**1.**

Function Self seconds

initializeBigArray() \_\_0.00\_\_\_\_\_\_\_\_

countAdjaceent() \_\_\_11.2\_\_\_\_\_\_\_

funkyFunction() \_\_\_0.00\_\_\_\_\_\_\_

**2.** (8 Points) How did it do the operation (i % 16) == 0x4? Was it done as a modulus (the same as an expensive division, but returns the remainder instead of the quotient) or something else? Show the assembly language for this C code using gdb to dissassemble funkyFunction() of q1None.

Dump of assembler code for function funkyFunction:

0x0804862c <+0>: push %ebp

0x0804862d <+1>: mov %esp,%ebp

0x0804862f <+3>: sub $0x1c,%esp

0x08048632 <+6>: call 0x8048450 <mcount@plt>

0x08048637 <+11>: movl $0x0,-0x4(%ebp)

0x0804863e <+18>: movl $0x0,-0x8(%ebp)

0x08048645 <+25>: jmp 0x80486a4 <funkyFunction+120>

**0x08048647 <+27>: mov -0x8(%ebp),%eax**

**0x0804864a <+30>: and $0xf,%eax**

**0x0804864d <+33>: cmp $0x4,%eax**

0x08048650 <+36>: jne 0x8048677 <funkyFunction+75>

0x08048652 <+38>: mov 0x8(%ebp),%eax

0x08048655 <+41>: sub $0x2,%eax

0x08048658 <+44>: movl $0x1,0x8(%esp)

0x08048660 <+52>: mov 0xc(%ebp),%edx

0x08048663 <+55>: mov %edx,0x4(%esp)

0x08048667 <+59>: mov %eax,(%esp)

0x0804866a <+62>: call 0x80485dc <countAdjacent>

0x0804866f <+67>: shl $0x2,%eax

0x08048672 <+70>: add %eax,-0x4(%ebp)

0x08048675 <+73>: jmp 0x80486a0 <funkyFunction+116>

0x08048677 <+75>: mov 0x8(%ebp),%eax

0x0804867a <+78>: sub $0x2,%eax

0x0804867d <+81>: movl $0xffffffff,0x8(%esp)

0x08048685 <+89>: mov 0xc(%ebp),%edx

0x08048688 <+92>: mov %edx,0x4(%esp)

0x0804868c <+96>: mov %eax,(%esp)

0x0804868f <+99>: call 0x80485dc <countAdjacent>

0x08048694 <+104>: mov %eax,%edx

0x08048696 <+106>: mov %edx,%eax

0x08048698 <+108>: shl $0x2,%eax

0x0804869b <+111>: add %edx,%eax

0x0804869d <+113>: add %eax,-0x4(%ebp)

0x080486a0 <+116>: addl $0x1,-0x8(%ebp)

0x080486a4 <+120>: mov 0x8(%ebp),%eax

0x080486a7 <+123>: sub $0x1,%eax

0x080486aa <+126>: cmp -0x8(%ebp),%eax

0x080486ad <+129>: ja 0x8048647 <funkyFunction+27>

0x080486af <+131>: mov -0x4(%ebp),%eax

0x080486b2 <+134>: leave

0x080486b3 <+135>: ret

and then look for the code that sets up the calls to countAdjacent(). The (i % 16) == 0x4 test is done before either countAdjacent() call.

**3.** (8 Points) Compile it for profiling but with optimization with:

What are the number of self seconds taken by:

Function Self seconds

initializeBigArray() \_\_\_0.00\_\_\_\_\_\_\_

countAdjacent() \_3.59\_\_\_\_\_\_\_\_\_

funkyFunction() \_\_0.00\_\_\_\_\_\_\_\_

**4.** (8 Points) Use gdb to dissassemble countAdjacent() of both q1None and q1Compiler.

Don't try to understand all the code but in general how did the optimizer make q1Compiler's countAdjacent() faster than q1None's? Give a specific example by comparing both assembly codes.

unoptimized

Dump of assembler code for function countAdjacent:

0x080485dc <+0>: push %ebp

0x080485dd <+1>: mov %esp,%ebp

0x080485df <+3>: sub $0x10,%esp

0x080485e2 <+6>: call 0x8048450 <mcount@plt>

0x080485e7 <+11>: movl $0x0,-0x4(%ebp)

0x080485ee <+18>: movl $0x0,-0x8(%ebp)

0x080485f5 <+25>: jmp 0x804861f <countAdjacent+67>

0x080485f7 <+27>: mov -0x8(%ebp),%eax

0x080485fa <+30>: shl $0x2,%eax

0x080485fd <+33>: add 0xc(%ebp),%eax

0x08048600 <+36>: mov (%eax),%edx

0x08048602 <+38>: mov -0x8(%ebp),%eax

0x08048605 <+41>: add $0x1,%eax

0x08048608 <+44>: shl $0x2,%eax

0x0804860b <+47>: add 0xc(%ebp),%eax

0x0804860e <+50>: mov (%eax),%eax

0x08048610 <+52>: add 0x10(%ebp),%eax

0x08048613 <+55>: cmp %eax,%edx

0x08048615 <+57>: jne 0x804861b <countAdjacent+63>

0x08048617 <+59>: addl $0x1,-0x4(%ebp)

0x0804861b <+63>: addl $0x1,-0x8(%ebp)

