CS232 Lab 3
Computer Architecture

Arnav Aditya Singh
210050018

1 Q1

For analysis I use the command `objdump -CD -M intel --no-show-raw-insn <exec_name>` (removing the `--no-show-raw-insn` flag to see bytes when required)

1.1 part_a

```
0000000000001443 <main>:
    1443: endbr64
    1447: push rbp
    1448: mov rbp,rsp
    144b: push r13
    144d: push r12
    144f: push rbx
    1450: sub rsp,0x38

    1454: mov rax,QWORD PTR fs:0x28
    145d: mov QWORD PTR [rbp-0x28],rax
    1461: xor eax,eax

    1463: lea rsi,[rip+0x1cce] # 3138 <std::__detail::_S_invalid_state_id+0x90>
    146a: lea rdi,[rip+0x3bcf] # 5040 <std::cout@@GLIBCXX_3.4>
    1471: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    1476: mov DWORD PTR [rbp-0x40],0x1e
    147d: mov DWORD PTR [rbp-0x3c],0x2e
    1484: mov DWORD PTR [rbp-0x38],0x38
    148b: mov DWORD PTR [rbp-0x34],0x39
    1492: mov DWORD PTR [rbp-0x30],0x36
    1499: mov DWORD PTR [rbp-0x2c],0x0
    14a0: lea rax,[rbp-0x40]
    14a4: mov r12,rax
    14a7: mov r13d,0x6
    14ad: mov rcx,r12
    14b0: mov rbx,r13
    14b3: mov rax,r12
    14b6: mov rdx,r13
    14b9: mov rax,rdx
    14bc: mov rsi,rcx
    14bf: mov rdx,rax
    14c2: lea rdi,[rip+0x3db7] # 5280 <v>
    14c9: call 175c <std::vector<int, std::allocator<int> >::operator=(std::initializer_list<int>)>

    14ce: lea rsi,[rip+0x1cb3] # 3188 <std::__detail::_S_invalid_state_id+0xe0>
    14d5: lea rdi,[rip+0x3b64] # 5040 <std::cout@@GLIBCXX_3.4>
    14dc: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    14e1: lea rax,[rbp-0x48]
    14e5: mov rsi,rax
    14e8: lea rdi,[rip+0x3c71] # 5160 <std::cin@@GLIBCXX_3.4>
    14ef: call 11a0 <std::istream::operator>>(int&)@plt>

    14f4: lea rsi,[rip+0x1ca5] # 31a0 <std::__detail::_S_invalid_state_id+0xf8>
    14fb: lea rdi,[rip+0x3b3e] # 5040 <std::cout@@GLIBCXX_3.4>
    1502: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    1507: lea rax,[rbp-0x44]
    150b: mov rsi,rax
    150e: lea rdi,[rip+0x3c4b] # 5160 <std::cin@@GLIBCXX_3.4>
    1515: call 11a0 <std::istream::operator>>(int&)@plt>

    151a: mov edx,DWORD PTR [rbp-0x48]
    151d: mov eax,DWORD PTR [rbp-0x44]
    1520: add eax,edx
```

```
1522: mov edi,eax
1524: call 11d0 <srand@plt>

1529: mov eax,DWORD PTR [rbp-0x44]
152c: mov edi,eax
152e: call 1369 <part_a(int)>

1533: mov eax,0x0
1538: mov rbx,QWORD PTR [rbp-0x28]
153c: xor rbx,QWORD PTR fs:0x28
1545: je 154c <main+0x109>
1547: call 1200 <__stack_chk_fail@plt>

154c: add rsp,0x38
1550: pop rbx
1551: pop r12
1553: pop r13
1555: pop rbp
1556: ret
```

- `0x1443-0x1450` Function frame setup, allocating `0x38` bytes for local variables
- `0x1454-0x1461` (and `0x1533-0x1547`) stack protection check, see this answer
- `0x1463-0x1471` Printing `"============================= Welcome to Part I! =============================\n"`; parameters are
  - `rdi` = pointer to `std::cout` (here `0x5040`)
  - `rsi` = pointer to null-ended char array (here `0x3138`)
- `0x1476-0x14c9` Assign to static uninitialised `std::vector<int> v` using initializer list `{0x1e,0x2e,0x38,0x39,0x36,0x0}` (*static uninitialised* as `v` is in `.bss` section); parameters are
  - `rdi` = pointer to `v` (here `0x5280`)
  - `rsi` = pointer to the list (here `rbp-0x40` obtained at `0x14a0`)
  - `rdx` = the length of the list (here `0x06` obtained at `0x14a7`
- `0x14ce-0x14dc` Printing `"Enter your roll number:"` (address `0x3188` contains the null-ended string)
- `0x14e1-0x14ef` Taking roll number input `roll`, stored at address `rbp-0x48`[1]; parameters are
  - `rdi` = pointer to `std::cin` (here `0x5160`)
  - `rsi` = pointer to location at which input is to be stored (here `int` to be stored at `4` bytes starting at `rbp-0x48`)
- `0x14f4-0x1502` Printing `"Enter the key to unlock this: "` (address `0x31a0` contains the null-ended string)
- `0x1507-0x1515` Taking key input `key`, stored at address `rbp-0x44`[1]
- `0x151a-0x1524` Calling `srand(roll + key)`; parameters are
  - `rdi` = `roll + key`, obtained by adding `[rbp-0x48]` and `[rbp-0x44]`
- `0x1529-0x152e` Calling `part_a(key)`; parameters are
  - `rdi` = `key`, obtained from `[rbp-0x44]`
- `0x154c-0x1556` Function frame dismantle, return

