

系统开发工具基础实验报告

23090031040 王璐鑫

2025 年 9 月 16 日

1 练习内容

1. 使用 Linux 上的或 macOS 上的命令来获取最近一天中超级用户的登录信息及其所执行的指令
2. 这里有一些排序算法的实现。请使用 cProfile 来比较插入排序和快速排序的性能
3. 这里有一些排序算法的实现。请使用 line profiler 来比较插入排序和快速排序的性能
4. 使用 memoryprofiler 来检查内存消耗，为什么插入排序更好一些？然后再看看原地排序版本的快排
5. 将代码拷贝到文件中使其变为一个可执行的程序。首先安装 pycallgraph 和 graphviz(如果您能够执行 dot, 则说明已经安装了 GraphViz.)。并使用 `pycallgraph graphviz - ./fib.py` 来执行代码并查看 `pycallgraph.png` 这个文件
6. 让我们通过进程的 PID 查找相应的进程。首先执行启动一个最简单的 Web 服务器来监听端口。在另外一个终端中，执行打印出所有监听端口的进程及相应的端口
7. 限制进程资源也是一个非常有用的技术。执行 `stress -c 3` 并使用 `htop` 对 CPU 消耗进行可视化。现在，执行 `taskset -cpu-list 0,2 stress -c 3` 并可视化。stress 占用了 3 个 CPU 吗？为什么没有？

8. curl ipinfo.io 命令或执行 HTTP 请求并获取关于您 IP 的信息。打开 Wireshark 并抓取 curl 发起的请求和收到的回复报文。
9. 元编程构建系统
10. 指定版本要求的方法很多，让我们学习一下 Rust 的构建系统的依赖管理。大多数的包管理仓库都支持类似的语法。对于每种语法 (尖号、波浪号、通配符、比较、乘积)，构建一种场景使其具有实际意义
11. 初识 PyTorch：张量
12. 初识 PyTorch：Numpy 的操作
13. PyTorch 自动微分：张量的自动微分
14. PyTorch 自动微分：梯度
15. 神经网络：定义网络
16. 神经网络：打印网络参数、打印某一层参数的形状
17. 神经网络：随机输入一个向量，查看前向传播输出
18. 神经网络：将梯度初始化、随机一个梯度进行反向传播
19. 神经网络：损失函数，随机一个真值，并用随机的输入计算损失
20. 神经网络：计算 19 中 loss 的反向传播

2 结果展示

练习 1:

```
ouc@islouc-vm:~/Desktop$ journalctl | grep sudo
9月 25 14:41:06 islouc-vm sudo[2085]: pam_unix(sudo:auth): authentication failure; logname= uid=1000 euid=0 tty=/dev/pts/0 ruser=ouc rhost= user=ouc
9月 25 14:41:10 islouc-vm sudo[2085]:      ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt in
stall open-vm-tools
9月 25 14:41:10 islouc-vm sudo[2085]: pam_unix(sudo:session): session opened for user root by (uid=0)
9月 25 14:41:10 islouc-vm sudo[2085]: pam_unix(sudo:session): session closed for user root
9月 25 14:41:18 islouc-vm sudo[2136]:      ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt in
stall open-vm-tools open-vm-tools-desktop
9月 25 14:41:18 islouc-vm sudo[2136]: pam_unix(sudo:session): session opened for user root by (uid=0)
9月 25 14:41:19 islouc-vm sudo[2136]: pam_unix(sudo:session): session closed for user root
9月 25 14:42:08 islouc-vm sudo[2405]:      ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt in
stall net-tools vim git
9月 25 14:42:08 islouc-vm sudo[2405]: pam_unix(sudo:session): session opened for user root by (uid=0)
9月 25 14:42:27 islouc-vm sudo[2405]: pam_unix(sudo:session): session closed for user root
9月 25 14:45:04 islouc-vm sudo[4950]:      ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ; COMMAND=/usr/bin/apt in
stall ssh
9月 25 14:45:04 islouc-vm sudo[4950]: pam_unix(sudo:session): session opened for user root by (uid=0)
9月 25 14:45:16 islouc-vm sudo[4950]: pam_unix(sudo:session): session closed for user root
9月 25 14:46:40 islouc-vm sudo[6745]: pam_unix(sudo:auth): authentication failure; logname=ouc uid=1000 euid=0 tty=/dev/
pts/1 ruser=ouc rhost= user=ouc
9月 25 14:46:44 islouc-vm sudo[6745]:      ouc : TTY=pts/1 ; PWD=/home/ouc/Downloads ; USER=root ; COMMAND=/usr/bin/dpkg
-i tigervncserver_1.13.1-1ubuntu1_amd64.deb
9月 25 14:46:44 islouc-vm sudo[6745]: pam_unix(sudo:session): session opened for user root by ouc(uid=0)
9月 25 14:46:46 islouc-vm sudo[6745]: pam_unix(sudo:session): session closed for user root
9月 25 14:49:19 islouc-vm sudo[7854]:      ouc : TTY=pts/1 ; PWD=/home/ouc/Downloads ; USER=root ; COMMAND=/usr/bin/apt
install tigervnc
9月 25 14:49:19 islouc-vm sudo[7854]: pam_unix(sudo:session): session opened for user root by ouc(uid=0)
9月 25 14:49:19 islouc-vm sudo[7854]: pam_unix(sudo:session): session closed for user root
9月 25 14:49:23 islouc-vm sudo[7878]:      ouc : TTY=pts/1 ; PWD=/home/ouc/Downloads ; USER=root ; COMMAND=/usr/bin/apt
install tigervncserver
9月 25 14:49:23 islouc-vm sudo[7878]: pam_unix(sudo:session): session opened for user root by ouc(uid=0)
9月 25 14:49:23 islouc-vm sudo[7878]: pam_unix(sudo:session): session closed for user root
9月 25 14:49:45 islouc-vm sudo[7922]:      ouc : TTY=pts/1 ; PWD=/home/ouc/Downloads ; USER=root ; COMMAND=/usr/bin/apt
install tigervnc-viewer
9月 25 14:49:45 islouc-vm sudo[7922]: pam_unix(sudo:session): session opened for user root by ouc(uid=0)
9月 25 14:49:54 islouc-vm sudo[7922]: pam_unix(sudo:session): session closed for user root
```

