**CSC 407: Computer Systems II**

Professor Name: Joseph Phillips
Student Name: Abdulaziz Alqulaysh

# Assignment #1

## 1. Timing: Part 1

*Run the program twice* timing it both times, and answer the following:

quickSort() : 00.80 Cumulative seconds.

bubbleSort() : 33.64 Cumulative seconds.

## 2. Timing: Part 2

*Run the program twice* timing it both times, and answer the following:

quickSort() : 00.50 Cumulative seconds.

bubbleSort() : 21.19 Cumulative seconds.

## 3. Parts of an executable:

Please find the following inside of `sort00`, either by using `objdump` (if it exists in the executable) or by disassembling the code and showing where the code manipulates the heap or stack. Show a *disassembly* or *objdump*.

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

```
[aalqulay@cdmlinux Ass1]$ objdump -d -j .rodata sort00

sort00:     file format elf32-i386

Disassembly of section .rodata:

08048cf4 <_fp_hw>:
 8048cf4:       03 00 00 00                                         ....

08048cf8 <_IO_stdin_used>:
 8048cf8:       01 00 02 00                                         ....

08048cfc <__dso_handle>:
 8048cfc:       00 00 00 00 42 55 47 21 20 4e 6f 6e 2d 73 65 6e     ....BUG! Non-sen
 8048d0c:       73 69 63 61 6c 20 71 75 69 63 6b 53 6f 72 74 5f     sical quickSort_
 8048d1c:       72 65 63 75 72 73 69 76 65 28 29 20 63 6f 6e 64     recursive() cond
 8048d2c:       69 74 69 6f 6e 20 30 0a 00 00 00 00 42 55 47 21     ition 0.....BUG!
 8048d3c:       20 4e 6f 6e 2d 73 65 6e 73 69 63 61 6c 20 71 75      Non-sensical qu
 8048d4c:       69 63 6b 53 6f 72 74 5f 72 65 63 75 72 73 69 76     ickSort_recursiv
 8048d5c:       65 28 29 20 63 6f 6e 64 69 74 69 6f 6e 20 31 0a     e() condition 1.
 8048d6c:       00 00 00 00 48 6f 77 20 77 6f 75 6c 64 20 79 6f     ....How would yo
 8048d7c:       75 20 6c 69 6b 65 20 74 6f 20 73 6f 72 74 20 25     u like to sort %
 8048d8c:       64 20 69 6e 74 65 67 65 72 73 3a 0a 31 3a 20 42     d integers:.1: B
 8048d9c:       75 62 62 6c 65 2d 73 6f 72 74 0a 32 3a 20 51 75     ubble-sort.2: Qu
 8048dac:       69 63 6b 2d 73 6f 72 74 0a 59 6f 75 72 20 63 68     ick-sort.Your ch
 8048dbc:       6f 69 63 65 3f 20 00 25 64 0a 00                    oice? .%d..
```

B. The local variable `temp` in `exchange()`

The local variable `temp` is in the stack not in the executable file because the program does not run yet.

```
[aalqulay@cdmlinux Ass1]$ objdump -s -j .bss sort00

sort00:     file format elf32-i386

[aalqulay@cdmlinux Ass1]$ objdump -s -j .data sort00

sort00:     file format elf32-i386

Contents of section .data:
 8049fac 00000000                             ....
[aalqulay@cdmlinux Ass1]$
```

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

```
[aalqulay@cdmlinux Ass1]$ objdump -d -j .bss sort00

sort00:     file format elf32-i386

Disassembly of section .bss:

08049fc0 <stderr@@GLIBC_2.0>:
 8049fc0:       00 00 00 00                                         ....

08049fc4 <stdin@@GLIBC_2.0>:
 8049fc4:       00 00 00 00                                         ....

08049fc8 <called.3259>:
 8049fc8:       00 00 00 00                                         ....

08049fcc <dtor_idx.5793>:
 8049fcc:       00 00 00 00                                         ....

08049fd0 <completed.5791>:

08049fe0 <array>:
[aalqulay@cdmlinux Ass1]$
```

D. The code for `quickSort()`

```
08048a4a <quickSort>:
 8048a4a:       55                      push   %ebp
 8048a4b:       89 e5                   mov    %esp,%ebp
 8048a4d:       83 ec 18                sub    $0x18,%esp
 8048a50:       8b 45 08                mov    0x8(%ebp),%eax
 8048a53:       83 e8 01                sub    $0x1,%eax
 8048a56:       89 44 24 08             mov    %eax,0x8(%esp)
 8048a5a:       c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
 8048a61:       00
 8048a62:       8b 45 0c                mov    0xc(%ebp),%eax
 8048a65:       89 04 24                mov    %eax,(%esp)
 8048a68:       e8 63 fd ff ff          call   80487d0 <quickSort_recursive>
 8048a6d:       c9                      leave
 8048a6e:       c3                      ret
 8048a6f:       90                      nop
```

## 4. Compiler optimizations:

Look for and show examples of the following optimizations in either `sortO0` or `sortO2`. For each:

- Tell if it exists in either `sortO0`, `sortO2` or *both*, and,
- Show a *disassembly* of the function that has it.

