Assignment 1 Answers

3) a. How for 3 million numbers how many self seconds did generateList() take?

ANSWER: 0.96 Seconds

```
48: 29992 time(s)
[[pshoema2@cdmlinux Assignment1]$ gprof assign1-0
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 99.39      0.96     0.96        1   964.05   974.09  generateList
  1.04      0.97     0.01  3000000     0.00     0.00  getNextNumber
  0.00      0.97     0.00        4     0.00     0.00  promptUser
  0.00      0.97     0.00        3     0.00     0.00  obtainNumberBetween
  0.00      0.97     0.00        1     0.00   974.09  countWithList
  0.00      0.97     0.00        1     0.00     0.00  freeList
  0.00      0.97     0.00        1     0.00     0.00  printList
```

3) b. How for 3 million numbers how many self seconds did generateTree() take?

ANSWER: 0.20 Seconds

```
Choice 2
[[pshoema2@cdmlinux Assignment1]$ gprof assign1-0
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 80.34      0.20     0.20        1   200.84   251.05  generateTree
 20.08      0.25     0.05  3000000     0.00     0.00  getNextNumber
  0.00      0.25     0.00        4     0.00     0.00  promptUser
  0.00      0.25     0.00        3     0.00     0.00  obtainNumberBetween
  0.00      0.25     0.00        1     0.00   251.05  countWithTree
  0.00      0.25     0.00        1     0.00     0.00  freeTree
  0.00      0.25     0.00        1     0.00     0.00  printTree
```

4) a. How for 3 million numbers how many self seconds did generateList() take?

ANSWER: 0.36 Seconds

```
48: 29992 time(s)
[[pshoema2@cdmlinux Assignment1]$ gprof assign1-2
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 97.79      0.36     0.36        1   361.81   371.86  generateList
  2.72      0.37     0.01  3000000     0.00     0.00  getNextNumber
  0.00      0.37     0.00        1     0.00     0.00  printList

 %         the percentage of the total running time of the
time       program used by this function
```

4) b. How for 3 million numbers how many self seconds did generateTree() take?

ANSWER: 0.09 Seconds

```
Choice 2
[[pshoema2@cdmlinux Assignment1]$ gprof assign1-2
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 69.58      0.09     0.09        1    90.45   125.63  generateTree
 27.06      0.13     0.04  3000000     0.00     0.00  getNextNumber
  3.87      0.13     0.01                             frame_dummy
  0.00      0.13     0.00        1     0.00     0.00  printTree
```

5)

Which is faster:

- A bad algorithm and data-structure optimized with -O2
- A good algorithm and data-structure optimized with -O0

ANSWER: A good algorithm and data-structure optimized with -O0

6) a. The string "%d: %d time(s)\n" in printList()

ANSWER:

Command: objdump -s -j .rodata assign1-0

Result:

```
[pshoema2@cdmlinux Assignment1]$ objdump -s -j .rodata assign1-0

assign1-0:     file format elf64-x86-64

Contents of section .rodata:
 400e58 01000200 00000000 00000000 00000000  ................
 400e68 506c6561 73652065 6e746572 20257320  Please enter %s
 400e78 25642074 68726f75 67682025 64002564  %d through %d.%d
 400e88 00000000 00000000 74686520 6c6f7765  ........the lowe
 400e98 7374206e 756d6265 7220696e 20746865  st number in the
 400ea8 2072616e 67650000 74686520 68696768   range..the high
 400eb8 65737420 6e756d62 65722069 6e207468  est number in th
 400ec8 65207261 6e676500 74686520 6e756d62  e range.the numb
 400ed8 6572206f 66206e75 6d626572 7320746f  er of numbers to
 400ee8 20636f6e 73696465 72000000 00000000   consider.......
 400ef8 57686174 20776f75 6c642079 6f75206c  What would you l
 400f08 696b6520 746f2064 6f3f0a28 31292043  ike to do?.(1) C
 400f18 6f756e74 20776974 68206120 6c697374  ount with a list
 400f28 0a283229 20436f75 6e742077 69746820  .(2) Count with
 400f38 61207472 65650a28 30292051 7569740a  a tree.(0) Quit.
 400f48 596f7572 2063686f 69636520 0043686f  Your choice .Cho
 400f58 69636520 31004368 6f696365 20320043  ice 1.Choice 2.C
 400f68 686f6963 6520302c 20457869 74002564  hoice 0, Exit.%d
 400f78 3a202564 2074696d 65287329 0a002564  : %d time(s)..%d
 400f88 3a202564 2074696d 65287329 0a00      : %d time(s)..
```

