (int j = 0; j <= i - 1; j++){

if (array[j + 1] < array[j]){

swap(array[j], array[j + 1]);

}

}

}

return array;

}

int\* selection\_sort(int array[], int length){

for (int i = length - 1; i >= 0; i--){

int maximum = i;

for (int j = 0; j <= i - 1; j++){

if (array[maximum] < array[j]){

maximum = j;

}

swap(array[i], array[maximum]);

}

}

return array;

}

int\* insertion\_sort(int array[], int length){

for (int i = 1; i <= length - 1; i++){

int key = array[i];

int j = i - 1;

while (j >= 0 && array[j] > key){

array[j + 1] = array[j];

j--;

}

array[j + 1] = key;

}

return array;

}

int main() {

srand(time(NULL));

int a1[10000];

int a2[10000];

int\* a3 = new int [10000];

int\* d1 = new int [10000];

int\* d2 = new int [10000];

int\* d3 = new int [10000];

for (int i =0; i < 10000; i++){

a1[i] = i;

a2[i] = i;

a3[i] = i;

d1[i] = 10000-i;

d2[i] = 10000-i;

d3[i] = 10000-i;

}

int ClockStart = 0, ClockStop = 0;

ClockStart = clock();

bubble\_sort(a1, 10000);

ClockStop = clock();

cout << "Time taken for bubble sort for (sorted) 10000 element array: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

selection\_sort(a2, 10000);

ClockStop = clock();

cout << "Time taken for selection sort (sorted) for 10000 element array: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

insertion\_sort(a3, 10000);

ClockStop = clock();

cout << "Time taken for insertion sort for (sorted) 10000 element array: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

bubble\_sort(d1, 10000);

ClockStop = clock();

cout << "Time taken for bubble sort for (unsorted) 10000 element array: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

selection\_sort(d2, 10000);

ClockStop = clock();

cout << "Time taken for selection sort (unsorted) for 10000 element array: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

ClockStart = clock();

insertion\_sort(d3, 10000);

ClockStop = clock();

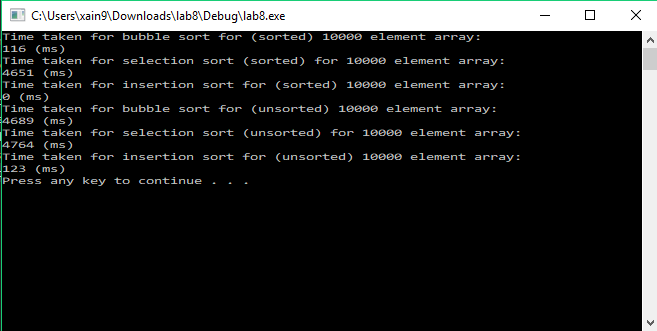
cout << "Time taken for insertion sort for (unsorted) 10000 element array: " << endl;

cout << (ClockStop - ClockStart) / double(CLOCKS\_PER\_SEC) \* 1000 << " (ms)" << endl;

system("pause");

return 0;

}



Output: Bubble sort has most variation. Selection sort has least variation.

Unsorted = decending