Exercises

Debugging

Use `journalctl` on Linux or `log show` on macOS to get the super user accesses and commands in the last day. If there aren't any you can execute some harmless commands such as `sudo ls` and check again.

```
~$ man journalctl
# Show superuser accesses and commands in the last day
~$ journalctl -r --since yesterday --output=short | grep sudo
May 06 14:10:05 ramzel-Inspiron-14-3467 sudo[27380]:   ramzel : 
TTY=pts/0 ; PWD=/home/ramzel ; USER=root ; COMMAND=/usr/bin/apt 
autoremove
May 06 14:09:55 ramzel-Inspiron-14-3467 sudo[27376]:   ramzel : 
TTY=pts/0 ; PWD=/home/ramzel ; USER=root ; COMMAND=/usr/bin/apt clean
May 06 14:09:23 ramzel-Inspiron-14-3467 sudo[27354]:   ramzel : 
TTY=pts/0 ; PWD=/home/ramzel ; USER=root ; COMMAND=/usr/bin/lshw -c 
memory
May 06 14:06:09 ramzel-Inspiron-14-3467 sudo[27200]:   ramzel : 
TTY=pts/0 ; PWD=/home/ramzel ; USER=root ; COMMAND=/usr/bin/vim 
/etc/samba/smb.conf
May 06 13:55:46 ramzel-Inspiron-14-3467 sudo[16514]:   ramzel : 
TTY=pts/0 ; PWD=/home/ramzel ; USER=root ; COMMAND=/usr/bin/apt 
update
```

I mostly used sudo to do apt update and to install some applications in the past day.

Install `shellcheck` and try checking the following script. What is wrong with the code? Fix it. Install a linter plugin in your editor so you can get your warnings automatically.

```
#!/bin/sh
## Example: a typical script with several problems
for f in $(ls *.m3u)
# Iterating over ls output is fragile. Use globs.shellcheck(SC2045)

do
  grep -qi hq.*mp3 $f \
  # Quote the grep pattern so the shell won't interpret
it.shellcheck(SC2062)
  # Double quote to prevent globbing and word splitting.shellcheck(SC2086)


    && echo -e 'Playlist $f contains a HQ file in mp3 format'
    # Expressions don't expand in single quotes, use double quotes for
that.shellcheck(SC2016)
    # In POSIX sh, echo flags are undefined.shellcheck(SC3037)
done
```

I used the ShellCheck extension for VSCode and it's a really awesome tool. While coding, it immediately flags programming errors and other problems with the code. ShellCheck tells you what is wrong with it and how to fix it.

Here's the working script:

```
#!/bin/sh
## Fixed the script with several problems
for f in *.m3u
do
 [ -e "$f" ] || break
 grep -qi "hq.*mp3" "$f" \
   && echo "Playlist $f contains a HQ file in mp3 format"
done
```

I ran it through the terminal after creating some .m3u files.

```
$ chmod +x problem-script.sh
$ ./problem-script.sh
Playlist playlist-b.m3u contains a HQ file in mp3 format
Playlist playlist.m3u contains a HQ file in mp3 format
```

Profiling

Here are some sorting algorithm implementations. Use `cProfile` and `line_profiler` to compare the runtime of insertion sort and quicksort. What is the bottleneck of each algorithm? Use then `memory_profiler` to check the memory consumption, why is insertion sort better? Check now the inplace version of quicksort. Challenge: Use `perf` to look at the cycle counts and cache hits and misses of each algorithm.

```
# Using cProfile to profile the runtime of the sorting functions
$ python -m cProfile -s tottime sorts.py
         398688 function calls (332330 primitive calls) in 0.243
seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
    78438    0.065    0.000    0.070    0.000
random.py:177(randrange)
# quicksort function took 39 milliseconds
34064/1000    0.039    0.000    0.041    0.000 sorts.py:23(quicksort)
# quicksort_inplace ran for 27 milliseconds
34294/1000    0.027    0.000    0.031    0.000
sorts.py:32(quicksort_inplace)
78438    0.021    0.000    0.091    0.000 random.py:240(randint)
# insertionsort took 21 milliseconds
     1000    0.021    0.000    0.021    0.000
sorts.py:11(insertionsort)
...
```

cProfile showed that the quick sort function took about 1.8 times longer to run than the insertion sort.

Taking a look at the runtime of the quicksort_inplace function, it is a bit faster than quick sort but insertion sort still proved to be better.