练习 2:

```
ouc@islouc-vm:~/Desktop$ python3 -m cProfile -s time sorts.py
584276 function calls (519351 primitive calls) in 5.052 seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
77484   1.059    0.000    1.773    0.000 random.py:250(_randbelow_with_getr
andbits)
77484   0.702    0.000    2.475    0.000 random.py:200(randrange)
77484   0.664    0.000    3.139    0.000 random.py:244(randint)
33722/1000 0.583    0.000    0.871    0.001 sorts.py:23(quick sort)
33176/1000 0.446    0.000    0.585    0.001 sorts.py:32(quick sort_inplace)
98136   0.400    0.000    0.400    0.000 {method 'getrandbits' of '_random.
Random' objects}
3000    0.350    0.000    3.367    0.001 sorts.py:6(<listcomp>)
77484   0.314    0.000    0.314    0.000 {method 'bit_length' of 'int' obje
cts}
68863   0.280    0.000    0.280    0.000 {built-in method builtins.len}
16361   0.075    0.000    0.075    0.000 sorts.py:27(<listcomp>)
16361   0.075    0.000    0.075    0.000 sorts.py:28(<listcomp>)
3        0.057    0.019    5.044    1.681 sorts.py:4(test_sorted)
1000    0.022    0.000    0.026    0.000 sorts.py:11(insertionsort)
3000    0.016    0.000    0.016    0.000 {built-in method builtins.sorted}
2        0.001    0.000    0.003    0.002 <frozen importlib._bootstrap_exter
```

练习 3:

```

Function: test_sorted at line 3
=====
Line #      Hits          Time    Per Hit   % Time  Line Contents
=====
   3                                     @profile
   4     def test_sorted(fn, lters=1000):
   5         for i in range(lters):
   6             l = [random.randint(0, 100) for i in range(0, random.randint(0, 50))]
   7             assert fn(l) == sorted(l)
   8             # print(fn.__name__, fn(l))

Total time: 3.58723 s
File: sorts.py
Function: insertionsort at line 10
=====
Line #      Hits          Time    Per Hit   % Time  Line Contents
=====
  10                                     @profile
  11     def insertionsort(array):
  12         for i in range(len(array)):
  13             j = i-1
  14             v = array[i]
  15             while j >= 0 and v < array[j]:
  16                 array[j+1] = array[j]
  17                 j -= 1
  18             array[j+1] = v
  19             return array

Total time: 0.650393 s
File: sorts.py
Function: quicksort at line 22
=====
Line #      Hits          Time    Per Hit   % Time  Line Contents
=====
  22                                     @profile
  23     def quicksort(array):
  24         if len(array) <= 1:
  25             return array
  26         pivot = array[0]
  27         left = [i for i in array[1:] if i < pivot]
  28         right = [i for i in array[1:] if i >= pivot]
  29         return quicksort(left) + [pivot] + quicksort(right)

Total time: 2.86543 s
File: sorts.py
Function: quicksort_inplace at line 31
=====
Line #      Hits          Time    Per Hit   % Time  Line Contents
=====
  31                                     @profile
  32     def quicksort_inplace(array, low=0, high=None):
  33         if len(array) <= 1:
  34             return array
  35         if high is None:
  36             high = len(array)-1
  37         if low >= high:
  38             return array
  39         pivot = array[high]
  40         j = low-1
  41         for i in range(low, high):
  42             if array[i] <= pivot:
  43                 j += 1
  44                 array[i], array[j] = array[j], array[i]
  45             array[high], array[j+1] = array[j+1], array[high]
  46             quicksort_inplace(array, low, j)
  47             quicksort_inplace(array, j+2, high)
  48             return array
  49