0x0804861f <+67>: mov 0x8(%ebp),%eax

0x08048622 <+70>: cmp -0x8(%ebp),%eax

0x08048625 <+73>: ja 0x80485f7 <countAdjacent+27>

0x08048627 <+75>: mov -0x4(%ebp),%eax

0x0804862a <+78>: leave

0x0804862b <+79>: ret

End of assembler dump.

optimized

Dump of assembler code for function countAdjacent:

0x080485f7 <+0>: push %ebp

0x080485f8 <+1>: mov %esp,%ebp

0x080485fa <+3>: push %edi

0x080485fb <+4>: push %esi

0x080485fc <+5>: push %ebx

0x080485fd <+6>: sub $0x4,%esp

0x08048600 <+9>: call 0x8048460 <mcount@plt>

0x08048605 <+14>: mov 0xc(%ebp),%ecx

0x08048608 <+17>: mov 0x10(%ebp),%edi

0x0804860b <+20>: mov 0x8(%ebp),%eax

0x0804860e <+23>: mov %eax,-0x10(%ebp)

0x08048611 <+26>: mov $0x0,%eax

0x08048616 <+31>: mov $0x0,%edx

0x0804861b <+36>: jmp 0x8048632 <countAdjacent+59>

0x0804861d <+38>: mov (%ecx,%edx,4),%esi

0x08048620 <+41>: add $0x1,%edx

0x08048623 <+44>: mov %edi,%ebx

0x08048625 <+46>: add (%ecx,%edx,4),%ebx

0x08048628 <+49>: cmp %ebx,%esi

0x0804862a <+51>: sete %bl

0x0804862d <+54>: movzbl %bl,%ebx

0x08048630 <+57>: add %ebx,%eax

0x08048632 <+59>: cmp -0x10(%ebp),%edx

0x08048635 <+62>: jne 0x804861d <countAdjacent+38>

0x08048637 <+64>: add $0x4,%esp

0x0804863a <+67>: pop %ebx

0x0804863b <+68>: pop %esi

0x0804863c <+69>: pop %edi

0x0804863d <+70>: pop %ebp

0x0804863e <+71>: ret

**The main difference is the optimized version uses registers and reduces the number of math functions that are expensive to call. Instead it optimizes the code by moving bits around into registers which are much faster. This can be seen easily by the assembly removing the add function with move functions.**

**5.** (8 Points) One optimization the compiler may not have made would be do the two countAdjacent() calls once each before the loop in funkyFunction(), put them in variables, and then use those variables in the loop. Re-write the code this way:

What are the number of self seconds taken by:

Function Self seconds

initializeBigArray() \_0.00\_\_\_\_\_\_\_\_\_

countAdjacent() \_\_\_0.00\_\_\_\_\_\_\_

funkyFunction() \_\_\_0.00\_\_\_\_\_\_\_

**6.** (5 Points) Which optimizations ought to be done by the compiler? **Mathematical functions that have a clear optimization that will allow a programmer to express themselves simply.**

Which optimizations ought to be done by the programmer**? the programmer should be able to decide simple optimizations such as the timing of calls and calling heavy computational functions as seldom as possible.**

Linker operation (55 Points total)

**1.** (5 Points) Use objdump (not gdb!) to disassemble both main() and createArray() in whole. Find the call to createArray() in main(). Show the math of how the number in that call instruction is used to compute the address where createArray() actually is.

**2.** (5 Points) Which segment (that is: .text, .rodata, .data or .bss) of p1.o has string constant used in enterValue()'s printf() call?

Show this with objdump.

**3.** (5 Points) Can you find the memory for enterValue()'s variable min using objdump?

If you can, use objdump to show where it is.

If you can't, tell why not.

00000000 <enterValue>:

0: 55 push %ebp

1: 89 e5 mov %esp,%ebp

3: 83 ec 18 sub $0x18,%esp

6: b8 00 00 00 00 mov $0x0,%eax

**b: 8b 55 10 mov 0x10(%ebp),%edx**

**e: 89 54 24 0c mov %edx,0xc(%esp)**

12: 8b 55 0c mov 0xc(%ebp),%edx

15: 89 54 24 08 mov %edx,0x8(%esp)

19: 8b 55 08 mov 0x8(%ebp),%edx

1c: 89 54 24 04 mov %edx,0x4(%esp)

20: 89 04 24 mov %eax,(%esp)

23: e8 fc ff ff ff call 24 <enterValue+0x24>

**4.** (5 Points) Which segment of p2.o has the function freeArray()?

Show this with objdump.

**.TEXT**

**Disassembly of section .text:**

**…**

**000000f9 <freeArray>:**

**f9: 55 push %ebp**

**fa: 89 e5 mov %esp,%ebp**

**fc: 83 ec 18 sub $0x18,%esp**

**ff: a1 00 00 00 00 mov 0x0,%eax**

**104: 85 c0 test %eax,%eax**

**106: 74 17 je 11f <freeArray+0x26>**

**108: a1 00 00 00 00 mov 0x0,%eax**

**10d: 89 04 24 mov %eax,(%esp)**

**110: e8 fc ff ff ff call 111 <freeArray+0x18>**

**115: c7 05 00 00 00 00 00 movl $0x0,0x0**

**11c: 00 00 00**

**11f: c9 leave**

**120: c3 ret**

**5.** (5 Points) It is rather inelegant for both p1.c and p2.c to have the code:

// PURPOSE: To hold the length of C-string 'line'.

#define MAX\_LINE 256

because if we decide to make a change we have to remember to change all .c files. Suggest a more elegant solution, one that C encourages.

**The define could be placed in a header file which can then be included in ever necessary file**