National University of Computer and Emerging Sciences



Name: Muhammad Suleman

Roll #: 22F-3350

Lab#: 02

Section: BCS-4B

# Problem 01:

#include <iostream>

#include <string>

using namespace std;

class Record

{

private:

string name;

int regNum;

int marks[3];

int totalMarks;

public:

void setData();

void calTotalMarks();

void getData();

};

int main()

{

int N;

cout << "Enter the number of students: ";

cin >> N;

Record\* students = new Record[N];

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

{

cout << "Enter details for student " << i + 1 << endl;

students[i].setData();

students[i].calTotalMarks();

}

cout << "\nStudent Records:" << endl;

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

{

students[i].getData();

}

cout << "Time: " << \_\_TIME\_\_ << endl;

delete[] students;

students = nullptr;

system("pause");

return 0;

}

void Record::setData()

{

cout << "Enter name: ";

cin.ignore();

getline(cin, name);

cout << "Enter registration number: ";

cin >> regNum;

cout << "Enter marks for three subjects: ";

for (int i = 0; i < 3; i++)

{

cin >> marks[i];

}

}

void Record::getData()

{

cout << "Name: " << name << endl;

cout << "Registration Number: " << regNum << endl;

cout << "Marks for three subjects: ";

for (int i = 0; i < 3; i++)

{

cout << marks[i] << " ";

}

cout << endl;

cout << "Total Marks: " << totalMarks << endl;

}

void Record::calTotalMarks()

{

totalMarks = 0;

for (int i = 0; i < 3; i++)

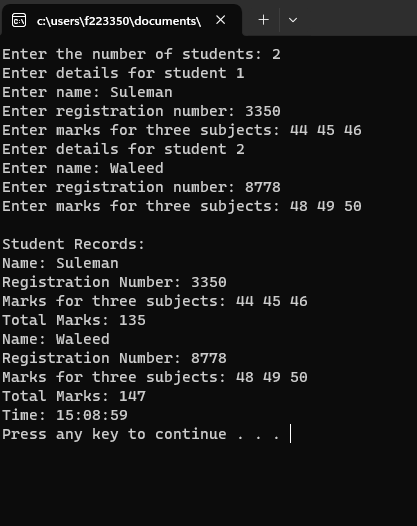
{

totalMarks += marks[i];

}

}

## Output:



# Problem 02:

#include <iostream>

using namespace std;

int lengthOfStr(char\* str)

{

const char\* ptr;

int len = 0;

for (ptr = str; \*ptr != '\0'; ptr++)

{

len++;

}

return len;

}

char\* concatenateStr(char\* str1, char\* str2)

{

int len1 = 0;

int len2 = 0;

// finding lengths of strings

len1 = lengthOfStr(str1);

len2 = lengthOfStr(str2);

char\* result = new char[len1 + len2 + 1];

// concatenation of strings

int i = 0;

for (; i < len1; i++)

{

result[i] = str1[i];

}

for (int j = 0; j < len2; j++, i++)

{

result[i] = str2[j];

}

result[i] = '\0';

return result;

}

void reverseStr(char\* str)

{

char \*start = str;

char \*end = str;

while (\*end != '\0')

{

end++;

}

end--;

while (start < end)

{

char temp = \*start;

\*start = \*end;

\*end = temp;

start++;

end--;

}

}

int main()

{

char\* str1 = "Hello";

cout << "String1: " << str1 << endl;

cout << "Length of String1: " << lengthOfStr(str1) << endl;

char\* str2 = "World";

cout << "String2: " << str2 << endl;

cout << "Length of String2: " << lengthOfStr(str2) << endl;

char\* str3 = concatenateStr(str1, str2);

cout << "String3: " << str3 << endl;

cout << "Length of String3: " << lengthOfStr(str3) << endl;

cout << "Time: " << \_\_TIME\_\_ << endl;

delete[] str3;

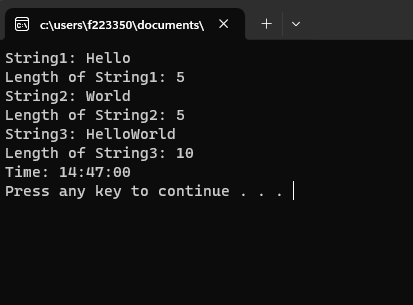
str3 = nullptr;

system("pause");

return 0;