```

练习 4:

```
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\MadeLine\Desktop> python -m memory_profiler sorts.py
Filename: sorts.py

Line #   Mem usage   Increment   Occurrences   Line Contents
=====
3       27.047 MiB   27.027 MiB         3   @profile
4       27.047 MiB   0.000 MiB       3003   def test_sorted(fn, iters=1000):
5       27.047 MiB   0.012 MiB      79351       for i in range(iters):
6       27.047 MiB   81140.203 MiB    3000         l = [random.randint(0, 100) for i in range(0, random.randint(0, 50))]
7       27.047 MiB   0.000 MiB         1         assert fn(l) == sorted(l)
8       27.047 MiB   0.000 MiB         1         # print(fn.__name__, fn(l))

Filename: sorts.py

Line #   Mem usage   Increment   Occurrences   Line Contents
=====
10      27.047 MiB  27046.875 MiB       1000   @profile
11      27.047 MiB   0.000 MiB       26354   def insertionsort(array):
12      27.047 MiB   0.000 MiB       25354       for i in range(len(array)):
13      27.047 MiB   0.000 MiB       25354         j = i-1
14      27.047 MiB   0.000 MiB       233037         v = array[i]
15      27.047 MiB   0.000 MiB       209683         while j >= 0 and v < array[j]:
16      27.047 MiB   0.000 MiB       25354           array[j+1] = array[j]
17      27.047 MiB   0.000 MiB       25354           j -= 1
18      27.047 MiB   0.000 MiB       1000         array[j+1] = v
19      27.047 MiB   0.000 MiB         1         return array

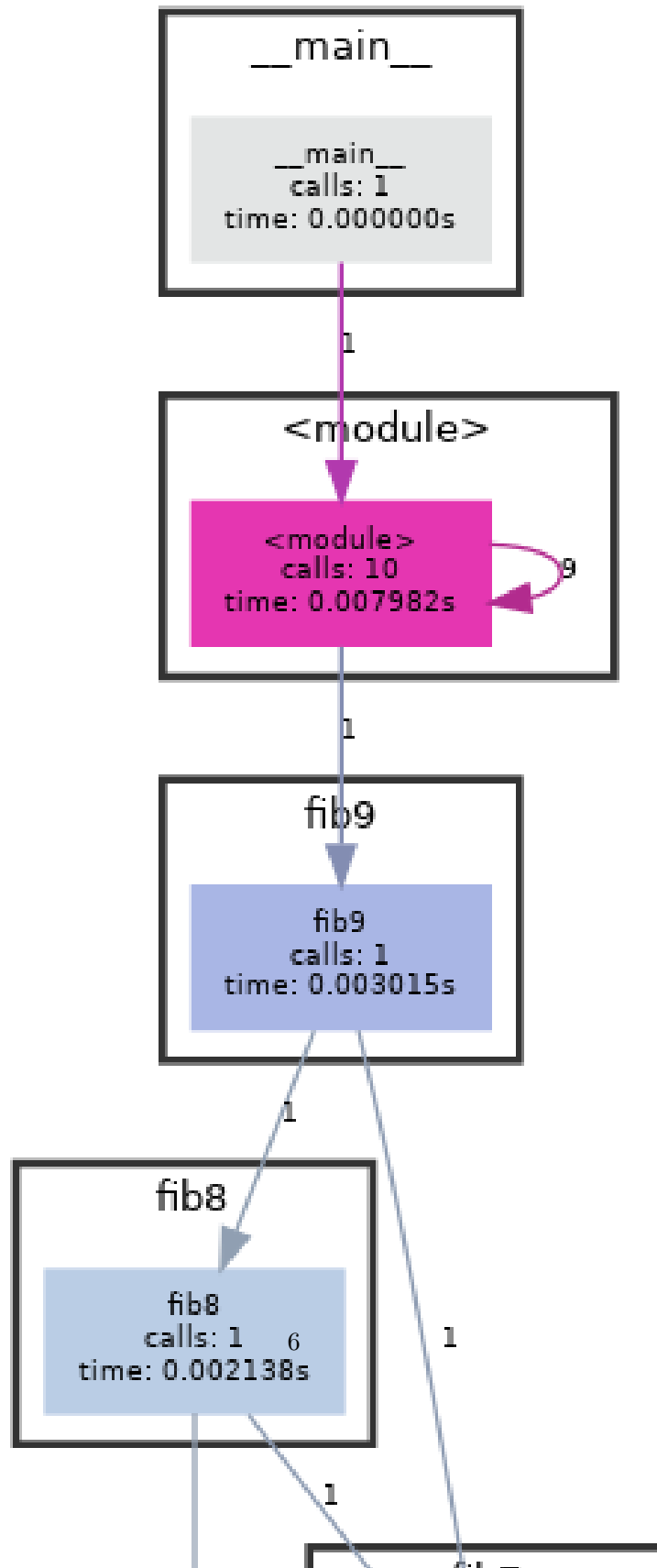
Filename: sorts.py

Line #   Mem usage   Increment   Occurrences   Line Contents
=====
22      27.047 MiB  27046.453 MiB       34318   @profile
23      27.047 MiB   0.000 MiB       17659   def quicksort(array):
24      27.047 MiB   0.000 MiB       16659     if len(array) <= 1:
25      27.047 MiB   0.000 MiB       127090       return array
26      27.047 MiB   0.000 MiB       16659     pivot = array[0]
27      27.047 MiB   0.000 MiB       16659     left = [i for i in array[1:] if i < pivot]
28      27.047 MiB   0.000 MiB       16659     right = [i for i in array[1:] if i >= pivot]
29      27.047 MiB   0.000 MiB         1     return quicksort(left) + [pivot] + quicksort(right)

Filename: sorts.py

Line #   Mem usage   Increment   Occurrences   Line Contents
=====
31      27.047 MiB  27046.875 MiB       34462   @profile
32      27.047 MiB   0.000 MiB       34462   def quicksort_inplace(array, low=0, high=None):
33      27.047 MiB   0.000 MiB         1     if len(array) <= 1:
```

