3. wordCounterO0
buildTree() self seconds (words with tree): almost instant
buildTree() self seconds (words with linked-list): ~2 seconds
4x version
buildTree() self seconds (words with tree): almost instant
buildTree() self seconds (words with linked-list): ~12 seconds

4. wordCounterO2
buildTree() self seconds (words with tree): almost instant
buildTree() self seconds (words with linked-list): ~2 seconds
4x version
buildTree() self seconds (words with tree): almost instant
buildTree() self seconds (words with linked-list): ~7 seconds

5.Which is faster?

- **A bad algorithm and data-structure optimized with -O2**
    - **This one is faster as shown with buildTree() with linked list being faster with O2 than O0**
- A good algorithm and data-structure optimized with -O0

6. Parts of an executable

Please find the following inside of wordCounterO0 by using objdump to show it (if it exists in the executable) or by using objdump to disassemble the code and showing where the code manipulates the heap or stack.

Show a *disassembly* or *objdump*. You do not have to show *all* of the objdump result if it is too long, but (1) please show the relevant output, and (2) please show the objdump command that you used to generate it.

A. The string "File name to read: " in main()
B. The local variable rootPtr in buildTree()
C. The code for printList()
D. The global variable textLen

Question	Command	Result
(A)	401248 46696c65 206e616d 6520746f 20726561 401258 643a2000 7200436f 756c6420	 File name to rea

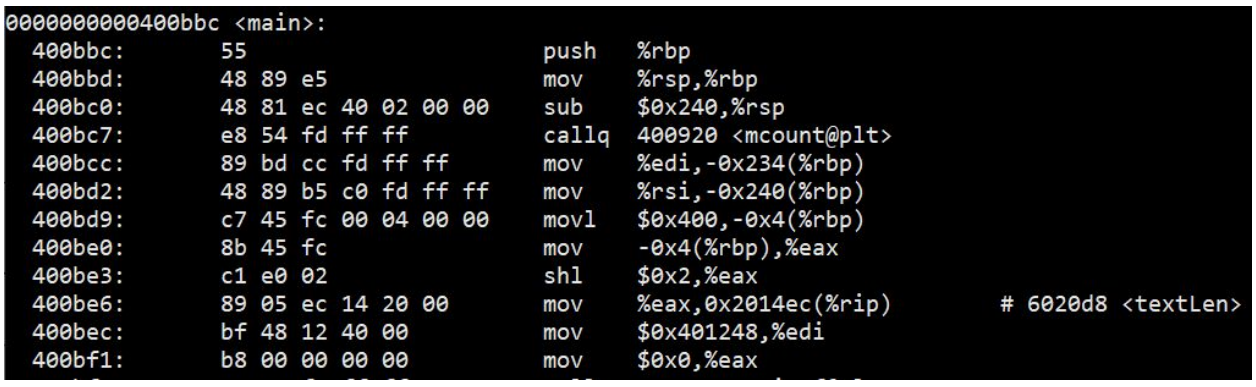
	6e6f7420	d: .r.Could not
(B)	Because its a local variable, there is no stack to hold the variable	N/A
(C)	00000000004010e1 <printList>: 4010e1: 55 push %rbp 4010e2: 48 89 e5 mov %rsp,%rbp 4010e5: 48 83 ec 20 sub $0x20,%rsp 4010e9: e8 32 f8 ff ff callq 400920 <mcount@plt> 4010ee: 48 89 7d e8 mov %rdi,-0x18(%rbp) 4010f2: 48 8b 45 e8 mov -0x18(%rbp),%rax 4010f6: 48 89 45 f8 mov %rax,-0x8(%rbp) 4010fa: eb 2c jmp 401128 <printList+0x47> 4010fc: 48 8b 45 f8 mov -0x8(%rbp),%rax 401100: 8b 50 08 mov 0x8(%rax),%edx 401103: 48 8b 45 f8 mov -0x8(%rbp),%rax 401107: 48 8b 00 mov (%rax),%rax 40110a: 48 89 c6 mov %rax,%rsi 40110d: bf f2 12 40 00 mov $0x4012f2,%edi 401112: b8 00 00 00 00 mov $0x0,%eax 401117: e8 64 f7 ff ff	Prints the list

	callq 400880 <printf@plt> 40111c: 48 8b 45 f8 mov -0x8(%rbp),%rax 401120: 48 8b 40 10 mov 0x10(%rax),%rax 401124: 48 89 45 f8 mov %rax,-0x8(%rbp) 401128: 48 83 7d f8 00 cmpq $0x0,-0x8(%rbp) 40112d: 75 cd jne 4010fc <printList+0x1b> 40112f: c9 leaveq 401130: c3 retq	
(D)	400d54: 8b 05 7e 13 20 00 mov 0x20137e(%rip),%eax	# 6020d8 <textLen>

7.

Example of variables being kept in ram in O0 vs optimized O2 having less kept in registers

O0:



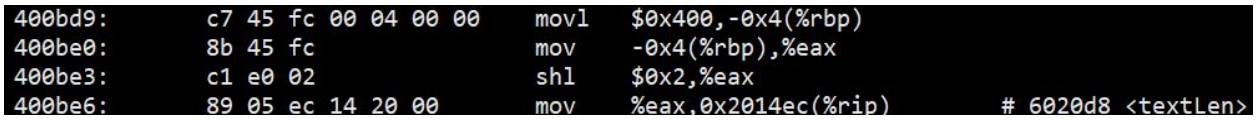
02:

```
0000000000400980 <main>:
  400980:       55                      push   %rbp
  400981:       48 89 e5                mov    %rsp,%rbp
  400984:       41 54                   push   %r12
  400986:       53                      push   %rbx
  400987:       48 81 ec 00 02 00 00    sub    $0x200,%rsp
  40098e:       e8 ad ff ff ff          callq  400940 <mcount@plt>
  400993:       bf 88 11 40 00          mov    $0x401188,%edi
  400998:       31 c0                   xor    %eax,%eax
  40099a:       c7 05 34 17 20 00 00    movl   $0x1000,0x201734(%rip)        # 6020d8 <textLen>
  4009a1:       10 00 00
```

Example 2

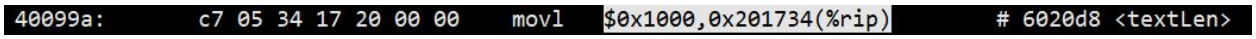
Here I have the initialization of textLen, in O0 it's shown to have more movement and calculations to make the number 4096. In O2 it's shown to just make textLen by giving it the hexadecimal 1000 in one movl function. This is an example of reduction in strength as its a cheaper operation in 02.

O0:



```
400bd9:       c7 45 fc 00 04 00 00    movl   $0x400,-0x4(%rbp)
400be0:       8b 45 fc                mov    -0x4(%rbp),%eax
400be3:       c1 e0 02                shl    $0x2,%eax
400be6:       89 05 ec 14 20 00       mov    %eax,0x2014ec(%rip)        # 6020d8 <textLen>
```

O2:



```
40099a:       c7 05 34 17 20 00 00    movl   $0x1000,0x201734(%rip)        # 6020d8 <textLen>
```