Samuel Larson

Project 9

11/19/18

The objective of this program is to record the time it takes for an insertion sort, a merge sort, and a quicksort to sort an identical array and display the time each of them takes. The program should let the user enter how many numbers should go in the array and a random seed for the program. The program should also give the option to display the array after each swap or not. When the program is done it should display how many swaps each sort needed to sort the array.

The program will prompt the user to enter how many numbers to generate and sort between 1 and 5000 and an integer value for a seed. The user will also be prompted to enter Y or N on whether to print the values of the array after each exchange

The program should generate 3 identical arrays then call 3 functions to perform an insertion, merge and quick sort on them. The program needs a count value for each of the functions to keep track of each exchange the sort makes. The program will display the amount of exchanges each sort made.

//Samuel Larson

//Project 9

//11/19/18

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int insertionsort(int array[], int, char);

int mergesort(int array[], int, int, int, char);

int merge(int array[], int, int, int, int, char);

int quicksort(int array[], int, int, int, char);

int partition(int array[], int, int, int, int&, int&, int, char);

void exchange(int array[], int, int);

void printarray(int array[], int);

int main()

{

int nums, seed, inserttime, mergetime, quicktime, randnum;

char print;

inserttime = 0;

mergetime = 0;

quicktime = 0;

do

{

cout << "Enter the number of values to generate a sort, between 1 and 5000: ";

cin >> nums;

}while(nums<1 || nums>5000);

cout << "Enter an integer seed value: ";

cin >> seed;

do

{

cout << "Print the values? Y/N: ";

cin >> print;

}while(print != 'Y' && print != 'N' && print != 'y' && print != 'n');

srand(seed);

int insert[nums];

int merge[nums];

int quick[nums];

for(int i=0; i<nums; i++)//creates 3 identical arrays

{

randnum = rand()%100;

insert[i] = randnum;

merge[i] = randnum;

quick[i] = randnum;

}

if(print == 'y' || print == 'Y')

{

cout << "Original array: ";

printarray(insert, nums);

}

if(print == 'y' || print == 'Y')

cout << "Insertion Sort: " << endl;

inserttime = insertionsort(insert, nums, print);//calls and records time of insertion sort

if(print == 'y' || print == 'Y')

cout << "Merge Sort: " << endl;

mergetime = mergesort(merge, 0, nums-1, nums, print);//calls and records time of merge sort

if(print == 'y' || print == 'Y')

cout << "Quick Sort: " << endl;

quicktime = quicksort(quick, 0, nums-1, nums, print);//calls and records time of quick sort

cout << "Insertion Sort count = " << inserttime << endl;

cout << "Merge Sort count = " << mergetime << endl;

cout << "Quick Sort count = " << quicktime << endl;

return 0;

}

//pre: an unsorted array post: a sorted array

int insertionsort(int a[], int length, char print)

{

int temp, count;

count = 0;

temp = a[0];

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

{

temp = a[i];

int j = 0;

for(j = i; j > 0; j--)

if(temp < a[j - 1])

a[j] = a[j - 1];

else break;

a[j] = temp;

if(print == 'y' || print == 'Y')

printarray(a,length);

count++;

}

return count;

}

//pre: an unsorted array post: a sorted array

int mergesort(int a[], int low, int high, int nums, char print)

{

int mid, count = 0;

if(low<high)

{

mid = low + (high-low)/2;

count = count + mergesort(a, low, mid, nums, print);

count = count + mergesort(a, mid+1, high, nums, print);

count = count + merge(a, low, mid, high, nums, print);

}

return count;

}

//pre: 2 chunks of an array post: merged 2 parts of an array in order

int merge(int a[], int low, int mid, int high, int nums, char print)

{

int l[nums];

int count = 0;

int first1, last1, first2, last2;

first1=low;

last1=mid;

first2=mid+1;

last2=high;

int index = first1;

while(first1<=last1 && first2<=last2)

{

if(a[first1]<=a[first2])