练习 5:



练习 6:

```
ouc@islouc-vm: ~/Desktop
ouc@islouc-vm:~/Desktop$ python -m http.server 4444444
Command 'python' not found, did you mean:
  command 'python3' from deb python3
  command 'python' from deb python-is-python3

ouc@islouc-vm:~/Desktop$ python3 -m http.server 4444444
Serving HTTP on 0.0.0.0 port 11036 (http://0.0.0.0:11036/) ...
Terminated
ouc@islouc-vm:~/Desktop$

ouc@islouc-vm:~/Desktop$ lsdf | grep LISTEN
lsdf: WARNING: can't stat() overlay file system /var/lib/docker/overl
951349cf05de2af229cd21250ce5abf9bbfae61539300f727dc208de/merged
  Output information may be incomplete.
lsdf: WARNING: can't stat() nfs file system /run/docker/netns/27b4ab61
  Output information may be incomplete.
python3 13486                                     ouc 3u IPv4
67 0t0 TCP *:11036 (<LISTEN>)
ouc@islouc-vm:~/Desktop$ KILL 13486
KILL: command not found
ouc@islouc-vm:~/Desktop$ kill 13486
ouc@islouc-vm:~/Desktop$ S
```

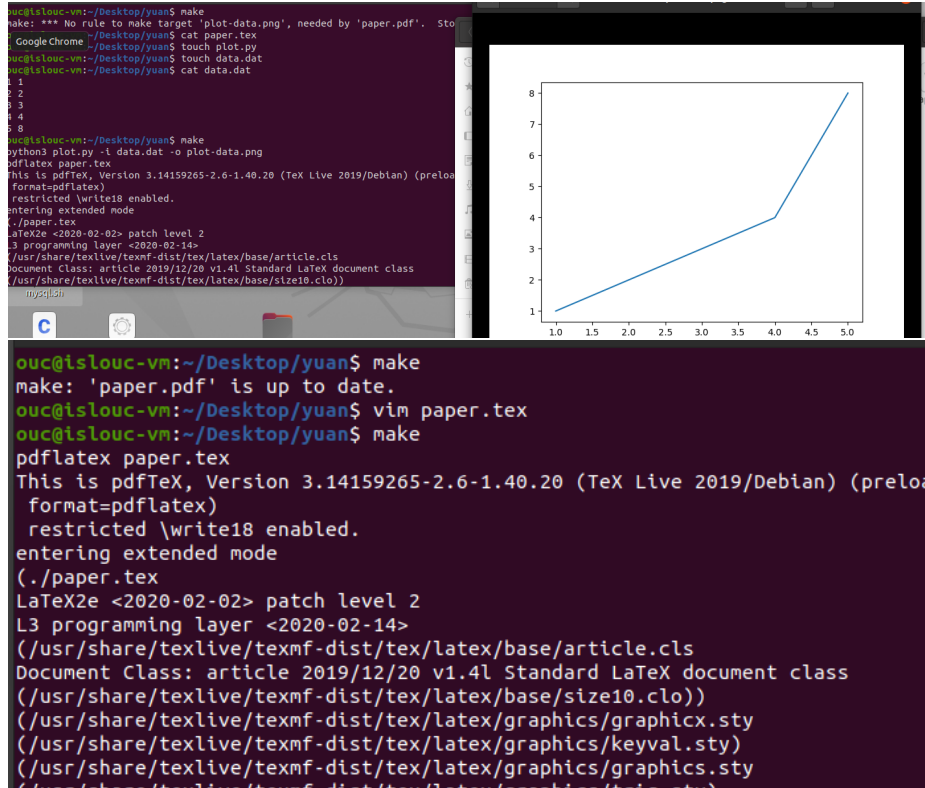
练习 7: 答: 虽然 stress 请求了 3 个 CPU, 但系统只允许它在 2 个 CPU 上运行