A. usage of registers to hold vars (as opposed to the stack)

They both uses registers. However, `sortO2` uses more registers to keep variables and here is an example: exchange(). In `sortO2`, it virtually only uses registers rather than the stack. On the other hand, `sortO0` most variables are kept on the stack.

sortO0

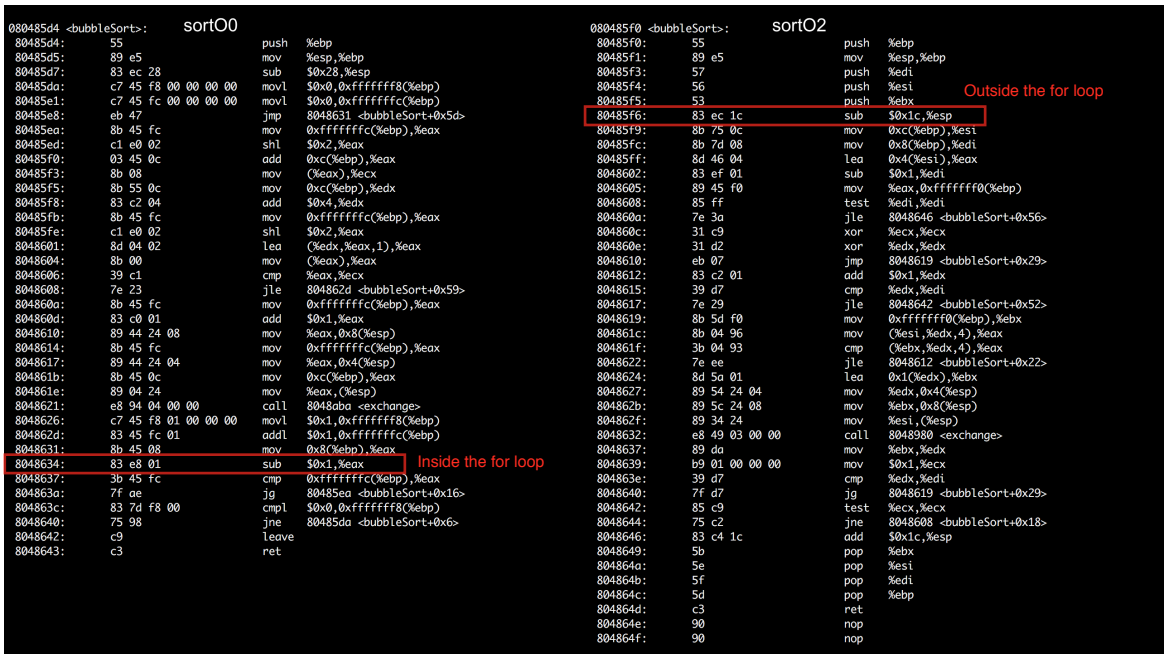
```
08048aba <exchange>:
 8048aba:       55                      push   %ebp
 8048abb:       89 e5                   mov    %esp,%ebp
 8048abd:       83 ec 10                sub    $0x10,%esp
 8048ac0:       8b 45 0c                mov    0xc(%ebp),%eax
 8048ac3:       c1 e0 02                shl    $0x2,%eax
 8048ac6:       03 45 08                add    0x8(%ebp),%eax
 8048ac9:       8b 00                   mov    (%eax),%eax
 8048acb:       89 45 fc                mov    %eax,0xfffffffc(%ebp)
 8048ace:       8b 45 0c                mov    0xc(%ebp),%eax
 8048ad1:       c1 e0 02                shl    $0x2,%eax
 8048ad4:       89 c2                   mov    %eax,%edx
 8048ad6:       03 55 08                add    0x8(%ebp),%edx
 8048ad9:       8b 45 10                mov    0x10(%ebp),%eax
 8048adc:       c1 e0 02                shl    $0x2,%eax
 8048adf:       03 45 08                add    0x8(%ebp),%eax
 8048ae2:       8b 00                   mov    (%eax),%eax
 8048ae4:       89 02                   mov    %eax,(%edx)
 8048ae6:       8b 45 10                mov    0x10(%ebp),%eax
 8048ae9:       c1 e0 02                shl    $0x2,%eax
 8048aec:       89 c2                   mov    %eax,%edx
 8048aee:       03 55 08                add    0x8(%ebp),%edx
 8048af1:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 8048af4:       89 02                   mov    %eax,(%edx)
 8048af6:       c9                      leave
 8048af7:       c3                      ret
```

sortO2

```
08048980 <exchange>:
 8048980:       55                      push   %ebp
 8048981:       89 e5                   mov    %esp,%ebp
 8048983:       8b 55 08                mov    0x8(%ebp),%edx
 8048986:       8b 4d 0c                mov    0xc(%ebp),%ecx
 8048989:       8b 45 10                mov    0x10(%ebp),%eax
 804898c:       53                      push   %ebx
 804898d:       8d 0c 8a                lea    (%edx,%ecx,4),%ecx
 8048990:       8d 04 82                lea    (%edx,%eax,4),%eax
 8048993:       8b 19                   mov    (%ecx),%ebx
 8048995:       8b 10                   mov    (%eax),%edx
 8048997:       89 11                   mov    %edx,(%ecx)
 8048999:       89 18                   mov    %ebx,(%eax)
 804899b:       5b                      pop    %ebx
 804899c:       5d                      pop    %ebp
 804899d:       c3                      ret
 804899e:       66 90                   xchg   %ax,%ax
```

B. code motion

Code motion exists in `sortO2` when the compiler takes the code that doesn't need to be in the loop and moving it outside of it to make the program more efficient. Here is an example: bubbleSort(). `for (index = 0; index < arrayLen-1; index++)` in `sortO2`, it takes (arrayLen-1) to the outside of the for loop.



C. reduction in strength

Reduction in strength existed in both. I noticed that in both they changed from expensive operations to cheaper ones for example: quickSort_choosePivot() it has a division instruction but in both executable files have used Bitwise operations (right shift) instead of Int div.

```
int    midIndex    = (loIndex + hiIndex) / 2;
```