[1] These are inside the allocated space for local variables as we push 3 registers (`0x18` bytes) on the stack after setting `rbp` and before subtracting `0x38` from `rsp`, i.e. local variables are stored in `rsp=rbp-0x50` to `rbp-0x18`.

```
0000000000001369 <part_a(int)>:
    1369: endbr64
    136d: push rbp
    136e: mov rbp,rsp
    1371: push rbx
    1372: sub rsp,0x28

    1376: mov DWORD PTR [rbp-0x24],edi
    1379: cmp DWORD PTR [rbp-0x24],0x1387
    1380: jle 1428 <part_a(int)+0xbf>

    1386: mov DWORD PTR [rbp-0x14],0x0
    138d: lea rdi,[rip+0x3eec] # 5280 <v>
    1394: call 1710 <std::vector<int, std::allocator<int> >::size() const>
    1399: cmp DWORD PTR [rbp-0x14],eax
    139c: setl al
    139f: test al,al
    13a1: je 13df <part_a(int)+0x76>
    13a3: mov rbx,QWORD PTR [rip+0x3c66] # 5010 <letters>
    13aa: mov eax,DWORD PTR [rbp-0x14]
    13ad: cdqe
    13af: mov rsi,rax
    13b2: lea rdi,[rip+0x3ec7] # 5280 <v>
    13b9: call 1738 <std::vector<int, std::allocator<int> >::operator[](unsigned long)>
    13be: mov eax,DWORD PTR [rax]
    13c0: cdqe
    13c2: add rax,rbx
    13c5: movzx eax,BYTE PTR [rax]
    13c8: movsx eax,al
    13cb: mov esi,eax
    13cd: lea rdi,[rip+0x3c6c] # 5040 <std::cout@@GLIBCXX_3.4>
    13d4: call 1210 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
       ↪ basic_ostream<char, std::char_traits<char> >&, char)@plt>
    13d9: add DWORD PTR [rbp-0x14],0x1
    13dd: jmp 138d <part_a(int)+0x24>__detail

    13df: lea rsi,[rip+0x1d23] # 3109 <std::__detail::_S_invalid_state_id+0x61>
    13e6: lea rdi,[rip+0x3c53] # 5040 <std::cout@@GLIBCXX_3.4>
    13ed: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
       ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>
    13f2: lea rsi,[rip+0x1d12] # 310b <std::__detail::_S_invalid_state_id+0x63>
    13f9: lea rdi,[rip+0x3c40] # 5040 <std::cout@@GLIBCXX_3.4>
    1400: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
       ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    1405: mov rbx,rax
    1408: call 1160 <rand@plt>
    140d: mov esi,eax
    140f: mov rdi,rbx
    1412: call 1260 <std::ostream::operator<<(int)@plt>

    1417: lea rsi,[rip+0x1ceb] # 3109 <std::__detail::_S_invalid_state_id+0x61>
    141e: mov rdi,rax
    1421: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
       ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>
    1426: jmp 143b <part_a(int)+0xd2>

    1428: lea rsi,[rip+0x1cf4] # 3123 <std::__detail::_S_invalid_state_id+0x7b>
    142f: lea rdi,[rip+0x3c0a] # 5040 <std::cout@@GLIBCXX_3.4>
    1436: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
       ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    143b: nop
    143c: add rsp,0x28
    1440: pop rbx
    1441: pop rbp
    1442: ret
```

- `0x1369-0x1372` Function frame setup, allocating `0x28` bytes for local variables
- `0x1376-0x1380` KEY Comparing argument (here `key`) to `0x1387` = 4999, and jumping to `0x1428` if `key` $\leq$ 4999
- `01386-0x13dd` Here the following takes place

  ```
  char const* letters = "{}abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_!";
  ...
  for (int i = 0; i < v.size(); ++i)
  std::cout << *(letters + v[i]);
  ```

  - pointer `letters` is stored at `0x5010` (in section `.data`) and the character `'{'` at `0x30b0`
  - variable `i` is stored at `rbp-0x14`
  - comparison `i < v.size()`. takes place in `0x138d-0x13a1`

  This corresponds to printing `CS230{`

- `0x13df-0x1400` Printing `"\n"` and `"Your secret number is: "` (addresses `0x3109` and `0x310b` respectively contain the null-ended strings)
- `0x1405-9x1412` Calling `rand()` and printing the result. This is the secret number.
- `0x1417-0x1426` Printing `"\n"` (address `0x3109` contains the null-ended string) and jumping over printing the failure message below.
- `0x1428-0x1436` Printing `"Sorry, Try Again!!!\n"` (address `0x3123` contains the null-ended string)
- `0x143b-0x1442` Function frame dismatle, return

In summary, any KEY `key > 4999` will suffice.

```
 $ ./part_a
============================= Welcome to Part I! =============================
Enter your roll number:210050018
Enter the key to unlock this: 5000
CS230{
Your secret number is: 510332108
```