[illegible]

练习 8:

[illegible]

练习 9:



练习 10:

版本字符串称为**版本要求**。它指定了在**解析依赖项**时可以选择的版本范围。在本例中，表示版本范围。如果在该范围内，则允许更新。在这种情况下，如果我们运行了，cargo 应该如果它是最新版本，则将我们更新到 version，但不会将我们更新到。"0.1.12" "0.1.12" >=0.1.12, <0.2.0 cargo update
time 0.1.13 0.1.z 0.2.0

版本要求语法

默认要求

默认要求指定能够更新到 SemVer 兼容版本的最低版本。如果版本最左边的非零主要/次要/补丁组件相同，则认为版本是兼容的。这与 [SemVer 不同](#)，SemVer 认为所有 1.0.0 之前的软件包都是不兼容的。

1.2.3 是默认要求的一个示例。

```
1.2.3 := >=1.2.3, <2.0.0
1.2   := >=1.2.0, <2.0.0
1     := >=1.0.0, <2.0.0
0.2.3 := >=0.2.3, <0.3.0
0.2   := >=0.2.0, <0.3.0
0.0.3 := >=0.0.3, <0.0.4
0.0   := >=0.0.0, <0.1.0
0     := >=0.0.0, <1.0.0
```

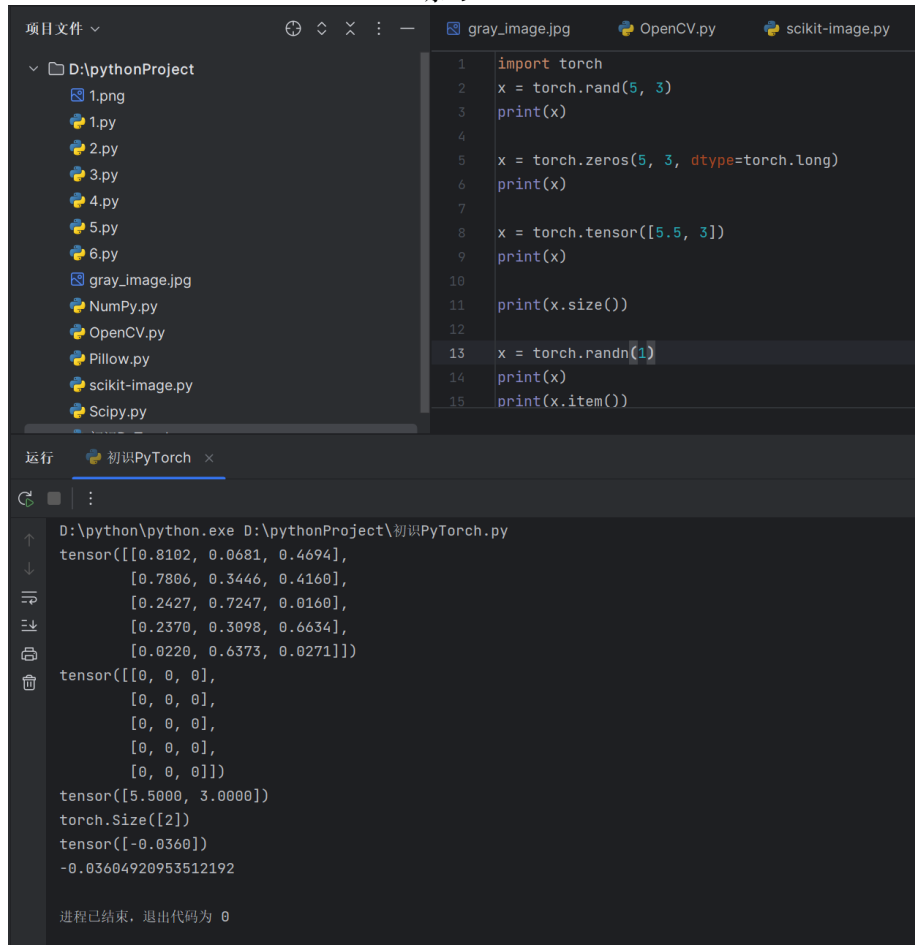
插入符号要求

插入符号要求是默认版本要求策略。此版本策略允许 SemVer 兼容更新。它们被指定为带有前导插入符号 (^) 的版本要求。^

^1.2.3 是插入符号要求的一个示例。

省略插入符号是与使用插入符号要求的简化等效语法。虽然插入符号要求是默认的，但建议使用 尽可能简

练习 11:



The screenshot shows an IDE with a project named 'D:\pythonProject'. The file explorer on the left lists various files including '1.png', '1.py', '2.py', '3.py', '4.py', '5.py', '6.py', 'gray_image.jpg', 'NumPy.py', 'OpenCV.py', 'Pillow.py', 'scikit-image.py', and 'Scipy.py'. The main editor displays a Python script with the following code:

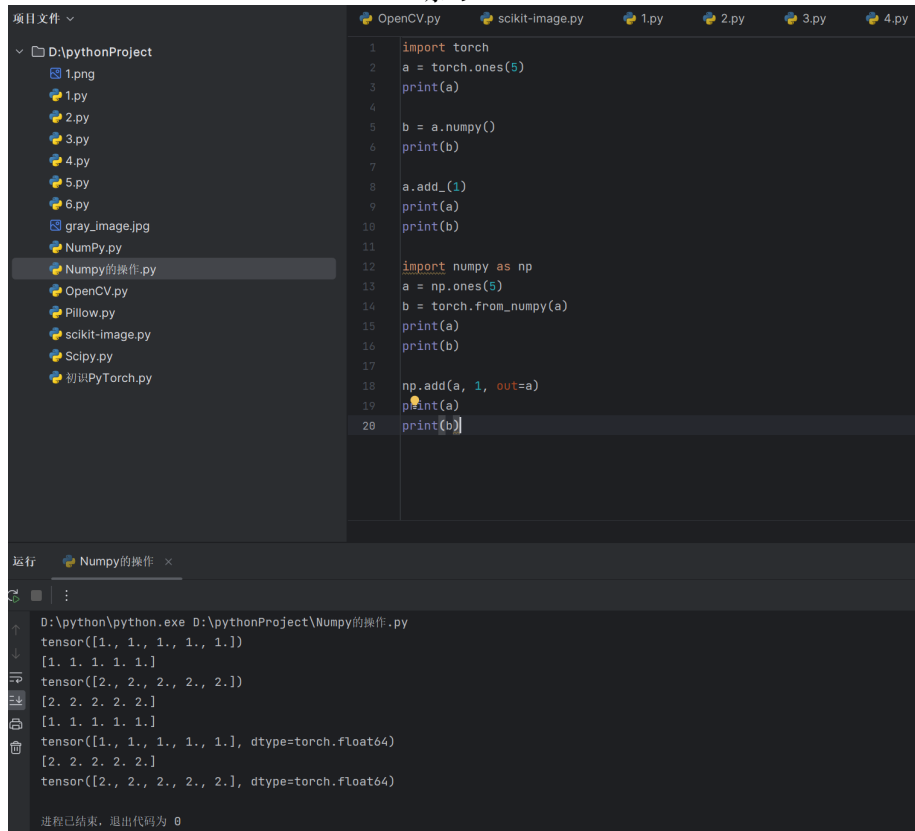
```
1 import torch
2 x = torch.rand(5, 3)
3 print(x)
4
5 x = torch.zeros(5, 3, dtype=torch.long)
6 print(x)
7
8 x = torch.tensor([5.5, 3])
9 print(x)
10
11 print(x.size())
12
13 x = torch.randn(1)
14 print(x)
15 print(x.item())
```

The bottom panel shows the execution output for the script '初识PyTorch.py':

```
D:\python\python.exe D:\pythonProject\初识PyTorch.py
tensor([[0.8102, 0.0681, 0.4694],
        [0.7806, 0.3446, 0.4160],
        [0.2427, 0.7247, 0.0160],
        [0.2370, 0.3098, 0.6634],
        [0.0220, 0.6373, 0.0271]])
tensor([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
tensor([5.5000, 3.0000])
torch.Size([2])
tensor([-0.0360])
-0.03604920953512192

进程已结束，退出代码为 0
```

练习 12:



The screenshot shows a Python IDE interface. The left sidebar displays a file explorer for a project named 'D:\pythonProject'. It contains various files including '1.png', '1.py', '2.py', '3.py', '4.py', '5.py', '6.py', 'gray_image.jpg', 'NumPy.py', 'Numpy的操作.py' (selected), 'OpenCV.py', 'Pillow.py', 'scikit-image.py', 'Scipy.py', and '初识PyTorch.py'. The main editor window shows the code for 'Numpy的操作.py'. The code imports 'torch' and 'numpy as np', creates a 5x5 tensor 'a' with ones, prints it, converts it to a NumPy array 'b', prints 'b', increments 'a' by 1, and prints the updated 'a' and 'b'. The bottom terminal window shows the execution output, displaying the tensor and array representations before and after the increment operation.