6) b. The code for getNextNumber()

ANSWER:

Command: objdump -d -j .text assign1-0

Result:

```
000000000040081d <getNextNumber>:
  40081d:       55                      push   %rbp
  40081e:       48 89 e5                mov    %rsp,%rbp
  400821:       e8 8a fe ff ff          callq  4006b0 <mcount@plt>
  400826:       e8 b5 fe ff ff          callq  4006e0 <rand@plt>
  40082b:       8b 0d 57 18 20 00       mov    0x201857(%rip),%ecx        # 602088 <high>
  400831:       8b 15 55 18 20 00       mov    0x201855(%rip),%edx        # 60208c <low>
  400837:       29 d1                   sub    %edx,%ecx
  400839:       89 ca                   mov    %ecx,%edx
  40083b:       8d 4a 01                lea    0x1(%rdx),%ecx
  40083e:       99                      cltd
  40083f:       f7 f9                   idiv   %ecx
  400841:       8b 05 45 18 20 00       mov    0x201845(%rip),%eax        # 60208c <low>
  400847:       01 d0                   add    %edx,%eax
  400849:       5d                      pop    %rbp
  40084a:       c3                      retq
```

6) c. The global variable high

ANSWER:

Command: objdump -s -t assign1-0

Result:

```
0000000000000000       F *UND*  0000000000000000              putchar@@GLIBC_2.2.5
0000000000602088 g     O .bss   0000000000000004              high
0000000000602078  w      .data  0000000000000000              data_start
0000000000000000       F *UND*  0000000000000000              puts@@GLIBC_2.2.5
```

6) d. treePtr in countWithTree()

ANSWER:

Command: objdump -d -j .text assign1-0; although treePtr specifically cannot be identified as it is a pointer and will be created on the stack at run-time