1.2 part_b

```
000000000000145a <main>:
    145a: endbr64
    145e: push rbp
    145f: mov rbp,rsp
    1462: push r13
    1464: push r12
    1466: push rbx
    1467: sub rsp,0x48

    146b: mov rax,QWORD PTR fs:0x28
    1474: mov QWORD PTR [rbp-0x28],rax
    1478: xor eax,eax

    147a: lea rsi,[rip+0x1cb7] # 3138 <std::__detail::_S_invalid_state_id+0x90>
    1481: lea rdi,[rip+0x3bb8] # 5040 <std::cout@@GLIBCXX_3.4>
    1488: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    148d: mov DWORD PTR [rbp-0x50],0xa
    1494: mov DWORD PTR [rbp-0x4c],0x14
    149b: mov DWORD PTR [rbp-0x48],0x40
    14a2: mov DWORD PTR [rbp-0x44],0x6
    14a9: mov DWORD PTR [rbp-0x40],0x2
    14b0: mov DWORD PTR [rbp-0x3c],0x14
    14b7: mov DWORD PTR [rbp-0x38],0x1a
    14be: mov DWORD PTR [rbp-0x34],0x41
    14c5: mov DWORD PTR [rbp-0x30],0x41
    14cc: mov DWORD PTR [rbp-0x2c],0x1
    14d3: lea rax,[rbp-0x50]
    14d7: mov r12,rax
    14da: mov r13d,0xa
    14e0: mov rcx,r12
    14e3: mov rbx,r13
    14e6: mov rax,r12
    14e9: mov rdx,r13
    14ec: mov rax,rdx
    14ef: mov rsi,rcx
    14f2: mov rdx,rax
    14f5: lea rdi,[rip+0x3d84] # 5280 <v>
    14fc: call 17c8 <std::vector<int, std::allocator<int> >::operator=(std::initializer_list<int>)>

    1501: lea rsi,[rip+0x1c81] # 3189 <std::__detail::_S_invalid_state_id+0xe1>
    1508: lea rdi,[rip+0x3b31] # 5040 <std::cout@@GLIBCXX_3.4>
    150f: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    1514: lea rax,[rbp-0x54]
    1518: mov rsi,rax
    151b: lea rdi,[rip+0x3c3e] # 5160 <std::cin@@GLIBCXX_3.4>
    1522: call 11a0 <std::istream::operator>>(int&)@plt>

    1527: lea rsi,[rip+0x1c7a] # 31a8 <std::__detail::_S_invalid_state_id+0x100>
    152e: lea rdi,[rip+0x3b0b] # 5040 <std::cout@@GLIBCXX_3.4>
    1535: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    153a: lea rax,[rbp-0x60]
    153e: mov rsi,rax
    1541: lea rdi,[rip+0x3c18] # 5160 <std::cin@@GLIBCXX_3.4>
    1548: call 11a0 <std::istream::operator>>(int&)@plt>

    154d: mov rdx,rax
    1550: lea rax,[rbp-0x5c]
    1554: mov rsi,rax
    1557: mov rdi,rdx
    155a: call 11a0 <std::istream::operator>>(int&)@plt>
```

```
155f: mov rdx,rax
1562: lea rax,[rbp-0x58]
1566: mov rsi,rax
1569: mov rdi,rdx
156c: call 11a0 <std::istream::operator>>(int&)@plt>

1571: mov eax,DWORD PTR [rbp-0x54]
1574: mov edx,eax
1576: mov eax,DWORD PTR [rbp-0x60]
1579: add edx,eax
157b: mov eax,DWORD PTR [rbp-0x5c]
157e: add edx,eax
1580: mov eax,DWORD PTR [rbp-0x58]
1583: add eax,edx
1585: mov edi,eax
1587: call 11d0 <srand@plt>

158c: mov edx,DWORD PTR [rbp-0x58]
158f: mov ecx,DWORD PTR [rbp-0x5c]
1592: mov eax,DWORD PTR [rbp-0x60]
1595: mov esi,ecx
1597: mov edi,eax
1599: call 1369 <part_b(int, int, int)>

159e: mov eax,0x0
15a3: mov rbx,QWORD PTR [rbp-0x28]
15a7: xor rbx,QWORD PTR fs:0x28
15b0: je 15b7 <main+0x15d>
15b2: call 1200 <__stack_chk_fail@plt>

15b7: add rsp,0x48
15bb: pop rbx
15bc: pop r12
15be: pop r13
15c0: pop rbp
15c1: ret
```

- `0x145a-0x1467` Function frame setup, allocating `0x48` bytes for local variables
- `0x146b-0x1478` (and `0x159e-0x15b2`) stack protection check, see this answer
- `0x147a-0x1488` Printing `"============================= Welcome to Part II! =============================\n"` (address `0x3138` contains the null-ended string)
- `0x148d-0x14fc` Assign to static uninitialised `std::vector<int> v` using initializer list `{0xa,0x14,0x40,0x6,0x2,0x14,0x1a,0x41,`
  (*static uninitialised* as `v` is in `.bss` section)
- `0x1501-0x150f` Printing `"Enter your roll number:"` (address `0x3189` contains the null-ended string)
- `0x1514-0x1522` Taking roll number input `roll`, stored at address `rbp-0x54`
- `0x1527-0x1535` Printing `"Enter the keys to unlock this: "` (address `0x31a8` contains the null-ended string)
- `0x1507-0x1548,0x154d-0x155a,0x155f-0x156c` Taking key inputs `key1,key2,key3`, stored at addresses `rbp-0x60,rbp-0x5c,rbp-`
- `0x1571-0x1587` Calling `srand(roll + key1 + key2 + key3)`
- `0x1529-0x152e` Calling `part_b(key1,key2,key3)`
- `0x154c-0x1556` Function frame dismantle, return

```
0000000000001369 <part_b(int, int, int)>:
    1369: endbr64
    136d: push rbp
    136e: mov rbp,rsp
    1371: push rbx
    1372: sub rsp,0x28

    1376: mov DWORD PTR [rbp-0x24],edi
    1379: mov DWORD PTR [rbp-0x28],esi
    137c: mov DWORD PTR [rbp-0x2c],edx
    137f: mov eax,DWORD PTR [rbp-0x24]
    1382: imul eax,eax
    1385: mov edx,eax
    1387: mov eax,DWORD PTR [rbp-0x28]
    138a: imul eax,eax
    138d: add edx,eax
    138f: mov eax,DWORD PTR [rbp-0x2c]
    1392: imul eax,eax
    1395: cmp edx,eax
    1397: jne 143f <part_b(int, int, int)+0xd6>

    139d: mov DWORD PTR [rbp-0x14],0x0
    13a4: lea rdi,[rip+0x3ed5] # 5280 <v>
    13ab: call 177c <std::vector<int, std::allocator<int> >::size() const>
    13b0: cmp DWORD PTR [rbp-0x14],eax
    13b3: setl al
    13b6: test al,al
    13b8: je 13f6 <part_b(int, int, int)+0x8d>
    13ba: mov rbx,QWORD PTR [rip+0x3c4f] # 5010 <letters>
    13c1: mov eax,DWORD PTR [rbp-0x14]
    13c4: cdqe
    13c6: mov rsi,rax
    13c9: lea rdi,[rip+0x3eb0] # 5280 <v>
    13d0: call 17a4 <std::vector<int, std::allocator<int> >::operator[](unsigned long)>
    13d5: mov eax,DWORD PTR [rax]
    13d7: cdqe
    13d9: add rax,rbx
    13dc: movzx eax,BYTE PTR [rax]
    13df: movsx eax,al
    13e2: mov esi,eax
    13e4: lea rdi,[rip+0x3c55] # 5040 <std::cout@@GLIBCXX_3.4>
    13eb: call 1210 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char)@plt>
    13f0: add DWORD PTR [rbp-0x14],0x1
    13f4: jmp 13a4 <part_b(int, int, int)+0x3b>

    13f6: lea rsi,[rip+0x1d0c] # 3109 <std::__detail::_S_invalid_state_id+0x61>
    13fd: lea rdi,[rip+0x3c3c] # 5040 <std::cout@@GLIBCXX_3.4>
    1404: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    1409: lea rsi,[rip+0x1cfb] # 310b <std::__detail::_S_invalid_state_id+0x63>
    1410: lea rdi,[rip+0x3c29] # 5040 <std::cout@@GLIBCXX_3.4>
    1417: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

    141c: mov rbx,rax
    141f: call 1160 <rand@plt>
    1424: mov esi,eax
    1426: mov rdi,rbx
    1429: call 1260 <std::ostream::operator<<(int)@plt>

    142e: lea rsi,[rip+0x1cd4] # 3109 <std::__detail::_S_invalid_state_id+0x61>
    1435: mov rdi,rax
    1438: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
        ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>
    143d: jmp 1452 <part_b(int, int, int)+0xe9>
```

```
143f: lea rsi,[rip+0x1cdd] # 3123 <std::__detail::_S_invalid_state_id+0x7b>
1446: lea rdi,[rip+0x3bf3] # 5040 <std::cout@@GLIBCXX_3.4>
144d: call 11e0 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
    ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

1452: nop
1453: add rsp,0x28
1457: pop rbx
1458: pop rbp
1459: ret
```