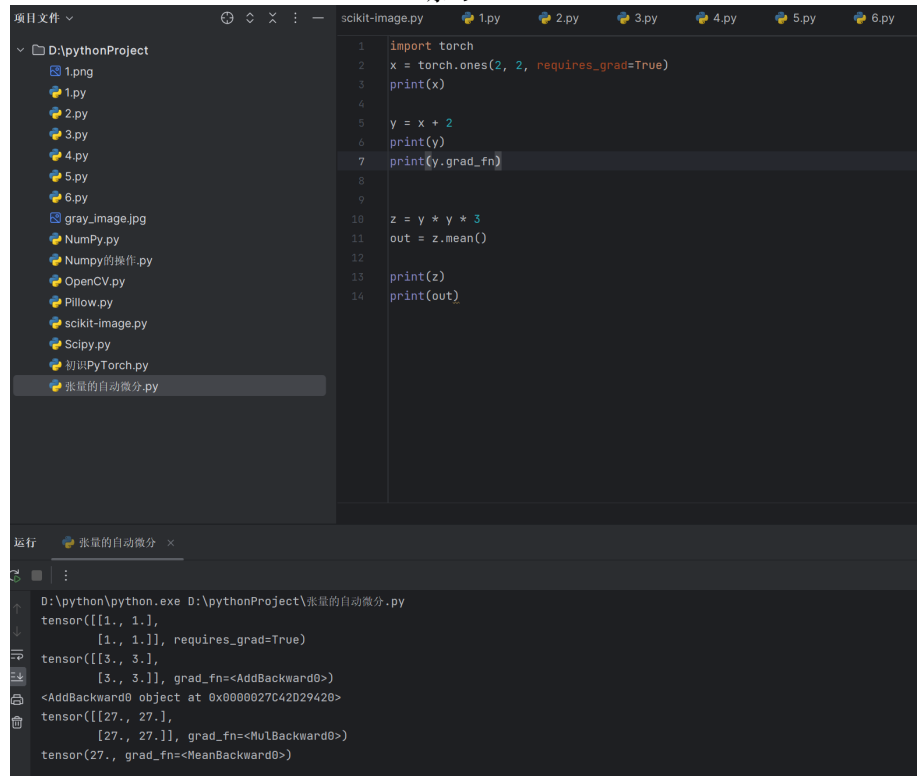
```
1 import torch
2 a = torch.ones(5)
3 print(a)
4
5 b = a.numpy()
6 print(b)
7
8 a.add_(1)
9 print(a)
10 print(b)
11
12 import numpy as np
13 a = np.ones(5)
14 b = torch.from_numpy(a)
15 print(a)
16 print(b)
17
18 np.add(a, 1, out=a)
19 print(a)
20 print(b)
```

运行 Numpy的操作 ×

```
D:\python\python.exe D:\pythonProject\Numpy的操作.py
tensor([1., 1., 1., 1., 1.])
[1. 1. 1. 1. 1.]
tensor([2., 2., 2., 2., 2.])
[2. 2. 2. 2. 2.]
[1. 1. 1. 1. 1.]
tensor([1., 1., 1., 1., 1.], dtype=torch.float64)
[2. 2. 2. 2. 2.]
tensor([2., 2., 2., 2., 2.], dtype=torch.float64)

进程已结束，退出代码为 0
```

练习 13:

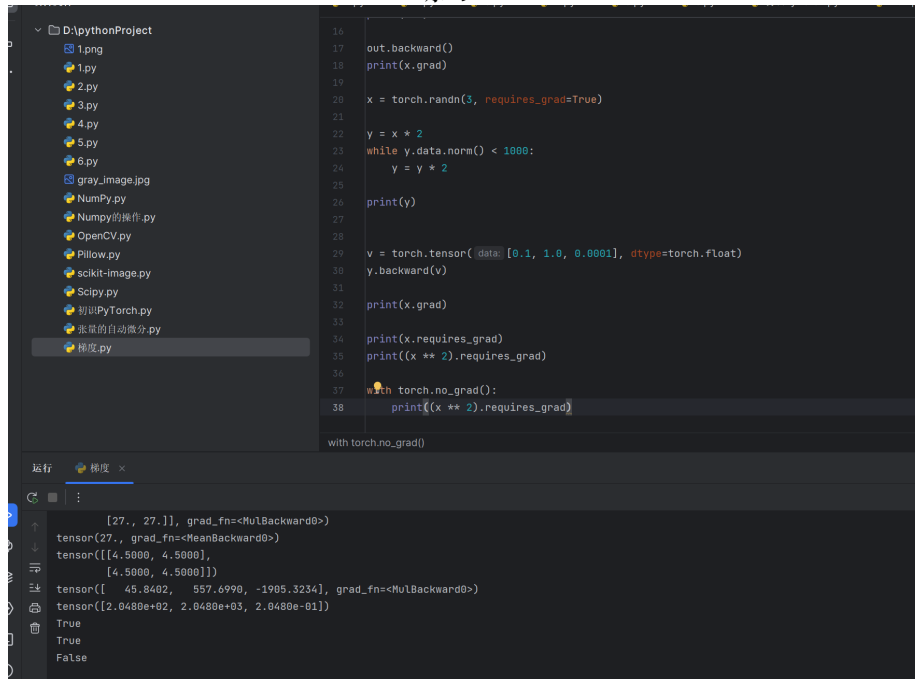


```
1 import torch
2 x = torch.ones(2, 2, requires_grad=True)
3 print(x)
4
5 y = x + 2
6 print(y)
7 print(y.grad_fn)
8
9
10 z = y * y * 3
11 out = z.mean()
12
13 print(z)
14 print(out)
```