}

## Output:



# Problem 03:

#include <iostream>

using namespace std;

void findIntersection(int array1[], int size1, int array2[], int size2, int result[], int& size3)

{

size3 = 0;

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size2; j++)

{

if (array1[i] == array2[j])

{

bool isDuplicate = false;

for (int k = 0; k < size3; k++)

{

if (array1[i] == result[k])

{

isDuplicate = true;

break;

}

}

if (!isDuplicate)

{

result[size3++] = array1[i];

}

break;

}

}

}

}

int main()

{

int array1[] = { 1, 7, 3, 4, 5, 6, 9 };

int array2[] = { 5, 3, 5, 7 };

int array3[100];

int size1 = 7;

int size2 = 4;

int size3;

findIntersection(array1, size1, array2, size2, array3, size3);

cout << "Intersection: ";

for (int i = 0; i < size3; i++)

{

cout << array3[i] << " ";

}

cout << endl;

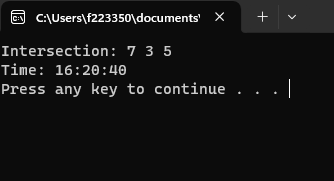
cout << "Time: " << \_\_TIME\_\_ << endl;

system("pause");

return 0;

}

## Output:



# Problem 04:

#include <iostream>

using namespace std;

int main()

{

int\* ptr = new int;

\*ptr = 42;

delete ptr;

// this statement will throw dangling ptr exception

//\*ptr = 20;

if (ptr == nullptr)

{

cout << "Not a dangling pointer" << endl;

}

else

{

cout << "Dangling pointer" << endl;

}

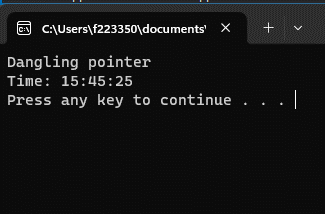
cout << "Time: " << \_\_TIME\_\_ << endl;

system("pause");

return 0;

}

## Output:



# Problem 05:

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <chrono>

using namespace std;

using namespace std::chrono;

void bubbleSort(int arr[], int n)

{

bool swapped;

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

{

swapped = false;

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

{

if (arr[j] > arr[j + 1])

{

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

swapped = true;

}

}

if (!swapped)

{

break;

}

}

}

void insertionSort(int arr[], int n)

{

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

{

int key = arr[i];

int j = i - 1;

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

{

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

j--;

}

arr[j + 1] = key;

}

}

int main()

{

srand(time(0));

int arr1[1000];

int arr2[1000];

int arr3[50];

int arr4[50];

for (int i = 0; i < 50; i++)

{

arr3[i] = rand() % 100;

arr4[i] = rand() % 100;

}

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

{

arr1[i] = 1000 - i;

arr2[i] = 1000 - i;

}

auto startBubbleSort = high\_resolution\_clock::now();

bubbleSort(arr1, 1000);

auto endBubbleSort = high\_resolution\_clock::now();

auto bubbleSortDuration = duration\_cast<nanoseconds>(endBubbleSort - startBubbleSort);

auto startInsertionSort = high\_resolution\_clock::now();

insertionSort(arr2, 1000);

auto endInsertionSort = high\_resolution\_clock::now();

auto insertionSortDuration = duration\_cast<nanoseconds>(endInsertionSort - startInsertionSort);

cout << "Execution time for Bubble Sort: " << bubbleSortDuration.count() << " nanoseconds" << endl;

cout << "Execution time for Insertion Sort: " << insertionSortDuration.count() << " nanoseconds" << endl;

startBubbleSort = high\_resolution\_clock::now();

bubbleSort(arr3, 50);

endBubbleSort = high\_resolution\_clock::now();

bubbleSortDuration = duration\_cast<nanoseconds>(endBubbleSort - startBubbleSort);

startInsertionSort = high\_resolution\_clock::now();

insertionSort(arr4, 50);

endInsertionSort = high\_resolution\_clock::now();

insertionSortDuration = duration\_cast<nanoseconds>(endInsertionSort - startInsertionSort);

cout << "Execution time for Bubble Sort: " << bubbleSortDuration.count() << " nanoseconds" << endl;

cout << "Execution time for Insertion Sort: " << insertionSortDuration.count() << " nanoseconds" << endl;

cout << "Time: " << \_\_TIME\_\_ << endl;

return 0;