- `0x1369-0x1372` Function frame setup, allocating `0x28` bytes for local variables
- `0x1376-0x1397` Storing arguments (here `key1,key2,key3`) to local variables (addresses `rbp-0x24,rbp-0x28,rbp-0x2c`) and KEY comparing $\texttt{key1}^2+\texttt{key2}^2$ and $\texttt{key3}^2$, jumping to `0x143f` if not equal
- `0139d-0x13f4` Here the following takes place

  ```
  char const* letters = "{}abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_!";
  ...
  for (int i = 0; i < v.size(); ++i)
      std::cout << *(letters + v[i]);
  ```

  – pointer `letters` is stored at `0x5010` (in section `.data`) and the character `'{'` at `0x30b0`

  – variable `i` is stored at `rbp-0x14`

  – comparison `i < v.size()`. takes place in `0x138d-0x13a1`

  This corresponds to printing `is_easy!!}`
- `0x13f6-0x14004,0x1409-0x1417` Printing `"\n"` and `"Your secret number is: "` (addresses `0x3109` and `0x310b` respectively contain the null-ended strings)
- `0x141c-0x1429` Calling `rand()` and printing the result. This is the secret number.
- `0x142e-0x143d` Printing `"\n"` (address `0x3109` contains the null-ended string) and jumping over printing the failure message below.
- `0x143f-0x144d` Printing `"Sorry, Try Again!!!\n"` (address `0x3123` contains the null-ended string)
- `0x143b-0x1442` Function frame dismatle, return

In summary, any KEY Pythagorean triplet `key1,key2,key3` will suffice.

```
 $ ./part_b
============================== Welcome to Part II! ==============================
Enter your roll number:210050018
Enter the keys to unlock this: 5 12 13
is_easy!!}
Your secret number is: 199973188
```

1.3 part_c

```
00000000000015ad <main>:
   15ad: endbr64
   15b1: push rbp
   15b2: mov rbp,rsp
   15b5: push r13
   15b7: push r12
   15b9: push rbx
   15ba: sub rsp,0x138

   15c1: mov rax,QWORD PTR fs:0x28
   15ca: mov QWORD PTR [rbp-0x28],rax
   15ce: xor eax,eax

   15d0: lea rsi,[rip+0x1b61] # 3138 <std::__detail::_S_invalid_state_id+0x90>
   15d7: lea rdi,[rip+0x3a62] # 5040 <std::cout@@GLIBCXX_3.4>
   15de: call 1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
       ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

   15e3: mov DWORD PTR [rbp-0x140],0x2b
   15ed: mov DWORD PTR [rbp-0x13c],0x37
   15f7: mov DWORD PTR [rbp-0x138],0x15
   1601: mov DWORD PTR [rbp-0x134],0x37
   160b: mov DWORD PTR [rbp-0x130],0x11
   1615: mov DWORD PTR [rbp-0x12c],0x12
   161f: mov DWORD PTR [rbp-0x128],0x4
   1629: mov DWORD PTR [rbp-0x124],0x3e
   1633: mov DWORD PTR [rbp-0x120],0x1e
   163d: mov DWORD PTR [rbp-0x11c],0xd
   1647: mov DWORD PTR [rbp-0x118],0x6
   1651: mov DWORD PTR [rbp-0x114],0x8
   165b: mov DWORD PTR [rbp-0x110],0xd
   1665: mov DWORD PTR [rbp-0x10c],0x4
   166f: mov DWORD PTR [rbp-0x108],0x37
   1679: mov DWORD PTR [rbp-0x104],0x11
   1683: mov DWORD PTR [rbp-0x100],0x8
   168d: mov DWORD PTR [rbp-0xfc],0xd
   1697: mov DWORD PTR [rbp-0xf8],0x6
   16a1: mov DWORD PTR [rbp-0xf4],0x3e
   16ab: lea rax,[rbp-0x140]
   16b2: mov r12,rax
   16b5: mov r13d,0x14
   16bb: mov rcx,r12
   16be: mov rbx,r13
   16c1: mov rax,r12
   16c4: mov rdx,r13
   16c7: mov rax,rdx
   16ca: mov rsi,rcx
   16cd: mov rdx,rax
   16d0: lea rdi,[rip+0x3ba9] # 5280 <v>
   16d7: call 197e <std::vector<int, std::allocator<int> >::operator=(std::initializer_list<int>)>

   16dc: lea rsi,[rip+0x1aa7] # 318a <std::__detail::_S_invalid_state_id+0xe2>
   16e3: lea rdi,[rip+0x3956] # 5040 <std::cout@@GLIBCXX_3.4>
   16ea: call 1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
       ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

   16ef: lea rax,[rbp-0x144]
   16f6: mov rsi,rax
   16f9: lea rdi,[rip+0x3a60] # 5160 <std::cin@@GLIBCXX_3.4>
   1700: call 1210 <std::istream::operator>>(int&)@plt>

   1705: lea rsi,[rip+0x1a9c] # 31a8 <std::__detail::_S_invalid_state_id+0x100>
   170c: lea rdi,[rip+0x392d] # 5040 <std::cout@@GLIBCXX_3.4>
   1713: call 1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
       ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>
```

```
1718: lea rax,[rbp-0xf0]
171f: mov rsi,rax
1722: lea rdi,[rip+0x1a9f] # 31c8 <std::__detail::_S_invalid_state_id+0x120>
1729: mov eax,0x0
172e: call 1280 <__isoc99_scanf@plt>

1733: mov eax,DWORD PTR [rbp-0x144]
1739: add eax,0x8
173c: mov edi,eax
173e: call 1240 <srand@plt>

1743: lea rax,[rbp-0xf0]
174a: mov rdi,rax
174d: call 1409 <part_c(char*)>

1752: mov eax,0x0
1757: mov rbx,QWORD PTR [rbp-0x28]
175b: xor rbx,QWORD PTR fs:0x28
1764: je 176b <main+0x1be>
1766: call 1270 <__stack_chk_fail@plt>

176b: add rsp,0x138
1772: pop rbx
1773: pop r12
1775: pop r13
1777: pop rbp
1778: ret
```

