Samuel Larson

Project 11

12/10/18

The objective of this program was to add an additional sorting algorithm to project 9. The program has to measure the amount of time it takes for an insert sort, a merge sort, a quicksort, and a heap sort to sort an array. The program should allow the user to enter how many numbers should be in the array and a random seed to generate the numbers from. The program should also allow the user to choose if the program displays the array every time values are swapped. Once the program is done it should display how many swaps it needed to sort each array.

The program will prompt the user to enter how many numbers it should generate and sort between 1 and 5000 and an integer value for a seed. The program should then prompt the user for Y or N on whether the program should print the array after each swap.

The program should generate 4 identical arrays then call 4 functions to perform a insertion sort, merge sort, quick sort, and heap sort on each of them. Each function will return a count value that keeps track of how many times numbers are swapped.

Each sort will have the array, the size of the array the lowest number and if it should print out the array or not. The heap sort finds the biggest values and moves them towards the top of the tree. Then groups up the smaller values towards the bottom of the tree.

//Samuel Larson

//Project 11

//12/10/18

#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 heapsort(int array[], int n, int, char);

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

int main()

{

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

char print;

inserttime = 0;

mergetime = 0;

quicktime = 0;

heaptime = 0;

do

{

cout << "Enter the number of values to generate and 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];

int heap[nums];

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

{

randnum = rand()%100;

insert[i] = randnum;

merge[i] = randnum;

quick[i] = randnum;

heap[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

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

cout << "Heap Sort: " << endl;

heaptime = heapsort(heap, nums, nums, print);

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

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

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

cout << "Heap Sort count = " << heaptime << 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;

}

//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;

}

//pre: unsorted array post: sorted array

int heapsort(int a[], int n, int nums, char print)

{

int count = 0;

for(int i = (n/2)-1; i >= 0; i--)

count = count + reheap(a, n, i, nums, print);

for(int i = n-1; i>=0; i--)

{

exchange(a, 0, i);

count++;

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

printarray(a, nums);

count = count + reheap(a, i, 0, nums, print);

}

return count;

}

//pre: chunk of numbers that is unsorted post: reheaped chunk of numbers

int reheap(int a[], int n, int i, int nums, char print)

{

int count = 0;

int largest = i;

int l = (2\*i)+1;

int r = (2\*i)+2;

if(l < n && a[l] > a[largest]) //if left child is larger than the root

largest = l;

if(r < n && a[r] > a[largest])//if right child is larger than the root and left child

largest = r;

if(largest != i)//if largest is not the root

{

exchange(a, i, largest);

count++;

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

printarray(a, nums);

count = count + reheap(a, n, largest, nums, print);