{

l[index] = a[first1];

first1++;

count++;

}

else

{

l[index] = a[first2];

first2++;

}

index++;

}

while(first1<=last1)

{

l[index]=a[first1];

first1++;

index++;

}

while(first2<=last2)

{

l[index]=a[first2];

first2++;

index++;

}

for(index=low; index<=high; index++)

a[index]=l[index];

if(print == 'y' || print == 'Y')

printarray(a, nums);

return count;

}

//pre: an unsorted array post: a sorted array

int quicksort(int a[], int low, int high, int nums, char print)

{

int pivot;

int count = 0;

int lasts1, firsts3;

if(low<high)

{

pivot = a[low];

count = count + partition(a, low, high, pivot, lasts1, firsts3, nums, print);

count = count + quicksort(a, low, lasts1, nums, print);

count = count + quicksort(a, firsts3, high, nums, print);

}

return count;

}

//pre:pivot and unsorted array post: sorted array

int partition(int a[], int low, int high, int pivot, int& lasts1, int& firsts3, int nums, char print)

{

int firstu = low+1;

int count = 0;

lasts1 = low-1;

firsts3 = high+1;

while(firstu < firsts3)

{

if(a[firstu]<pivot)//s1

{

++lasts1;

exchange(a, firstu, lasts1);

if(print == 'y' || print == 'Y')

printarray(a, nums);

count++;

++firstu;

}

else if(a[firstu] == pivot)//s2

++firstu;

else//s3

{

--firsts3;

exchange(a, firstu, firsts3);

if(print == 'y' || print == 'Y')

printarray(a, nums);

count++;

}

}

return count;

}

//pre: 2 entries in an array post: entries swapped

void exchange(int a[], int first, int second)

{

int holder;

holder = a[first];

a[first] = a[second];

a[second] = holder;

return;

}

//pre: an array to print post: a printed array

void printarray(int a[],int length)

{

if(length > 0)

cout << a[0];