- `0x15ad-0x15ba` Function frame setup, allocating `0x138` bytes for local variables
- `0x15c1-0x15ce` (and `0x1752-0x1766`) stack protection check, see this answer
- `0x15d0-0x15de` Printing `"============================== Welcome to Part III! ==============================\n"`
- `0x15e3-0x16d7` Assign to static uninitialised `std::vector<int> v` using initializer list `{0x2b,0x37,0x15,0x37,0x11,0x12,0x4,0x3e,0x1e,0xd,0x6,0x8,0xd,0x4,0x37,0x11,0x8,0xd,0x6,0x3e}` (*static uninitialised* as `v` is in `.bss` section)
- `0x16dc-0x16ea` Printing `"Enter your roll number:"` (address `0x318a` contains the null-ended string)
- `0x16ef-0x1700` Taking roll number input `roll`, stored at address `rbp-0x144`[2]
- `0x1705-0x1713` Printing `"Enter the key to unlock this: "` (address `0x31a8` contains the null-ended string)
- `0x1718-0x172e` Taking key input `key` using `scanf`, stored at address `rbp-0xf0`[2]; parameters are
  - `rdi` = pointer to null-ended string `"%s"` (at address `0x31c8`)
  - `rsi` = pointer to buffer at `rbp-0xf0` (on stack, so chance of stack overflow attack†)
- `0x1733-0x173e` Calling `srand(roll + 0x8)`
- `0x1743-0x174d` Calling `part_c(key)`
- `0x176b-0x1778` Function frame dismantle, return

[2]These are inside the allocated space for local variables as we push 3 registers (`0x18` bytes) on the stack after setting `rbp` and before subtracting `0x38` from `rsp`, i.e. local variables are stored in `rsp=rbp-0x150` to `rbp-0x18`.

```
0000000000001409 <part_c(char*)>:
   1409: endbr64
   140d: push rbp
   140e: mov rbp,rsp
   1411: push rbx
   1412: sub rsp,0x38

   1416: mov QWORD PTR [rbp-0x38],rdi
   141a: mov rax,QWORD PTR [rbp-0x38]
   141e: mov rdi,rax
   1421: call 11f0 <strlen@plt>
   1426: mov DWORD PTR [rbp-0x24],eax
   1429: cmp DWORD PTR [rbp-0x24],0x6
   142d: jle 143f <part_c(char*)+0x36>
   142f: cmp DWORD PTR [rbp-0x24],0xa
   1433: jg 143f <part_c(char*)+0x36>

   1435: mov eax,DWORD PTR [rbp-0x24]
   1438: and eax,0x1
   143b: test eax,eax
   143d: jne 1457 <part_c(char*)+0x4e>

   143f: lea rsi,[rip+0x1cc1] # 3107 <std::__detail::_S_invalid_state_id+0x5f>
   1446: lea rdi,[rip+0x3bf3] # 5040 <std::cout@@GLIBCXX_3.4>
   144d: call 1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
       ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>
   1452: jmp 15a6 <part_c(char*)+0x19d>

   1457: mov eax,DWORD PTR [rbp-0x24]
   145a: add eax,0x1
   145d: cdqe
   145f: mov rdi,rax
   1462: call 11b0 <operator new[](unsigned long)@plt>
   1467: mov QWORD PTR [rbp-0x20],rax

   146b: mov DWORD PTR [rbp-0x2c],0x0
   1472: mov eax,DWORD PTR [rbp-0x2c]
   1475: cmp eax,DWORD PTR [rbp-0x24]
   1478: jge 14a5 <part_c(char*)+0x9c>
   147a: mov eax,DWORD PTR [rbp-0x24]
   147d: sub eax,0x1
   1480: sub eax,DWORD PTR [rbp-0x2c]
   1483: movsxd rdx,eax
   1486: mov rax,QWORD PTR [rbp-0x38]
   148a: add rax,rdx
   148d: mov edx,DWORD PTR [rbp-0x2c]
   1490: movsxd rcx,edx
   1493: mov rdx,QWORD PTR [rbp-0x20]
   1497: add rdx,rcx
   149a: movzx eax,BYTE PTR [rax]
   149d: mov BYTE PTR [rdx],al
   149f: add DWORD PTR [rbp-0x2c],0x1
   14a3: jmp 1472 <part_c(char*)+0x69>

   14a5: mov eax,DWORD PTR [rbp-0x24]
   14a8: movsxd rdx,eax
   14ab: mov rax,QWORD PTR [rbp-0x20]
   14af: add rax,rdx
   14b2: mov BYTE PTR [rax],0x0

   14b5: mov edi,0x15
   14ba: call 11b0 <operator new[](unsigned long)@plt>
   14bf: mov QWORD PTR [rbp-0x18],rax

   14c3: mov rdx,QWORD PTR [rbp-0x20]
   14c7: mov rax,QWORD PTR [rbp-0x38]
   14cb: mov rsi,rdx
```

```
14ce: mov rdi,rax
14d1: call 12b0 <strcmp@plt>
14d6: test eax,eax
14d8: jne 1580 <part_c(char*)+0x177>

14de: mov DWORD PTR [rbp-0x28],0x0
14e5: lea rdi,[rip+0x3d94] # 5280 <v>
14ec: call 1932 <std::vector<int, std::allocator<int> >::size() const>
14f1: cmp DWORD PTR [rbp-0x28],eax
14f4: setl al
14f7: test al,al
14f9: je 1537 <part_c(char*)+0x12e>
14fb: mov rbx,QWORD PTR [rip+0x3b0e] # 5010 <letters>
1502: mov eax,DWORD PTR [rbp-0x28]
1505: cdqe
1507: mov rsi,rax
150a: lea rdi,[rip+0x3d6f] # 5280 <v>
1511: call 195a <std::vector<int, std::allocator<int> >::operator[](unsigned long)>
1516: mov eax,DWORD PTR [rax]
1518: cdqe
151a: add rax,rbx
151d: movzx eax,BYTE PTR [rax]
1520: movsx eax,al
1523: mov esi,eax
1525: lea rdi,[rip+0x3b14] # 5040 <std::cout@@GLIBCXX_3.4>
152c: call 1290 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
    ↪ basic_ostream<char, std::char_traits<char> >&, char)@plt>
1531: add DWORD PTR [rbp-0x28],0x1
1535: jmp 14e5 <part_c(char*)+0xdc>

1537: lea rsi,[rip+0x1bde] # 311c <std::__detail::_S_invalid_state_id+0x74>
153e: lea rdi,[rip+0x3afb] # 5040 <std::cout@@GLIBCXX_3.4>
1545: call 1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
    ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

154a: lea rsi,[rip+0x1bcd] # 311e <std::__detail::_S_invalid_state_id+0x76>
1551: lea rdi,[rip+0x3ae8] # 5040 <std::cout@@GLIBCXX_3.4>
1558: call 1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
    ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

155d: mov rbx,rax
1560: call 11c0 <rand@plt>
1565: mov esi,eax
1567: mov rdi,rbx
156a: call 1300 <std::ostream::operator<<(int)@plt>

156f: lea rsi,[rip+0x1ba6] # 311c <std::__detail::_S_invalid_state_id+0x74>
1576: mov rdi,rax
1579: call 1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
    ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>
157e: jmp 1593 <part_c(char*)+0x18a>

1580: lea rsi,[rip+0x1b80] # 3107 <std::__detail::_S_invalid_state_id+0x5f>
1587: lea rdi,[rip+0x3ab2] # 5040 <std::cout@@GLIBCXX_3.4>
158e: call 1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::
    ↪ basic_ostream<char, std::char_traits<char> >&, char const*)@plt>

1593: cmp QWORD PTR [rbp-0x20],0x0
1598: je 15a6 <part_c(char*)+0x19d>
159a: mov rax,QWORD PTR [rbp-0x20]
159e: mov rdi,rax
15a1: call 12a0 <operator delete[](void*)@plt>

15a6: add rsp,0x38
15aa: pop rbx
15ab: pop rbp
15ac: ret
```