}

### Bubble Sort

Best Case Time Complexity: O(n)

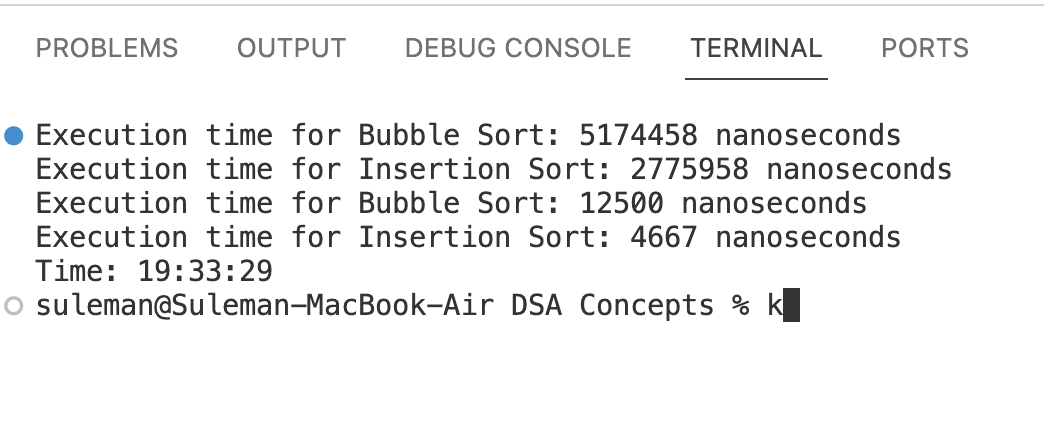
Worst Case Time Complexity: O(n^2)

### Insertion Sort

Best Case Time Complexity: O(n^2)

Worst Case Time Complexity: O(n^2)

## Output:



# Problem 06:

#include <iostream>

#include <string>

using namespace std;

class Student

{

private:

string name;

string rollNum;

int marks;

public:

// method getters and setters

void setName(string name)

{

this->name = name;

}

void setRollNum(string rollNum)

{

this->rollNum = rollNum;

}

void setMarks(int marks)

{

this->marks = marks;

}

string getName()

{

return name;

}

string getRollNum()

{

return rollNum;

}

int getMarks()

{

return marks;

}

};

int main()

{

int numOfStudents;

cout << "Enter the number of students: ";

cin >> numOfStudents;

Student \*students = new Student[numOfStudents];

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

{

string name;

string rollNum;

int marks;

cout << "\nEnter details for student " << i + 1 << endl;

cout << "Name: ";

cin.ignore();

getline(cin, name);

cout << "Roll Number: ";

getline(cin, rollNum);

cout << "Marks: ";

cin >> marks;

students[i].setName(name);

students[i].setRollNum(rollNum);

students[i].setMarks(marks);

}

cout << "\nStudent Information:\n";

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

{

cout << "Student " << i + 1 << ":\n";

cout << "Name: " << students[i].getName() << endl;

cout << "Roll Number: " << students[i].getRollNum() << endl;

cout << "Marks: " << students[i].getMarks() << endl << endl;

}

delete[] students;