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

{

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

}

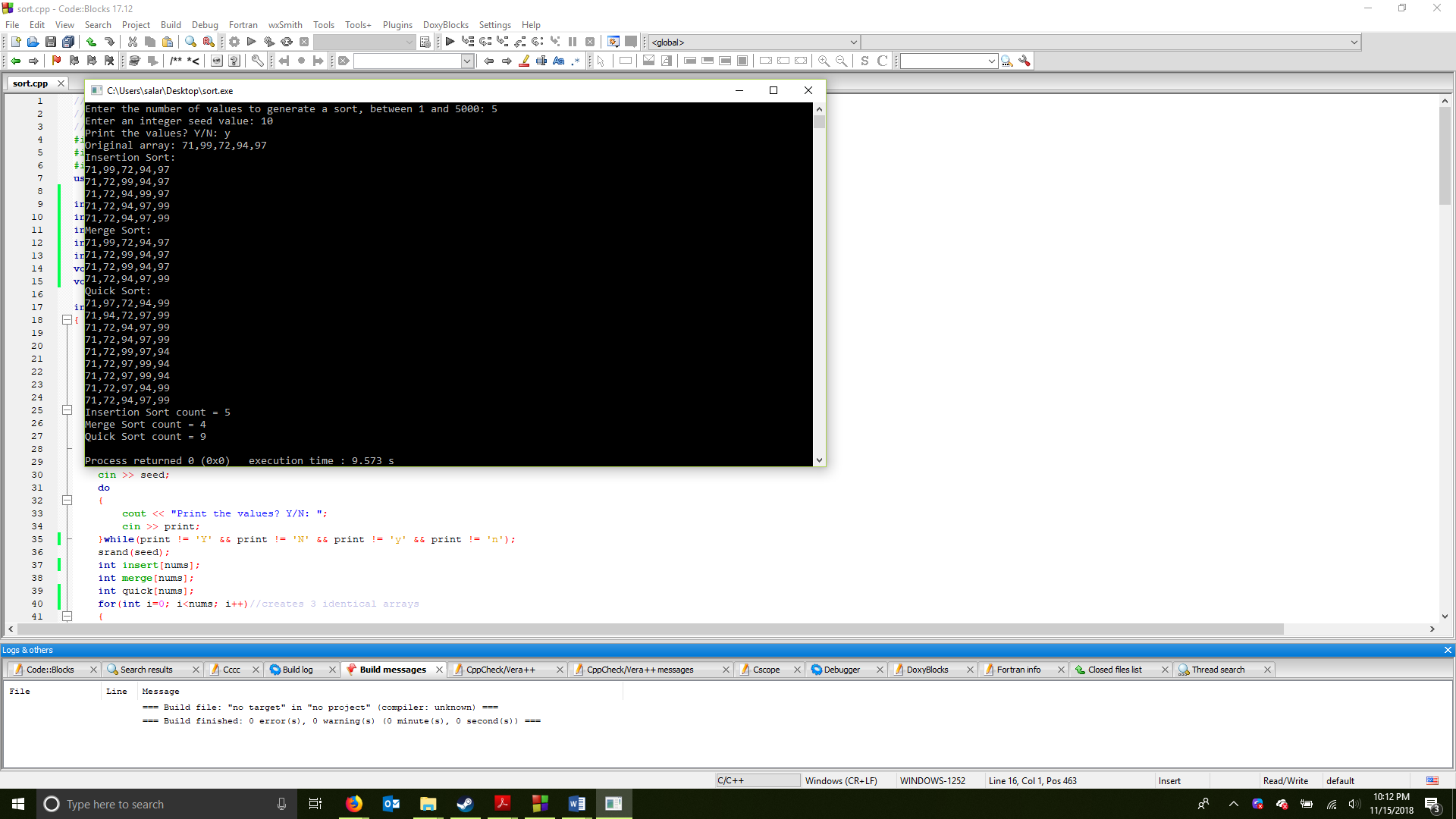
cout << endl;