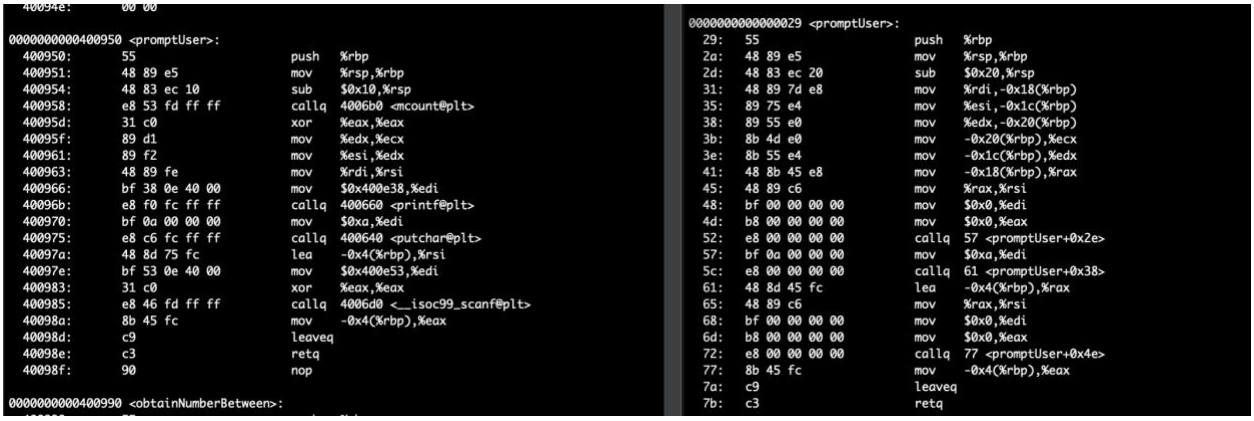
Result:

```
0000000000400bd5 <countWithTree>:
  400bd5:       55                      push   %rbp
  400bd6:       48 89 e5                mov    %rsp,%rbp
  400bd9:       48 83 ec 20             sub    $0x20,%rsp
  400bdd:       e8 ce fa ff ff          callq  4006b0 <mcount@plt>
  400be2:       89 7d ec                mov    %edi,-0x14(%rbp)
  400be5:       8b 45 ec                mov    -0x14(%rbp),%eax
  400be8:       89 c7                   mov    %eax,%edi
  400bea:       e8 26 fe ff ff          callq  400a15 <generateTree>
  400bef:       48 89 45 f8             mov    %rax,-0x8(%rbp)
  400bf3:       48 8b 45 f8             mov    -0x8(%rbp),%rax
  400bf7:       48 89 c7                mov    %rax,%rdi
  400bfa:       e8 38 ff ff ff          callq  400b37 <printTree>
  400bff:       48 8b 45 f8             mov    -0x8(%rbp),%rax
  400c03:       48 89 c7                mov    %rax,%rdi
  400c06:       e8 84 ff ff ff          callq  400b8f <freeTree>
  400c0b:       c9                      leaveq
  400c0c:       c3                      retq
```

7) Optimizations:

1: Using registers to hold variables. The left screen shows the optimized code while the right reflects not optimized code.

The optimized “promptUser” function stores variables more effectively using registers as shown below.

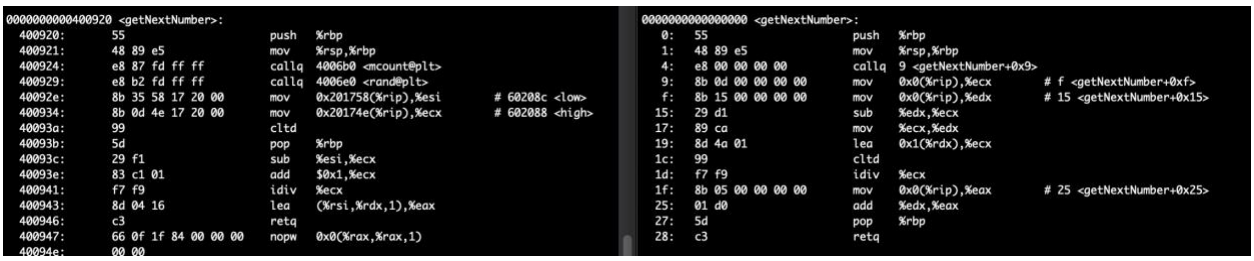


```
  40094e:       00 00

0000000000400950 <promptUser>:
  400950:       55                      push   %rbp
  400951:       48 89 e5                mov    %rsp,%rbp
  400954:       48 83 ec 10             sub    $0x10,%rsp
  400958:       e8 53 fd ff ff          callq  4006b0 <mcount@plt>
  40095d:       31 c0                   xor    %eax,%eax
  40095f:       89 d1                   mov    %edx,%ecx
  400961:       89 f2                   mov    %esi,%edx
  400963:       48 89 fe                mov    %rdi,%rsi
  400966:       bf 38 0e 40 00          mov    $0x400e38,%edi
  40096b:       e8 f0 fc ff ff          callq  400660 <printf@plt>
  400970:       bf 0a 00 00 00          mov    $0xa,%edi
  400975:       e8 c6 fc ff ff          callq  400640 <putchar@plt>
  40097a:       48 8d 75 fc             lea    -0x4(%rbp),%rsi
  40097e:       bf 53 0e 40 00          mov    $0x400e53,%edi
  400983:       31 c0                   xor    %eax,%eax
  400985:       e8 46 fd ff ff          callq  4006d0 <__isoc99_scanf@plt>
  40098a:       8b 45 fc                mov    -0x4(%rbp),%eax
  40098d:       c9                      leaveq
  40098e:       c3                      retq
  40098f:       90                      nop

0000000000400990 <obtainNumberBetween>:
```

```
0000000000000029 <promptUser>:
  29:   55                      push   %rbp
  2a:   48 89 e5                mov    %rsp,%rbp
  2d:   48 83 ec 20             sub    $0x20,%rsp
  31:   48 89 7d e8             mov    %rdi,-0x18(%rbp)
  35:   89 75 e4                mov    %esi,-0x1c(%rbp)
  38:   89 55 e0                mov    %edx,-0x20(%rbp)
  3b:   8b 4d e0                mov    -0x20(%rbp),%ecx
  3e:   8b 55 e4                mov    -0x1c(%rbp),%edx
  41:   48 8b 45 e8             mov    -0x18(%rbp),%rax
  45:   48 89 c6                mov    %rax,%rsi
  48:   bf 00 00 00 00          mov    $0x0,%edi
  4d:   b8 00 00 00 00          mov    $0x0,%eax
  52:   e8 00 00 00 00          callq  57 <promptUser+0x2e>
  57:   bf 0a 00 00 00          mov    $0xa,%edi
  5c:   e8 00 00 00 00          callq  61 <promptUser+0x38>
  61:   48 8d 45 fc             lea    -0x4(%rbp),%rax
  65:   48 89 c6                mov    %rax,%rsi
  68:   bf 00 00 00 00          mov    $0x0,%edi
  6d:   b8 00 00 00 00          mov    $0x0,%eax
  72:   e8 00 00 00 00          callq  77 <promptUser+0x4e>
  77:   8b 45 fc                mov    -0x4(%rbp),%eax
  7a:   c9                      leaveq
  7b:   c3                      retq
```