}

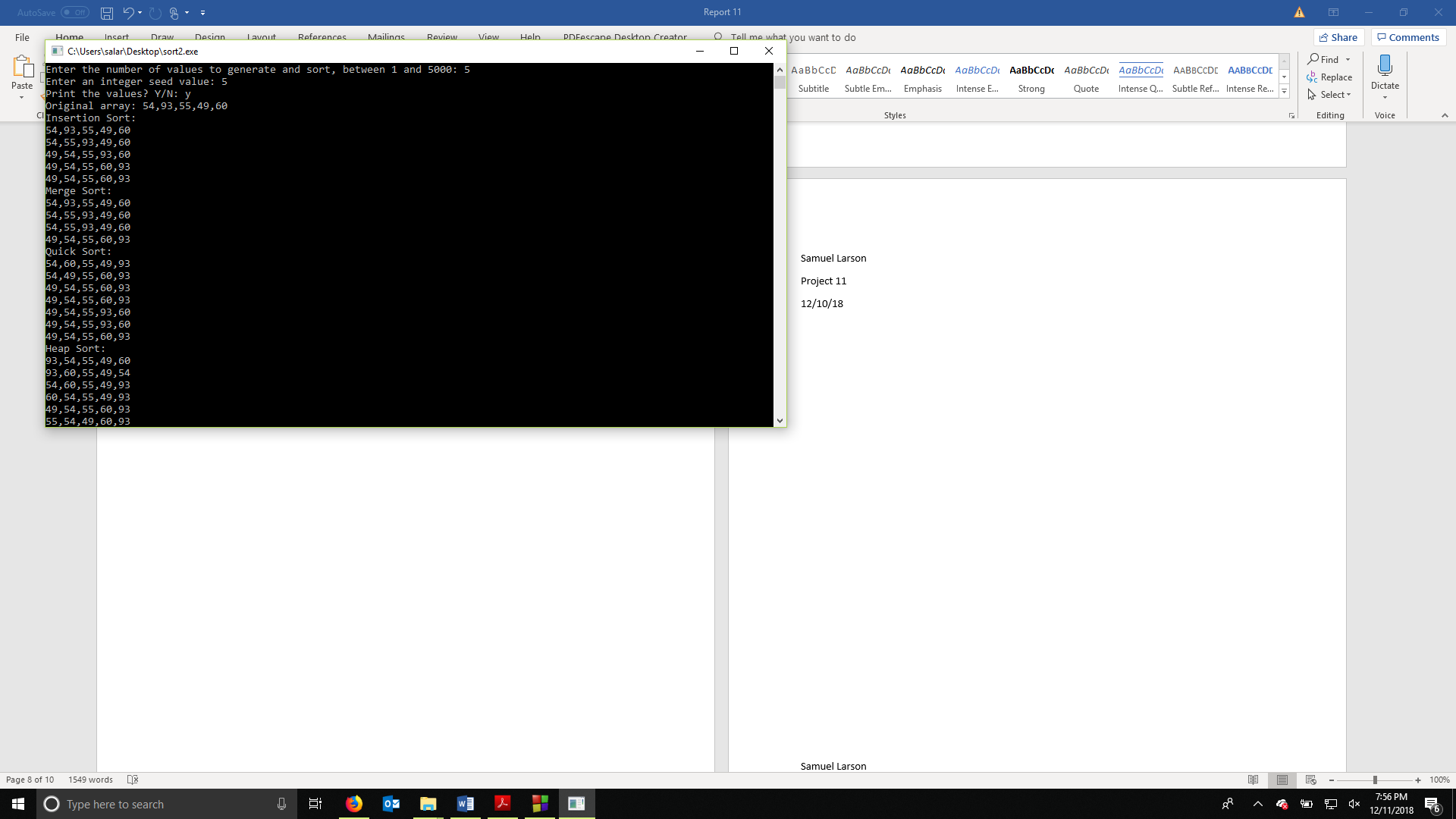
return count;

