

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_hwc>:
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_recursev
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..
```

CSC 407: Computer Systems II

- 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   %ebp
8048a6e:  c3             ret
8048a6f:  90             nop
```

4. Compiler optimizations:

Look for and show examples of the following optimizations in either `sort00` or `sort02`. For each:

- Tell if it exists in either `sort00`, `sort02` 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 use registers. However, `sort02` uses more registers to keep variables and here is an example: `exchange()`. In `sort02`, it virtually only uses registers rather than the stack. On the other hand, `sort00` most variables are kept on the stack.

08048aba <exchange>: sort00		08048980 <exchange>: sort02	
8048aba: 55	push %ebp	8048980: 55	push %ebp
8048abb: 89 e5	mov %esp,%ebp	8048981: 89 e5	mov %esp,%ebp
8048abd: 83 ec 10	sub \$0x10,%esp	8048983: 8b 55 08	mov 0x8(%ebp),%edx
8048ac0: 8b 45 0c	mov 0xc(%ebp),%eax	8048986: 8b 4d 0c	mov 0xc(%ebp),%ecx
8048ac3: c1 e0 02	shl \$0x2,%eax	8048989: 8b 45 10	mov 0x10(%ebp),%eax
8048ac6: 03 45 08	add 0x8(%ebp),%eax	804898c: 53	push %ebx
8048ac9: 8b 00	mov (%eax),%eax	804898d: 8d 0c 8a	lea (%edx,%ecx,4),%ecx
8048acb: 89 45 fc	mov %eax,0xffffffff(%ebp)	8048990: 8d 04 82	lea (%edx,%eax,4),%eax
8048ace: 8b 45 0c	mov 0xc(%ebp),%eax	8048993: 8b 19	mov (%ecx),%ebx
8048ad1: c1 e0 02	shl \$0x2,%eax	8048995: 8b 10	mov (%eax),%edx
8048ad4: 89 c2	mov %eax,%edx	8048997: 89 11	mov %edx,%ecx
8048ad6: 03 55 08	add 0x8(%ebp),%edx	8048999: 89 18	mov %ebx,%eax
8048ad9: 8b 45 10	mov 0x10(%ebp),%eax	804899b: 5b	pop %ebx
8048adc: c1 e0 02	shl \$0x2,%eax	804899c: 5d	pop %ebp
8048adf: 03 45 08	add 0x8(%ebp),%eax	804899d: c3	ret
8048ae2: 8b 00	mov (%eax),%eax	804899e: 66 90	xchg %ax,%ax
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 0xffffffff(%ebp),%eax		
8048af4: 89 02	mov %eax,(%edx)		
8048af6: c9	leave		
8048af7: c3	ret		

CSC 407: Computer Systems II

B. code motion

Code motion exists in `sort02` 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 `sort02`, it takes `(arrayLen-1)` to the outside of the for loop.