2: Using registers to hold variables. The left screen shows the optimized code while the right reflects not optimized code.

The optimized “getNextNumber” function stores variables more effectively using registers as shown below.



```
0000000000400920 <getNextNumber>:
  400920:       55                      push   %rbp
  400921:       48 89 e5                mov    %rsp,%rbp
  400924:       e8 87 fd ff ff          callq  4006b0 <mcount@plt>
  400929:       e8 b2 fd ff ff          callq  4006e0 <rand@plt>
  40092e:       8b 35 58 17 20 00       mov    0x201758(%rip),%esi        # 60208c <low>
  400934:       8b 0d 4e 17 20 00       mov    0x20174e(%rip),%ecx        # 602088 <high>
  40093a:       99                      cltd
  40093b:       5d                      pop    %rbp
  40093c:       29 f1                   sub    %esi,%ecx
  40093e:       83 c1 01                add    $0x1,%ecx
  400941:       f7 f9                   idiv   %ecx
  400943:       8d 04 16                lea    (%rsi,%rdx,1),%eax
  400946:       c3                      retq
  400947:       66 0f 1f 84 00 00 00    nopw   0x0(%rax,%rax,1)
  40094e:       00 00
```

```
0000000000000000 <getNextNumber>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   e8 00 00 00 00          callq  9 <getNextNumber+0x9>
   9:   8b 0d 00 00 00 00       mov    0x0(%rip),%ecx        # f <getNextNumber+0xf>
   f:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 15 <getNextNumber+0x15>
  15:   29 d1                   sub    %edx,%ecx
  17:   89 ca                   mov    %ecx,%edx
  19:   8d 4a 01                lea    0x1(%rdx),%ecx
  1c:   99                      cltd
  1d:   f7 f9                   idiv   %ecx
  1f:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 25 <getNextNumber+0x25>
  25:   01 d0                   add    %edx,%eax
  27:   5d                      pop    %rbp
  28:   c3                      retq
```