Goran Somic

CSC 407 Systems II Fall 2019

Homework #1

(**3**)

**TIMING PART 1**

A.) Insertion Sort --self seconds--- 12.16

B.) Bubble Sort -- self seconds--- 46.20

(**4**)

**TIMING PART 2**

A.) Insertion Sort --self seconds--- 02.63

B.) Bubble Sort -- self seconds--- 15.44

(**5**)

**ALGORITHM CHOICE VS COMPILER OPTIMIZATION**

According to my results, I found that a good algorithm does better with O0 than a bad algorithm with O2 optimization. For the case of Insertion sort, the added optimization did help the program significantly, dropping it from 12.16 seconds to 2.63 seconds. In this case, O2 optimization helped with insertion sort by decreasing the amount of time it took for the program to run to roughly 1/6th of its original run time with O0 optimization.

However, for bubble sort, the optimization did not help the program significantly. The bad algorithm still took priority here and the program was still extremely slow. Although the added optimization did help the program, it could not alone make up for the bad algorithm.

(**6**)

**PARTS OF AN EXECUTABLE**

A) The string "%d\n" in main()

**Command:** mov $0x10000, %edi

**Result**: moves 65536(hex ox10000) into edi register;the number of integers we are sorting in

each algorithm; the %d in the main program

B) The local variable haveExchanged in bubbleSort()

**Command**: movl $0x0, 0x4(%rbp)

**Result**: assigns 0 to haveExchanged

C) The global variable array[] in sortProg.c

**Command**: mov 0x10000, %esi

**Result**: passes the global variable array[] size

D) The code for insertionSort()

**Command**: callq <exchange>

Addl $0x1, -0x4(%rbp)

Mov –0x4(%rbp),%eax

Cmp –0x14(%rbp), %eax

Jl 400802 <insertionSort+0x28>

Addl $0x1, -0x8(%rbp)

Mov –0x14(%rbp), %eax

sub $0x1, %eax

Cmp –0x8(%rbp), %eax

Jg 4007f7 <insertionSort+0x1d>

**Result**: sorts array via insertion sort

(**7**)

**COMPILER OPTIMIZATION**

***Ex 1****) file: sortO2*

Xor %r12d, %r12d

***Ex 2****) file: sort02*

Xor % ecx, %ecx