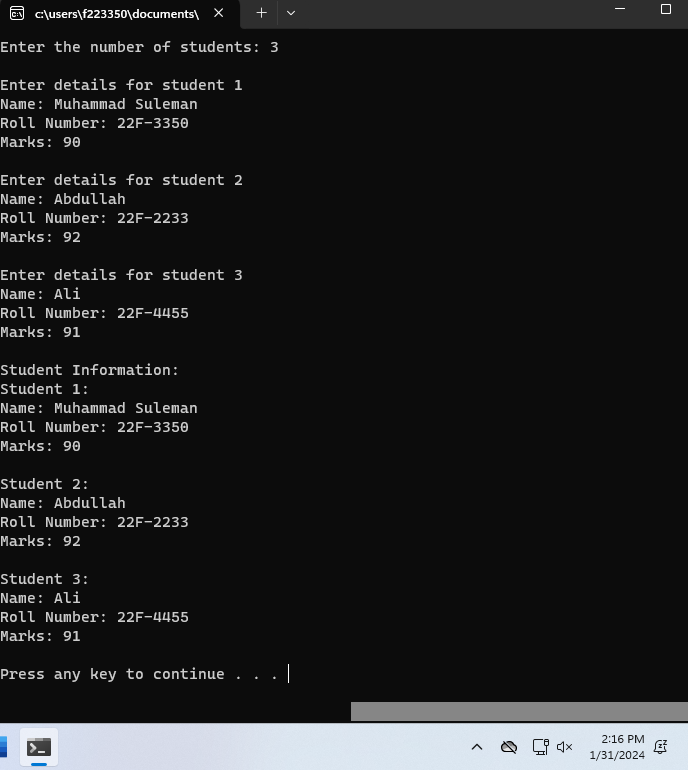
students = nullptr;

system("pause");

return 0;

}

## Output:



# Problem 07:

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <chrono>

using namespace std;

using namespace std::chrono;

int findMax(int arr[], int size)

{

int max = arr[0];

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

{

if (arr[i] > max)

{

max = arr[i];

}

}

return max;

}

int findMin(int arr[], int size)

{

int min = arr[0];

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

{

if (arr[i] < min)

{

min = arr[i];

}

}

return min;

}

int main()

{

srand(time(0));

int arr1[1000];

int arr2[1000];

int arr3[50];

int arr4[50];

int max;

int min;

for (int i = 0; i < 50; i++)

{

arr3[i] = rand() % 100;

arr4[i] = rand() % 100;

}

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

{

arr1[i] = 1000 - i;

arr2[i] = 1000 - i;

}

cout << "\nArray 1 of size 1000\n" << endl;

auto startTime = high\_resolution\_clock::now();

max = findMax(arr1, 1000);

auto endTime = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(endTime - startTime);

cout << "Duration of finding Max: " << duration.count() << " nanoseconds" << endl;

startTime = high\_resolution\_clock::now();

max = findMin(arr1, 1000);

endTime = high\_resolution\_clock::now();

duration = duration\_cast<nanoseconds>(endTime - startTime);

cout << "Duration of finding Min: " << duration.count() << " nanoseconds" << endl;

cout << "\nArray 2 of size 1000\n" << endl;

startTime = high\_resolution\_clock::now();

max = findMax(arr2, 1000);

endTime = high\_resolution\_clock::now();

duration = duration\_cast<nanoseconds>(endTime - startTime);

cout << "Duration of finding Max: " << duration.count() << " nanoseconds" << endl;

startTime = high\_resolution\_clock::now();

max = findMin(arr2, 1000);

endTime = high\_resolution\_clock::now();

duration = duration\_cast<nanoseconds>(endTime - startTime);

cout << "Duration of finding Min: " << duration.count() << " nanoseconds" << endl;

cout << "\nArray 3 of size 50\n" << endl;

startTime = high\_resolution\_clock::now();

max = findMax(arr3, 50);

endTime = high\_resolution\_clock::now();

duration = duration\_cast<nanoseconds>(endTime - startTime);

cout << "Duration of finding Max: " << duration.count() << " nanoseconds" << endl;

startTime = high\_resolution\_clock::now();

max = findMin(arr3, 50);

endTime = high\_resolution\_clock::now();

duration = duration\_cast<nanoseconds>(endTime - startTime);

cout << "Duration of finding Min: " << duration.count() << " nanoseconds" << endl;

cout << "\nArray 4 of size 50\n" << endl;

startTime = high\_resolution\_clock::now();

max = findMax(arr4, 50);

endTime = high\_resolution\_clock::now();

duration = duration\_cast<nanoseconds>(endTime - startTime);

cout << "Duration of finding Max: " << duration.count() << " nanoseconds" << endl;

startTime = high\_resolution\_clock::now();

max = findMin(arr4, 50);

endTime = high\_resolution\_clock::now();

duration = duration\_cast<nanoseconds>(endTime - startTime);

cout << "Duration of finding Min: " << duration.count() << " nanoseconds" << endl;