- `0x1409-0x1412` Function frame setup, allocating `0x38` bytes for local variables
- `0x1416-0x1433` Storing argument (here `key`) to local variable at `rbp-0x38` and using function `strlen`, obtaining length l of `key` and storing it to local variable at `rbp-0x24`; KEY jumping to `0x143f` if $l \leq 6$ or $l > 10$
- `0x1435-0x143d` Checking if KEY l is odd (l stored at `rbp-0x24`) and if so jumping to `0x1457`
- `0x143f-0x1452` Printing `"Sorry, Try Again!!!\n"` (address `0x3107` contains the null-ended string) and jumping to `0x15a6` (end of function)
- `0x1457-0x1467` Allocating on heap char array of size $l + 1$ and storing the returned `char*` to local variable at `rbp-0x20`
- `0x146b-0x14b2` Here the following takes place

  ```
  int i = 0
  for (i = 0; i < l; ++i)
      *(char*)(rev + i) = *(char*)(key + (l - 1 + i));
  *(char*)(rev + i) = 0;
  ```

  - pointer `rev` is stored at `rbp-0x20`
  - pointer `key` is stored at `rbp-0x38`
  - variable `i` is stored at `rbp-0x2c`
  - l is stored at `rbp-0x24`

  So `rev` ends up with the key string but reversed.
- `0x14b5-0x14bf` Useless allocation of 21 bytes, never used
- `0x14c3-0x14d8` `strcmp` is used to check if KEY `key` is identical to its reverse and if not so, jumps to `0x1580`
- `14de-0x1535` Here in a manner similar to previous two parts, using `v` and `letters` the string `R3v3rse_Engine3ring_` is printed
- `0x1537-0x1545, 0x154a-0x1558` Printing `"\n"` and `"Your secret number is: "` (addresses `0x311c` and `0x311e` respectively contain the null-ended strings)
- `0x155d-0x156a` Calling `rand()` and printing the result. This is the secret number.
- `0x156f-0x157e` Printing `"\n"` (address `0x311c` contains the null-ended string) and jumping over printing the failure message below.
- `0x1580-0x158e` Printing `"Sorry, Try Again!!!\n"` (address `0x3107` contains the null-ended string)
- `0x1593-0x15a1` Deleting `rev` (pointer stored at `rbp-0x20`) if not null
- `0x15a6-0x15ac` Function frame dismatle, return

In summary, any KEY palindrome of length 7 or 9 will suffice.

```
 $ ./part_c
============================= Welcome to Part III! =============================
Enter your roll number:210050018
Enter the keys to unlock this: hellolleh
R3v3rse_Engine3ring_
Your secret number is: 450171620
```

