**Header files (10 Points): Attached as file as well**

#ifndef SORT\_HEADER\_H

#define SORT\_HEADER\_H

#include <stdlib.h>

#include <stdio.h>

extern void exchange(int\* array, int i, int j);

extern void insertionSort(int arrayLen, int\* array);

extern void bubbleSort(int arrayLen, int\* array);

#endif

**Timing: Part 1 (20 Points):**

**./sortO0**

1. insertionSort() self seconds: 11.88
2. bubbleSort() self seconds: 43.65

**Timing Part 2 (20 Points):**

**./sortO2**

1. insertionSort() self seconds: 2.40
2. bubbleSort() self seconds: 15.50

**Parts of an executable (Points 20):**

|  |  |  |
| --- | --- | --- |
| **Question** | **Command** | **Result** |
| (A) | gdb sortProg0  break\* main +180  run  disas  x/s 0x400590 | (gdb) x/s 0x400590  0x400590 <printf@plt>: "\377%\202\n " |
| (B) | objdump -d sortO0 | 400761: c7 45 fc 00 00 00 00 movl $0x0,-0x4(%rbp) |
| (C) | objdump -t -j .bss sortO0 | sortO0: file format elf64-x86-64  SYMBOL TABLE:  0000000000601080 l d .bss 0000000000000000 .bss  0000000000601090 l O .bss 0000000000000004 called.4239  0000000000601094 l O .bss 0000000000000001 completed.6355  00000000006010a0 g O .bss 0000000000040000 array  0000000000601080 g O .bss 0000000000000008 stdin@@GLIBC\_2.2.5  00000000006410a0 g .bss 0000000000000000 \_end  0000000000601064 g .bss 0000000000000000 \_\_bss\_start |
| (D) | objdump -d -j .text sortO0 | 00000000004007da <insertionSort>:  4007da: 55 push %rbp  4007db: 48 89 e5 mov %rsp,%rbp  4007de: 48 83 ec 20 sub $0x20,%rsp  4007e2: e8 f9 fd ff ff callq 4005e0 <mcount@plt>  4007e7: 89 7d ec mov %edi,-0x14(%rbp)  4007ea: 48 89 75 e0 mov %rsi,-0x20(%rbp)  4007ee: c7 45 f8 00 00 00 00 movl $0x0,-0x8(%rbp)  4007f5: eb 5f jmp 400856 <insertionSort+0x7c>  4007f7: 8b 45 f8 mov -0x8(%rbp),%eax  4007fa: 83 c0 01 add $0x1,%eax  4007fd: 89 45 fc mov %eax,-0x4(%rbp)  400800: eb 48 jmp 40084a <insertionSort+0x70>  400802: 8b 45 f8 mov -0x8(%rbp),%eax  400805: 48 98 cltq  400807: 48 8d 14 85 00 00 00 lea 0x0(,%rax,4),%rdx  40080e: 00  40080f: 48 8b 45 e0 mov -0x20(%rbp),%rax  400813: 48 01 d0 add %rdx,%rax  400816: 8b 10 mov (%rax),%edx  400818: 8b 45 fc mov -0x4(%rbp),%eax  40081b: 48 98 cltq  40081d: 48 8d 0c 85 00 00 00 lea 0x0(,%rax,4),%rcx  400824: 00  400825: 48 8b 45 e0 mov -0x20(%rbp),%rax  400829: 48 01 c8 add %rcx,%rax  40082c: 8b 00 mov (%rax),%eax  40082e: 39 c2 cmp %eax,%edx  400830: 7e 14 jle 400846 <insertionSort+0x6c>  400832: 8b 55 fc mov -0x4(%rbp),%edx  400835: 8b 4d f8 mov -0x8(%rbp),%ecx  400838: 48 8b 45 e0 mov -0x20(%rbp),%rax  40083c: 89 ce mov %ecx,%esi  40083e: 48 89 c7 mov %rax,%rdi  400841: e8 77 00 00 00 callq 4008bd <exchange>  400846: 83 45 fc 01 addl $0x1,-0x4(%rbp)  40084a: 8b 45 fc mov -0x4(%rbp),%eax  40084d: 3b 45 ec cmp -0x14(%rbp),%eax  400850: 7c b0 jl 400802 <insertionSort+0x28>  400852: 83 45 f8 01 addl $0x1,-0x8(%rbp)  400856: 8b 45 ec mov -0x14(%rbp),%eax  400859: 83 e8 01 sub $0x1,%eax  40085c: 3b 45 f8 cmp -0x8(%rbp),%eax  40085f: 7f 96 jg 4007f7 <insertionSort+0x1d>  400861: c9 leaveq  400862: c3 retq  0000000000400863 <initializeArray>:  400863: 55 push %rbp  400864: 48 89 e5 mov %rsp,%rbp  400867: 53 push %rbx  400868: 48 83 ec 28 sub $0x28,%rsp  40086c: e8 6f fd ff ff callq 4005e0 <mcount@plt>  400871: 89 7d dc mov %edi,-0x24(%rbp)  400874: 48 89 75 d0 mov %rsi,-0x30(%rbp)  400878: c7 45 ec 00 00 00 00 movl $0x0,-0x14(%rbp)  40087f: eb 2d jmp 4008ae <initializeArray+0x4b>  400881: 8b 45 ec mov -0x14(%rbp),%eax  400884: 48 98 cltq  400886: 48 8d 14 85 00 00 00 lea 0x0(,%rax,4),%rdx  40088d: 00  40088e: 48 8b 45 d0 mov -0x30(%rbp),%rax  400892: 48 8d 1c 02 lea (%rdx,%rax,1),%rbx  400896: e8 75 fd ff ff callq 400610 <rand@plt>  40089b: 99 cltd  40089c: c1 ea 16 shr $0x16,%edx  40089f: 01 d0 add %edx,%eax  4008a1: 25 ff 03 00 00 and $0x3ff,%eax  4008a6: 29 d0 sub %edx,%eax  4008a8: 89 03 mov %eax,(%rbx)  4008aa: 83 45 ec 01 addl $0x1,-0x14(%rbp)  4008ae: 8b 45 ec mov -0x14(%rbp),%eax  4008b1: 3b 45 dc cmp -0x24(%rbp),%eax  4008b4: 7c cb jl 400881 <initializeArray+0x1e>  4008b6: 48 83 c4 28 add $0x28,%rsp  4008ba: 5b pop %rbx  4008bb: 5d pop %rbp  4008bc: c3 retq |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Question | Explanation/Answer | sortO0 Code | sortO2 Code | For both? |
| 6A. | In bubbleSort, sortO2 uses registers to store the value of the variable haveExchanged whereas in sortO0 it modifies the stack each time. | xor %esi,%esi  …  add $0x1,%esi | movl $0x0,-0x8(%rbp)  …  mov -0x20(%rbp),%rax  …  add $0x1,%rax | Only in sortO2 do you see this |
| 6A. | In the exchange function, there appears to be a lot of modifying the stack in order to “exchange” the integers to order them, whereas it’s all done with registers in sortO2. | mov %rdi,-0x18(%rbp)  mov %esi,-0x1c(%rbp)  mov %edx,-0x20(%rbp)  mov -0x1c(%rbp),%eax | mov (%rcx),%esi  mov (%rax),%edx  mov %edx,(%rcx)  mov %esi,(%rax) | Only in sortO2 do you see this |
| 6B. | In bubbleSort you can calculate ‘arrayLen-1’ outside of all loops and store that into a variable. | Change  for (index = 0; index < arrayLen-1; index++)  To  int arrayLenMinusOne = arrayLen - 1; //ouside of all loops  …  for(index = 0; index < arrayLenMinusOne; index++) | | You can apply this to both |
| 6B | In insertionSort you can calculate ‘arrayLen-1’ outside of all loops and store that into a variable. | Change  for (outerIndex = 0; outerIndex < arrayLen-1; outerIndex++)  To  int arrayLenMinusOne = arrayLen -1;  …  for (outerIndex = 0; outerIndex < arrayLenMinusOne; outerIndex++) | | You can apply this to both |
| 6C. | In the main(), you can move “char text[TEXT\_LEN]” outside of the do while loop and clear it each time the loop restarts. This way you don’t waste memory or worry about garbage collection for recreating the text[] variable. | Change  do  {  char text[TEXT\_LEN];  To  char text[TEXT\_LEN];  do  {  text = ""; | | You can apply this to both |
| 6C. | You can rewrite the switch statement into an if else block. Using a switch increases compile time (because it creates a jump table) and is inefficient when it comes to just two boolean/case values. Since the do while loop already restricts the values to be just 1 or 2 then an if else is more efficient. | Change  switch (choice)  {  case 1 :  insertionSort(ARRAY\_LEN,array);  break;  case 2 :  bubbleSort(ARRAY\_LEN,array);  break;  }  To  if(choice == 1)  insertionSort(ARRAY\_LEN, array);  else  bubbleSort(ARRAY\_LEN, array); | | You can apply this to both |