00485d4: <bubbleSort>: sort00	00485f0: <bubbleSort>: sort02
00485d4: 55 push %ebp	00485f0: 55 push %ebp
00485d5: 89 e5 mov %esp,%ebp	00485f1: 89 e5 mov %esp,%ebp
00485d7: 83 ec 28 sub \$0x28,%esp	00485f3: 57 push %edi
00485da: c7 45 f8 00 00 00 00 movl \$0x0,0xffffffff(%ebp)	00485f4: 56 push %esi
00485e1: c7 45 fc 00 00 00 00 movl \$0x0,0xffffffff(%ebp)	00485f5: 53 push %ebx
00485e8: eb 47 jmp 0048631 <bubbleSort+0x5d>	00485f6: 83 ec 1c sub \$0xc,%esp
00485ea: 8b 45 fc mov 0xffffffff(%ebp),%eax	00485f9: 8b 75 0c mov 0xc(%ebp),%edi
00485ed: c1 e0 02 shl \$0x2,%eax	00485fc: 8b 7d 08 mov 0x8(%ebp),%edi
00485f0: 03 45 0c add 0xc(%ebp),%eax	00485ff: 8d 46 04 lea 0x4(%esi),%eax
00485f3: 8b 08 mov (%eax),%ecx	0048602: 83 ef 01 sub \$0x1,%edi
00485f5: 8b 55 0c mov 0xc(%ebp),%edx	0048605: 89 45 f0 mov %eax,0xffffffff(%ebp)
00485f8: 83 c2 04 add \$0x4,%edx	0048608: 85 ff test %edi,%edi
00485fb: 8b 45 fc mov 0xffffffff(%ebp),%eax	004860b: 7e 3a jle 0048646 <bubbleSort+0x56>
00485fe: c1 e0 02 shl \$0x2,%eax	004860c: 31 c9 xor %ecx,%ecx
0048601: 8d 04 02 lea (%edx,%eax,1),%eax	004860e: 31 d2 xor %edx,%edx
0048604: 8b 00 mov (%eax),%eax	0048610: eb 07 jmp 0048619 <bubbleSort+0x29>
0048606: 39 c1 cmp %eax,%ecx	0048612: 83 c2 01 add \$0x1,%edx
0048608: 7e 23 jle 004862d <bubbleSort+0x59>	0048615: 39 d7 cmp %edx,%edi
004860a: 8b 45 fc mov 0xffffffff(%ebp),%eax	0048617: 7e 29 jle 0048642 <bubbleSort+0x52>
004860d: 83 c0 01 add \$0x1,%eax	0048619: 8b 5d f0 mov 0xffffffff(%ebp),%ebx
0048610: 89 44 24 08 mov %eax,0x8(%esp)	004861c: 8b 04 96 mov (%esi,%edx,4),%eax
0048614: 8b 45 fc mov 0xffffffff(%ebp),%eax	004861f: 3b 04 93 cmp (%ebx,%edx,4),%eax
0048617: 89 44 24 04 mov %eax,0x4(%esp)	0048622: 7e ee jle 0048612 <bubbleSort+0x22>
004861b: 8b 45 0c mov 0xc(%ebp),%eax	0048624: 8d 5a 01 lea 0x1(%edx),%ebx
004861e: 89 04 24 mov %eax,(%esp)	0048627: 89 54 24 04 mov %edx,0x4(%esp)
0048621: e8 94 04 00 00 call 00486a <exchange>	004862b: 89 5c 24 08 mov %ebx,0x8(%esp)
0048626: c7 45 f8 01 00 00 00 movl \$0x1,0xffffffff(%ebp)	004862f: 89 34 24 mov %esi,(%esp)
004862d: 83 45 fc 01 addl \$0x1,0xffffffff(%ebp)	0048632: e8 49 03 00 00 call 004890 <exchange>
0048631: 8b 45 08 mov 0x8(%ebp),%eax	0048637: 89 da mov %ebx,%edx
0048634: 83 e8 01 sub \$0x1,%eax	0048639: b9 01 00 00 00 mov \$0x1,%ecx
0048637: 3b 45 fc cmp 0xffffffff(%ebp),%eax	004863e: 39 d7 cmp %edx,%edi
004863a: 7f ae jg 00485ea <bubbleSort+0x16>	0048640: 7f d7 jg 0048619 <bubbleSort+0x29>
004863c: 83 7d f8 00 cmpl \$0x0,0xffffffff(%ebp)	0048642: 85 c9 test %ecx,%ecx
0048640: 75 98 jne 00485da <bubbleSort+0x6>	0048644: 75 c2 jne 0048608 <bubbleSort+0x18>
0048642: c9 leave	0048646: 83 c4 1c add \$0xc,%esp
0048643: c3 ret	0048649: 5b pop %ebx
	004864a: 5e pop %esi
	004864b: 5f pop %edi
	004864c: 5d pop %ebp
	004864d: c3 ret
	004864e: 90 nop
	004864f: 90 nop