Combined flag is

`CS230{R3v3rse_Engine3ring_is_easy!!}`

×

2 Q2

I implemented the following algorithm[3] (see comments in code for correspondence)

```
int modInverse(int a, int m)
{
    int m0 = m;
    int y = 0, x = 1;

    if (m == 1)
        return 0;

    while (a > 1) {
        // q is quotient
        int q = a / m;
        int t = m;

        // m is remainder now, process same as
        // Euclid's algo
        m = a % m, a = t;
        t = y;

        // Update y and x
        y = x - q * y;
        x = t;
    }

    // make x positive
    if (x < 0)
        x += m0;

    return x;
}
```

[3]taken from here

3 Q3

3.1 Inplace merge

We are given $0 \leq a_i \leq 10000 < 65536 = 2^{16}$, so upper 16 bits are never occupied for any element of the array. Thus we can use this space as a temporary array for merging instead of using a separate array.[4]

```
// Function to merge the two sorted halves arr[l:m+1] and arr[m+1:r+1] of array arr[l:r+1]
void merge(int arr[], int l, int m, int r)
{
    if (l > m || m > r)
        return;

    // Populate upper 16 bits at each location
    for (int i = l; i <= r; i++)
        arr[i] |= (arr[i] << 16);

    // Merge into arr[l:r+1]
    int i = l, j = m + 1, k = l;
    while (i <= m && j <= r) {
        int s = ((arr[k] >> 16) << 16;
        if (arr[i] <= arr[j]) {
            arr[k] = s | (arr[i] >> 16);
            i++;
        }
        else {
            arr[k] = s | (arr[j] >> 16);
            j++;
        }
        k++;
    }

    // Copy the remaining elements of arr[l:m+1], if there are any
    while (i <= m) {
        arr[k] = (((arr[k] >> 16) << 16) | (arr[i] >> 16);
        i++;
        k++;
    }

    // Copy the remaining elements of arr[m+1:r+1], if there are any
    while (j <= r) {
        arr[k] = (((arr[k] >> 16) << 16) | (arr[j] >> 16);
        j++;
        k++;
    }

    // Clear upper 16 bits at each location
    for (int i = l; i <= r; i++)
        arr[i] &= 0xffff;
}
```

3.2 Iterative mergesort

I implemented the following algorithm.[5]

```
void mergeSort(int arr[], int n)
{
   for (int curr_size = 1; curr_size <= n - 1; curr_size = 2 * curr_size) {
      for (int left_start = 0; left_start < n - 1; left_start += 2 * curr_size) {
         int mid = min(left_start + curr_size - 1, n - 1);
         int right_end = min(left_start + 2 * curr_size - 1, n - 1);
         merge(arr, left_start, mid, right_end);
      }
   }
}
```

[4]Merging algorithm taken from here
[5]Taken from here

4 Q4

4.1 Explanation

```
        shl r8, 3 ; r8 = c1 * 8
        shl r9, 3 ; r9 = c2 * 8

        mov rax, rsi
        mul r8
        mov rbx, rax ; rbx = r1 * c1 * 8

        mov rax, r8
        mul r9
        shr rax, 3
        mov r14, rax ; r14 = c1 * c2 * 8

        mov rsi, rdi
        add rsi, rbx ; rsi = a1 + r1 * c1 * 8

        mov r11, rcx
        add r11, r14 ; r11 = a2 + c1 * c2 * 8

.loop_i:
        mov r12, r10
        add r12, r9
.loop_k:
.loop_j:
        mov rax, [rdi]
        imul qword [rcx]
        add [r10], rax

        add rcx, 8
        add r10, 8
        cmp r10, r12
        jne .loop_j

        add rdi, 8
; add rcx, r9
; sub rcx, r9
        sub r10, r9
        cmp rcx, r11
        jne .loop_k

; add rdi, r8
; sub rdi, r8
        sub rcx, r14
        add r10, r9
        cmp rdi, rsi
        jne .loop_i
```

```
long int* p1 = mat1
long int* p2 = mat2
long int* p3 = mat3
// i loop
do {
   long int* lt = p3 + c2;
   // k loop
   do {
      // j loop
      do {
         (*p3) += (*p1) * (*p2)
         p2 += 1;
         p3 += 1;
      } while (p3 != lt);
      p1 += 1;
      p3 -= c2;
   } while (p2 != mat2 + c1 * c2);
   p2 -= c1 * c2;
   p3 += c2;
} while (p1 != mat1 + r1 * c1);
```

Above is the code for `ikj` variant translated into C. `i`,`j`,`k` aren't referred to explicitly; instead the pointers are incremented/decremented by appropriate amounts according to the variant.
For instance, `p1 = mat1[c1 * i + k]`, so `p1` is incremented inside the `k` loop, decreased by `c1` in the `i` loop to reset `k` and increased by `c1` in the `i` loop to increment `i`.

$$\boxed{f_{\text{TSC}} = 2194.916 \text{ MHz}}$$

4.2 Data

Table 1: ijk

N	c
128	4719387
256	103772898
512	839039615
1024	6347785557
2048	155459581350
4096	5030522522392

Table 2: ikj

N	c
128	4127663
256	32209088
512	247895461
1024	1711266090
2048	17508752902
4096	202930857993

Table 3: jik

N	c
128	5289805
256	121573265
512	899722891
1024	7169856681
2048	183341116593
4096	6135726285010

Table 4: jki

N	c
128	21216509
256	240579337
512	3267800846
1024	54454585016
2048	845004962924
4096	14229354641867