Using line_profiler:

```
~$ pip install line_profiler
```

Add decorator above functions to profile

insertionsort(array)

```
@profile
def insertionsort(array):

   for i in range(len(array)):
      ...
```

quicksort(array)

```
@profile
def quicksort(array):
   if len(array) <= 1:
       return array
   ...
```

Running line_profiler

```
# Profile functions quick sort and insertion sort
$ kernprof -l -v sorts.py
Wrote profile results to sorts.py.lprof
Timer unit: 1e-06 s

# Runtime of insertion sort
Total time: 0.219251 s
File: sorts.py
Function: insertionsort at line 11

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    11                                           @profile
    12                                           def 
insertionsort(array):
    13
    14     26103       6922.0      0.3      3.1      for i in
```

```
range(len(array)):
    15     25103       6599.0      0.3      3.0          j = i-1
    16     25103       6893.0      0.3      3.1          v = array[i]
    17    228923      73297.0      0.3     33.3          while j >= 0
and v < array[j]:
    18    203820      62416.0      0.3     28.4
array[j+1] = array[j]
    19    203820      56112.0      0.3     25.5              j -= 1
    20     25103       7506.0      0.3      3.4          array[j+1] =
v
    21      1000        232.0      0.2      0.1      return array

# Runtime of quick sort
Total time: 0.099628 s
File: sorts.py
Function: quicksort at line 23

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    24                                           @profile
    25                                           def
quicksort(array):
    26     33716      15667.0      0.5     16.2      if len(array) <=
1:
    27     17358       6727.0      0.4      7.0          return array
    28     16358       6797.0      0.4      7.0      pivot = array[0]
    29     16358      26267.0      1.6     27.2      left = [i for i
in array[1:] if i < pivot]
    30     16358      25663.0      1.6     26.5      right = [i for i
in array[1:] if i >= pivot]
    31     16358      15602.0      1.0     16.1      return
quicksort(left) + [pivot] + quicksort(right)
```

Memory Profile

Repeated memory profiling of functions quick sort and insertion sort consistently showed that insertion sort used less memory than insertion sort. However, the difference in memory usage isn't that significant at a thousand iterations.

Quick sort	*39.44 MiB*	*39.48 MiB*	*39.41 MiB*
Insertion sort	*39.07 MiB*	*39.285 MiB*	*39.289 MiB*

```
~$ pip install -U memory_profiler

# Memory profile of quicksort
$ python3.6 -m memory_profiler sorts.py
Filename: sorts.py

Line #    Mem usage    Increment  Occurences   Line Contents
============================================================
    24   39.414 MiB   39.273 MiB       34758   @profile
    25                                         def quicksort(array):
    26   39.414 MiB    0.141 MiB       34758       if len(array) <=
1:
    27   39.414 MiB    0.000 MiB       17879           return array
    28   39.414 MiB    0.000 MiB       16879       pivot = array[0]
    29   39.414 MiB    0.000 MiB      162711       left = [i for i in
array[1:] if i < pivot]
    30   39.414 MiB    0.000 MiB      162711       right = [i for i
in array[1:] if i >= pivot]
    31   39.414 MiB    0.000 MiB       16879       return
quicksort(left) + [pivot] + quicksort(right)

# Memory profile of insertionsort
$ python3.6 -m memory_profiler sorts.py
Filename: sorts.py

Line #    Mem usage    Increment  Occurences   Line Contents
============================================================
    12   39.070 MiB   39.070 MiB        1000   @profile
    13                                         def
insertionsort(array):
    14
    15   39.070 MiB    0.000 MiB       27026       for i in
range(len(array)):
    16   39.070 MiB    0.000 MiB       26026           j = i-1
    17   39.070 MiB    0.000 MiB       26026           v = array[i]
    18   39.070 MiB    0.000 MiB      243316           while j >= 0
and v < array[j]:
    19   39.070 MiB    0.000 MiB      217290               array[j+1]
= array[j]
    20   39.070 MiB    0.000 MiB      217290               j -= 1
    21    39.070 MiB     0.000 MiB       26026           array[j+1] = v
    22    39.070 MiB     0.000 MiB        1000       return array
```

A common issue is that a port you want to listen on is already taken by another process. Let’s learn how to discover that process pid. First execute `python -m http.server 4444` to start a minimal web server listening on port `4444`. On a separate terminal run `lsof | grep LISTEN` to print all listening processes and ports. Find that process pid and terminate it by running `kill <PID>`.

```
# Start a web server
$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
# Find its process id
$ sudo lsof | grep ":4444 .LISTEN"
python3    3934                 ramzel    3u      IPv4
1150791       0t0        TCP *:4444 (LISTEN)
# Terminate the process
~$ kill 3934
```

```
$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
Terminated
```