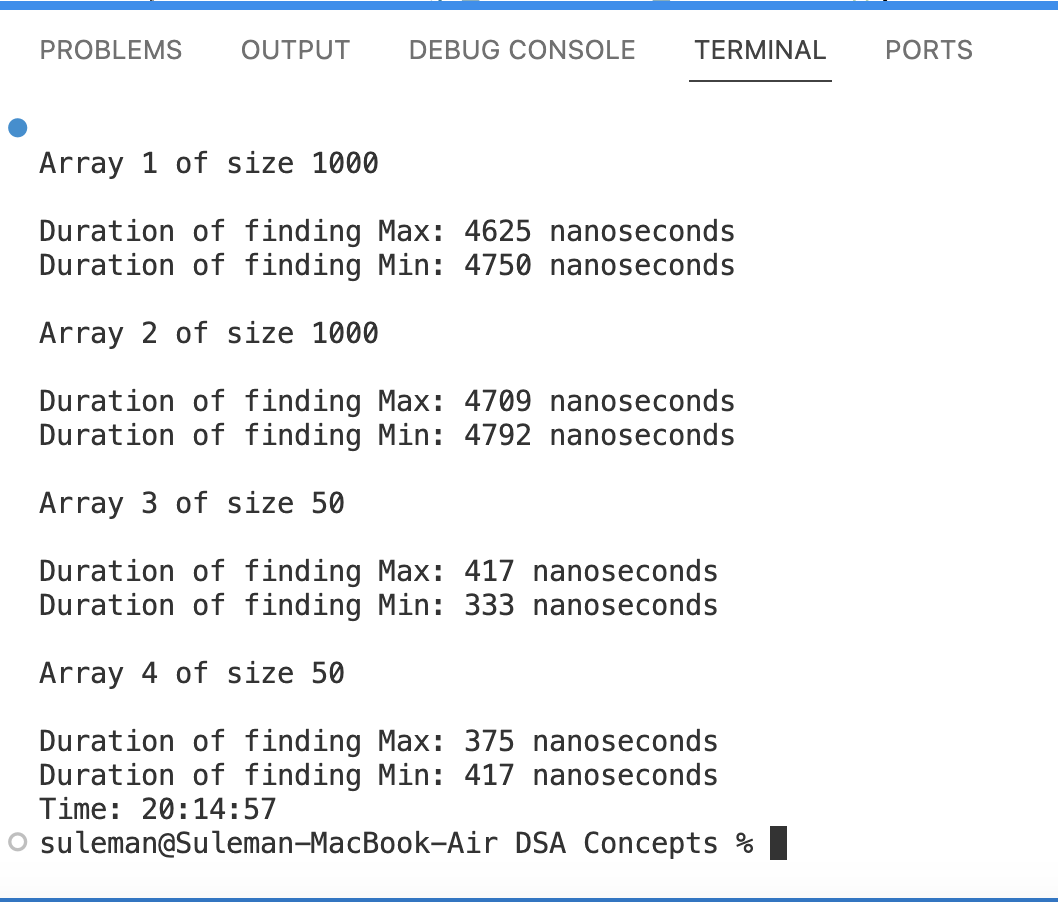
cout << "Time: " << \_\_TIME\_\_ << endl;

return 0;

}

## Output:

Time Complexity = O(1) because we know the sizes of array that are 1000 and 50 respectively.



# Problem 08:

a)

// previously it was uninitialized pointer

int \*number;

int num = 5;

number = &num;

cout << number << endl;

b)

// it is needed to be type casted

double \*realptr;

long \*integerPtr;

integerPtr = (long\*)realptr;

c)

// made y also a pointer

int \*x, \*y;

x = y;

d)

char s[] = "this is a character array";

// array name is a constant pointer so we cannot modify it

for (char\* ptr = s; \*ptr != '\0'; ++ptr)

cout << \*ptr << "'";

e)

short \*numptr, result;

void \*genericPtr = numptr;

// genericPtr needed to be type casted and than dereferenced

result = \*(short\*)genericPtr + 7;

f)

double xN = 19.34;

// give address by using & operator

double \*xPtr = &xN;

cout << xPtr << endl;