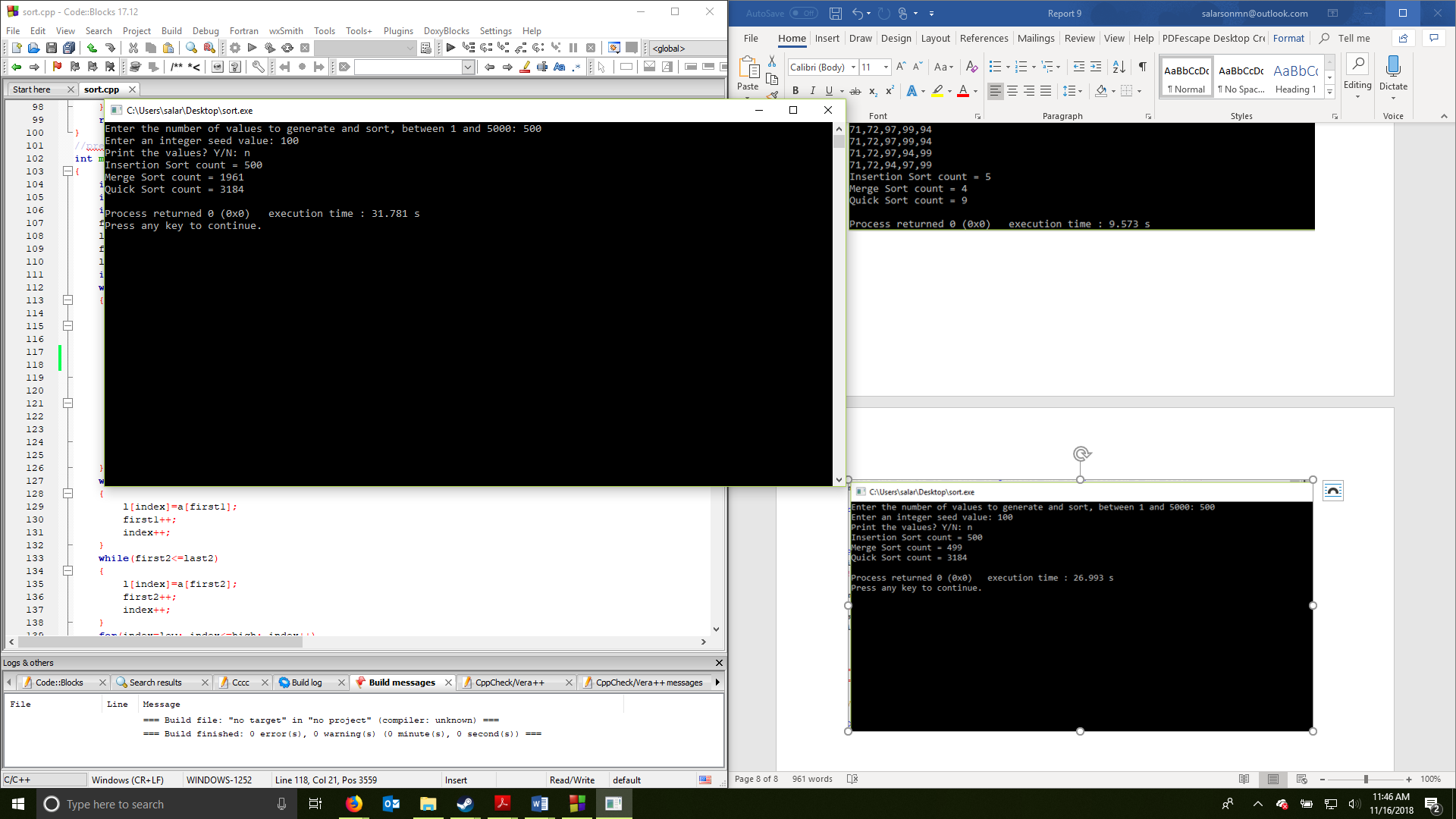
}

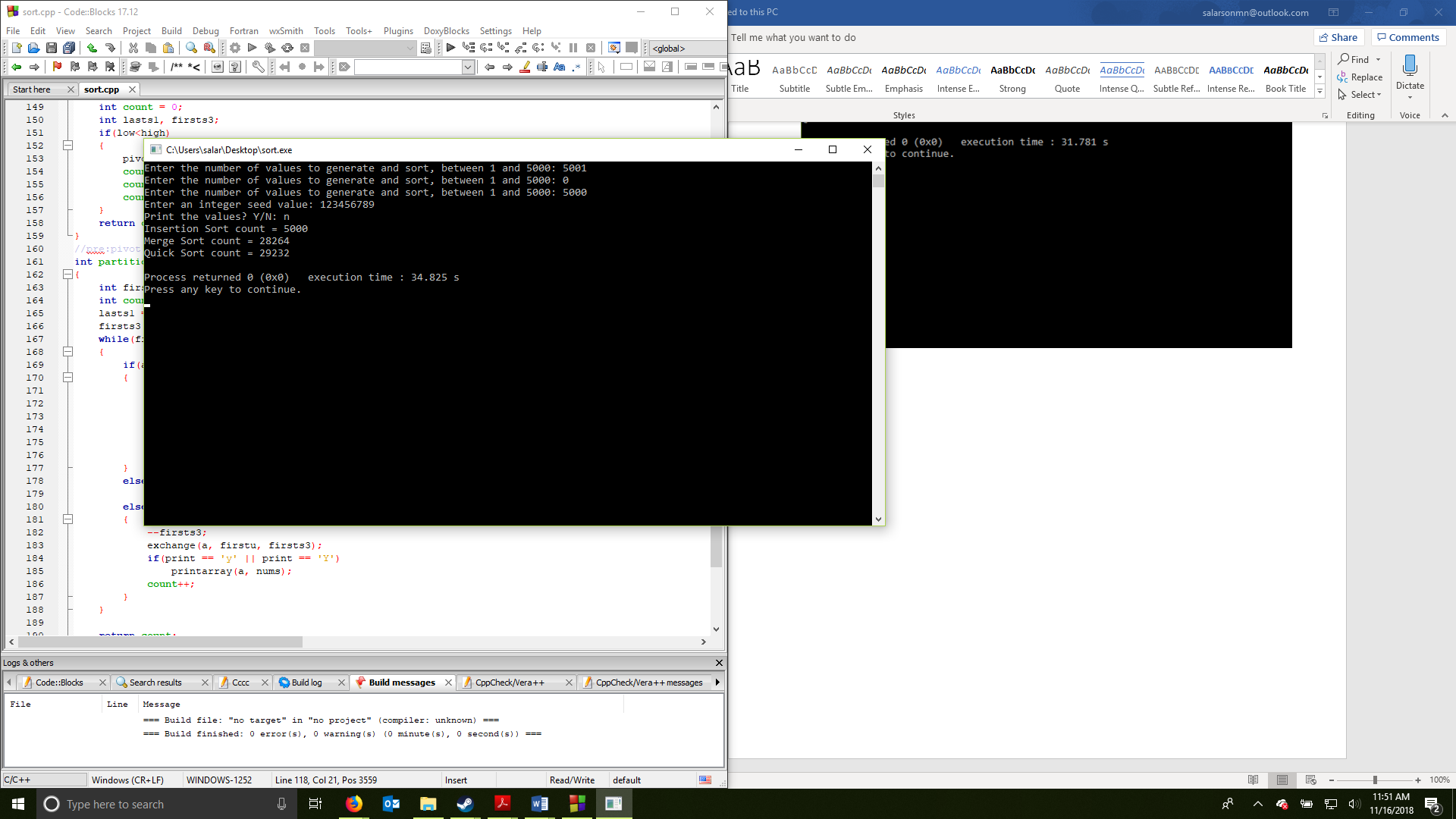
Samuel Larson

Project 9

11/19/18







Samuel Larson

Project 9

11/19/18

User Document

This program will prompt you to enter an amount of numbers the program will sort. You will need to enter an integer between 1 and 5000 into the terminal. The program will ask you for a seed and if you want to print each update to the array. Enter Y or y to print the numbers or N or n to not print the numbers. The program will then sort out the numbers in 3 different ways. This process will take longer depending on the amount of numbers to sort and if the user requested them to be printed.

Example of a possible output:  
  
Enter the number of values to generate and sort, between 1 and 5000: 4

Enter an integer seed value: 81

Print the values? Y/N: y

Original array: 3,72,44,53

Insertion Sort:

3,72,44,53

3,44,72,53

3,44,53,72

3,44,53,72

Merge Sort:

3,72,44,53

3,72,44,53

3,44,53,72

Quick Sort:

3,53,44,72

3,44,53,72

3,44,53,72

3,44,72,53

3,44,72,53

3,44,53,72

Insertion Sort count = 4

Merge Sort count = 3

Quick Sort count = 6

Process returned 0 (0x0) execution time : 9.209 s

Press any key to continue.

Samuel Larson

Project 9

11/19/18

Summary

The objective of this project was to create a program that can record how many swaps it takes for an insertion sort, merge sort or quick sort to sort an array of random integers generated by the program. The program required the user to be able to enter the amount of numbers the sorting algorithms would sort and a random integer to use as a seed. The program also had to ask the user if they wanted the program to print the array of integers after swaps.

In a big O sense, the program the program demonstrates how the Insertion sort highly depends on how many numbers are entered with a big O of n. Merge Sort and Quick sort were shown to have a possible big O of log n and a worst case possibility of n^2.

This program could have been made better by allowing the user to run the simulation multiple times and allowing the user to enter specific numbers instead of randomly generated ones. Or display which numbers quick sort are the pivot and what numbers are the chunks in the merge sort.