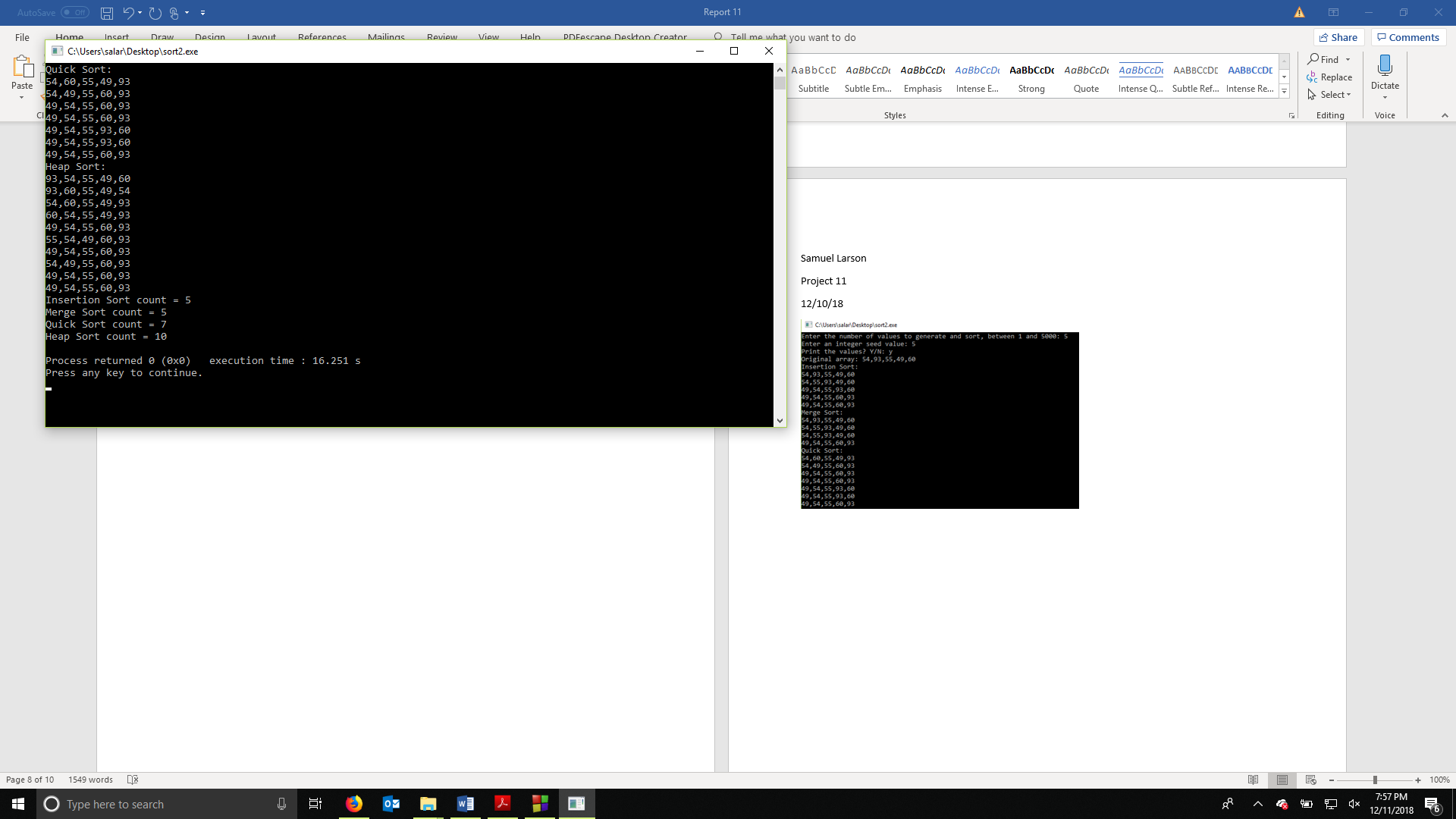
}

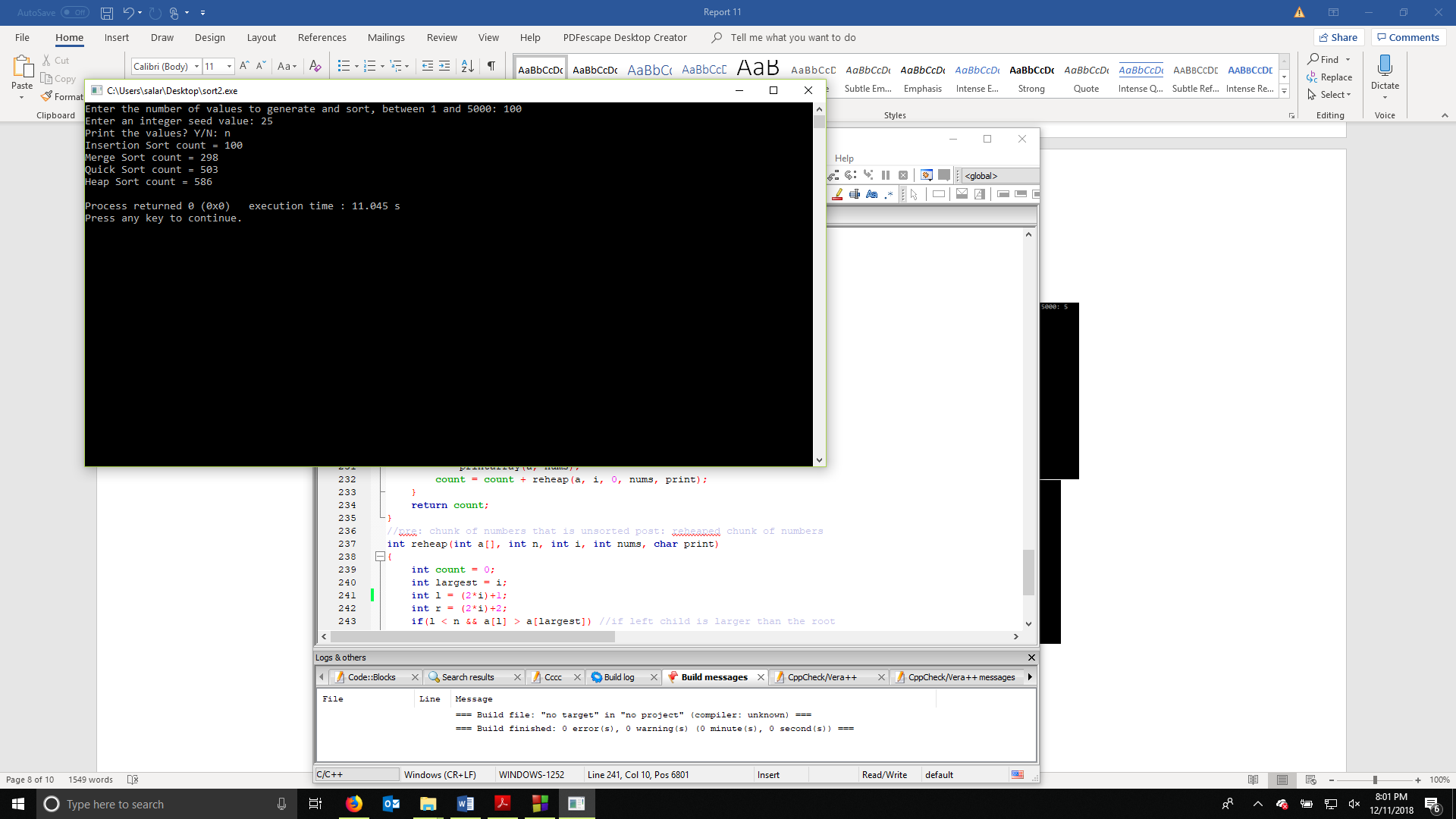
Samuel Larson

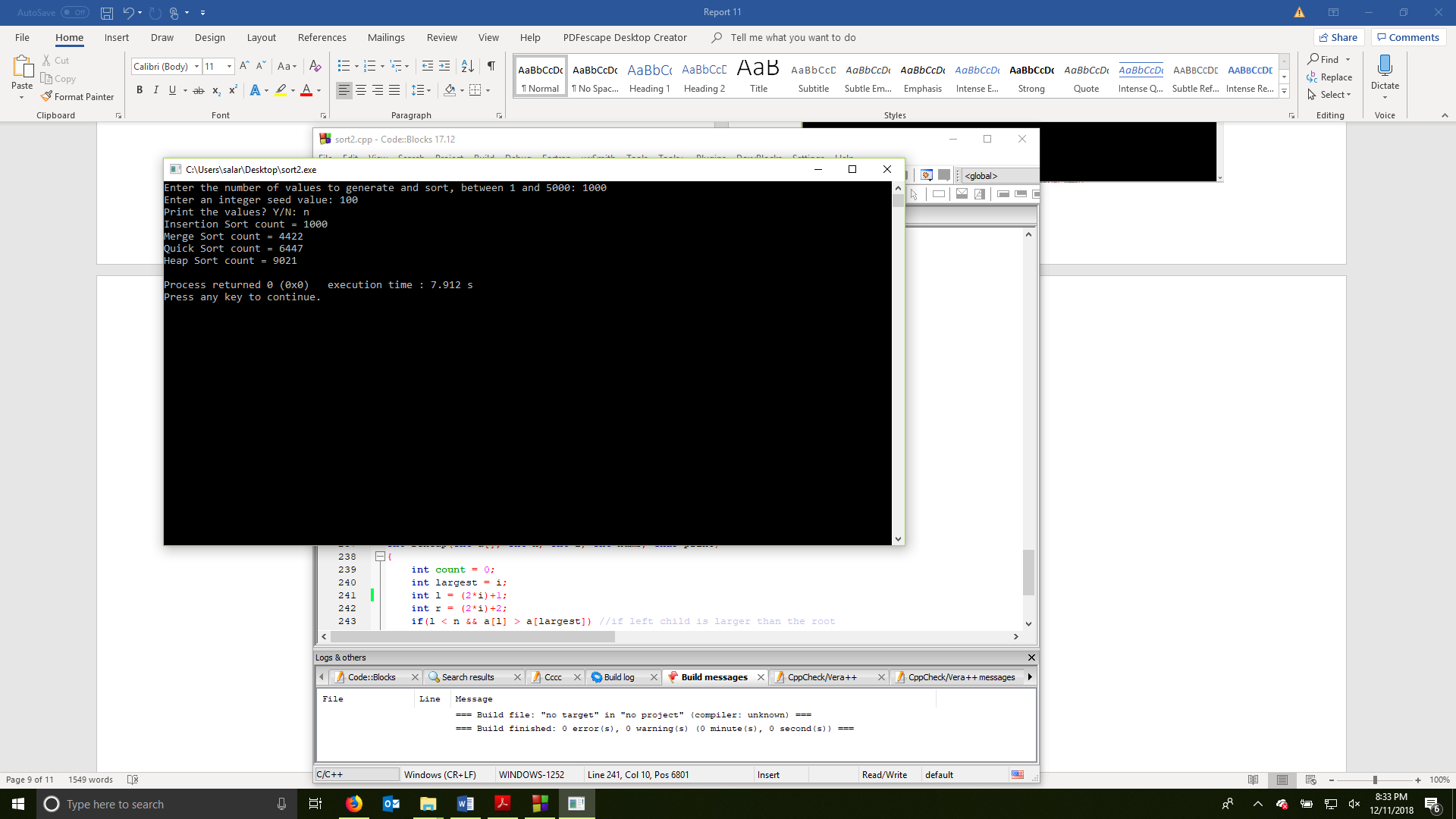
Project 11

12/10/18









Samuel Larson

Project 11

12/10/18

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 4 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: 3

Print the values? Y/N: y

Original array: 48,96,94,91

Insertion Sort:

48,96,94,91

48,94,96,91

48,91,94,96

48,91,94,96

Merge Sort:

48,96,94,91

48,96,91,94

48,91,94,96

Quick Sort:

48,91,94,96

48,94,91,96

48,94,91,96

48,91,94,96

48,91,94,96

Heap Sort:

96,48,94,91

91,48,94,96

94,48,91,96

91,48,94,96

48,91,94,96

48,91,94,96

Insertion Sort count = 4

Merge Sort count = 2

Quick Sort count = 5

Heap Sort count = 6

Process returned 0 (0x0) execution time : 9.059 s

Press any key to continue.

Samuel Larson

Project 11

12/10/18

Summary

The objective of this project was to build off program 9 to add a function to keep track of how long it takes for a heap sort to sort an array along with an insertion sort, merge sort, and quick sort. The program allows the user to enter a number of arrays and a random seed to generate the array identical arrays from. The program also had to allow the user to choose whether the program prints the arrays each time there is a swap or not.

This program proves the big O of these four functions by keeping track of how many times they swap numbers. The insertion sort has a big O(N) because how many swaps it makes is directly related to how many numbers are in the array, merge sort has a big O(nlogn) and worst case of O(n^2) because the values are divided into halves and sorted in pairs. The quick sort has a big O(nlogn) and a worst case of O(n^2) because it uses a random number within the array as a pivot and sorts each side of the pivot into larger or smaller. The heap sort has a big O(nlogn) with a worst case scenario of O(n^2) because the algorithm sorts by swapping a value with it’s larger child if it is larger than it’s parent until the largest values are near the root of the tree.