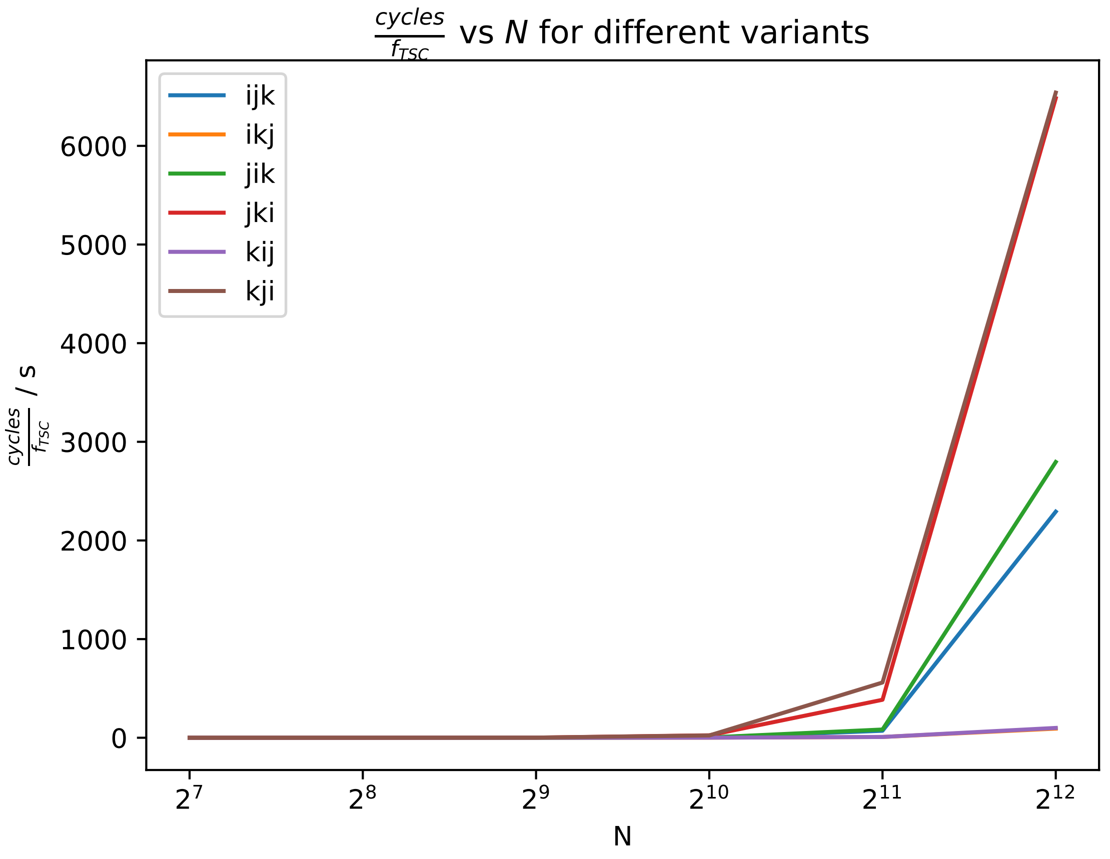
Table 5: kij

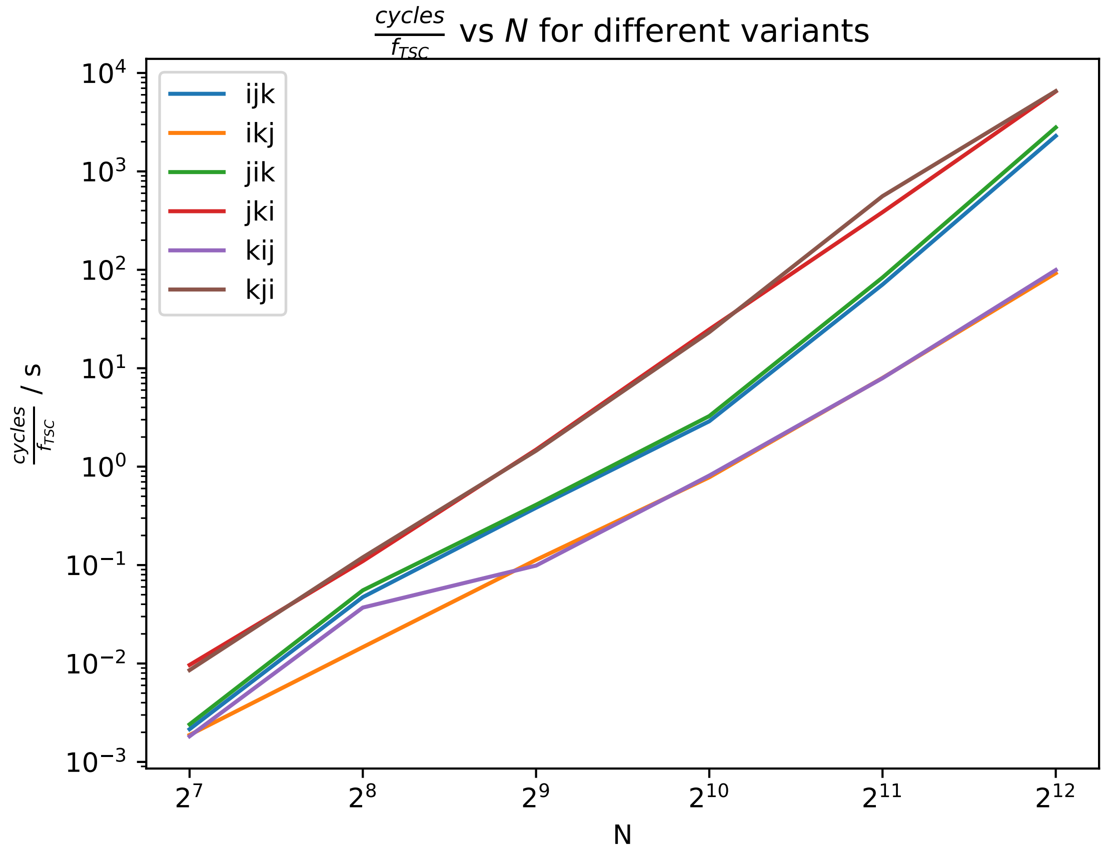
N	c
128	3999887
256	81231316
512	216912652
1024	1782564067
2048	17396323011
4096	218494463644

Table 6: kji

N	c
128	18799143
256	263118667
512	3183815575
1024	510.5901909
2048	1229481765563
4096	14354719246305

Table 7: Cycle data for variants


$\frac{cycles}{f_{TSC}}$ vs N for different variants
ijk
ikj
jik
jki
kij
kji
$\frac{cycles}{f_{TSC}}$ / s
0
1000
2000
3000
4000
5000
6000
2^7
2^8
2^9
2^{10}
2^{11}
2^{12}
N


$\frac{cycles}{f_{TSC}}$ vs N for different variants
ijk
ikj
jik
jki
kij
kji
$\frac{cycles}{f_{TSC}}$ / s
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
2^7
2^8
2^9
2^{10}
2^{11}
2^{12}
N