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;
```

0048644 <quickSort_choosePivot>:			sort00	0048650 <quickSort_choosePivot>:			sort02
0048644: 55	push	%ebp		0048650: 55	push	%ebp	
0048645: 83 e5	mov	%esp,%ebp		0048651: 83 e5	mov	%esp,%ebp	
0048647: 83 ec 1c	sub	\$0x1c,%esp		0048653: 83 ec 0c	sub	\$0xc,%esp	
004864a: 8b 45 10	mov	0x10(%ebp),%eax		0048656: 8b 1c 24	mov	%ebx,0(%esp)	
004864d: 8b 55 0c	mov	0xc(%ebp),%edx		0048659: 8b 5d 0c	mov	0xc(%ebp),%ebx	
0048650: 01 c2	add	%eax,%edx		004865c: 89 7c 24 08	mov	%edi,0x8(%esp)	
0048652: 89 d0	mov	%edx,%eax		0048660: 8b 7d 10	mov	0x10(%ebp),%edi	
0048654: c1 e8 1f	shr	\$0x1f,%eax	Bitwise Operations	0048663: 89 74 24 04	mov	%esi,0x4(%esp)	
0048657: 01 d0	add	%edx,%eax		0048667: 8b 75 08	mov	0x8(%ebp),%esi	
0048659: d1 f8	sar	%eax		004866a: 8d 14 1f	lea	(%edi,%ebx,1),%edx	
004865b: 89 45 1c	mov	%eax,0x1fffffc(%ebp)		004866d: 89 d0	mov	%edx,%eax	
004865e: 8b 45 0c	mov	0xc(%ebp),%eax		004866f: 8b 0c 9e	mov	(%esi,%ebx,4),%ecx	
0048661: c1 e0 02	shr	\$0x2,%eax		0048672: c1 e8 1f	shr	\$0x1f,%eax	Bitwise Operations
0048664: 03 45 08	add	0x8(%ebp),%eax		0048675: 01 d0	add	%edx,%eax	
0048667: 8b 10	mov	(%eax),%edx		0048677: 8b 14 be	mov	(%esi,%edi,4),%edx	
0048669: 8b 45 10	mov	0x10(%ebp),%eax		004867a: d1 f8	sar	%eax	
004866c: c1 e0 02	shr	\$0x2,%eax		004867c: 39 d1	cmp	%edx,%ecx	
004866f: 03 45 08	add	0x8(%ebp),%eax		004867e: 7d 20	jge	00486a0 <quickSort_choosePivot+0x50>	
0048672: 8b 00	mov	(%eax),%eax		0048680: 8b 34 86	mov	(%esi,%eax,4),%esi	
0048674: 39 c2	cmp	%eax,%edx		0048683: 39 f1	cmp	%esi,%ecx	
0048676: 7d 55	jge	00486cd <quickSort_choosePivot+0x89>		0048685: 7f 06	jg	004868d <quickSort_choosePivot+0x3d>	
0048678: 8b 45 fc	mov	0xfffffff(%ebp),%eax		0048687: 39 f2	cmp	%esi,%edx	
004867b: c1 e0 02	shr	\$0x2,%eax		0048689: 7d 20	jge	00486ab <quickSort_choosePivot+0x5b>	
004867e: 03 45 08	add	0x8(%ebp),%eax		004868b: 8b fb	mov	%edi,%ebx	
0048681: 8b 10	mov	(%eax),%edx		004868d: 89 d8	mov	%ebx,%eax	
0048683: 8b 45 0c	mov	0xc(%ebp),%eax		004868f: 8b 74 24 04	mov	0x4(%esp),%esi	
0048686: c1 e0 02	shr	\$0x2,%eax		0048693: 8b 1c 24	mov	0(%esp),%ebx	
0048689: 03 45 08	add	0x8(%ebp),%eax		0048696: 8b 7c 24 08	mov	0x8(%esp),%edi	
004868c: 8b 00	mov	(%eax),%eax		004869a: 89 ec	mov	%ebp,%esp	
004868e: 39 c2	cmp	%eax,%edx		004869c: 5d	pop	%ebp	
0048690: 7d 0b	jge	004869d <quickSort_choosePivot+0x59>		004869d: c3	ret		
0048692: 8b 45 0c	mov	0xc(%ebp),%eax		004869e: 66 90	xchg	%ax,%ax	
0048695: 89 45 e4	mov	%eax,0xffffffe4(%ebp)		00486a0: 8b 34 86	mov	(%esi,%eax,4),%esi	
0048698: e9 80 00 00 00	jmp	004871d <quickSort_choosePivot+0xd9>		00486a3: 39 f2	cmp	%esi,%edx	
004869d: 8b 45 fc	mov	0xfffffff(%ebp),%eax		00486a5: 7f e4	jg	00486ab <quickSort_choosePivot+0x3b>	
00486a0: c1 e0 02	shr	\$0x2,%eax		00486a7: 39 f1	cmp	%esi,%ecx	
00486a3: 03 45 08	add	0x8(%ebp),%eax		00486a9: 7c e2	jl	004868d <quickSort_choosePivot+0x3d>	
00486a6: 8b 10	mov	(%eax),%edx		00486ab: 89 c3	mov	%eax,%ebx	
00486a8: 8b 45 10	mov	0x10(%ebp),%eax		00486ad: 8d 76 00	lea	0x0(%esi),%esi	
00486ab: c1 e0 02	shr	\$0x2,%eax		00486b0: eb db	jmp	004868d <quickSort_choosePivot+0x3d>	
00486ae: 03 45 08	add	0x8(%ebp),%eax		00486b2: 8d b4 26 00 00 00 00	lea	0x0(%esi),%esi	
00486b1: 8b 00	mov	(%eax),%eax		00486b9: 8d bc 27 00 00 00 00	lea	0x0(%edi),%edi	
00486b3: 39 c2	cmp	%eax,%edx					
00486b5: 7e 08	jle	00486bf <quickSort_choosePivot+0x7b>					
00486b7: 8b 45 10	mov	0x10(%ebp),%eax					
00486ba: 45 e8	mov	%eax,0xfffff8(%ebp)					
00486bd: eb 06	jmp	00486c5 <quickSort_choosePivot+0x81>					
00486bf: 8b 45 fc	mov	0xfffffff(%ebp),%eax					