运行 张量的自动微分

```
D:\python\python.exe D:\pythonProject\张量的自动微分.py
tensor([[1., 1.],
        [1., 1.]])
tensor([[3., 3.],
        [3., 3.]])
<AddBackward0 object at 0x0000027C4D29420>
tensor([[27., 27.],
        [27., 27.]])
tensor(27., grad_fn=<MeanBackward0>)
```

练习 14:



```

16 out.backward()
17 print(x.grad)
18
19 x = torch.randn(3, requires_grad=True)
20
21 y = x * 2
22 while y.data.norm() < 1000:
23     y = y * 2
24
25 print(y)
26
27 v = torch.tensor([0.1, 1.0, 0.0001], dtype=torch.float)
28 y.backward(v)
29
30 print(x.grad)
31
32 print(x.requires_grad)
33 print((x ** 2).requires_grad)
34
35 with torch.no_grad():
36     print((x ** 2).requires_grad)
37
38 with torch.no_grad():

```

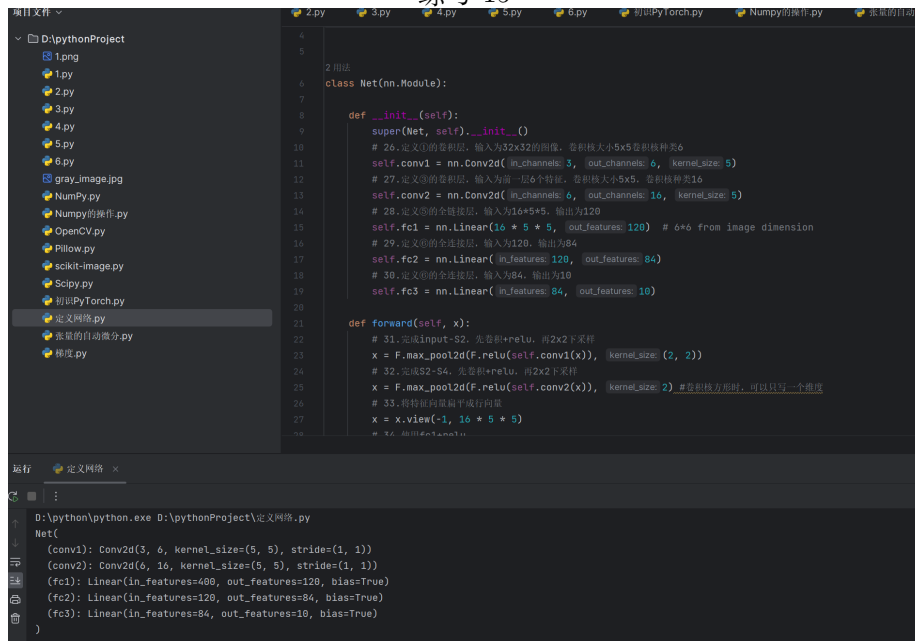
运行 梯度 ×

```

[27., 27.], grad_fn=<MulBackward0>)
tensor([27., 27.], grad_fn=<MulBackward0>)
tensor([4.5000, 4.5000])
[4.5000, 4.5000]]
tensor([ 45.8402, 557.6990, -1905.3234], grad_fn=<MulBackward0>)
tensor([2.0480e+02, 2.0480e+03, 2.0480e-01])
True
True
False

```

练习 15:



```

4
5
6 class Net(nn.Module):
7
8     def __init__(self):
9         super(Net, self).__init__()
10        # 26. 定义1的卷积层，输入为32x32的图像，卷积核大小5x5卷积核种类6
11        self.conv1 = nn.Conv2d(1, channels=3, out_channels=6, kernel_size=5)
12        # 27. 定义2的卷积层，输入为前一层6个特征，卷积核大小5x5，卷积核种类16
13        self.conv2 = nn.Conv2d(6, out_channels=16, kernel_size=5)
14        # 28. 定义3的全连接层，输入为16*5*5，输出为120
15        self.fc1 = nn.Linear(16 * 5 * 5, out_features=120) # 6*6 from image dimension
16        # 29. 定义4的全连接层，输入为120，输出为84
17        self.fc2 = nn.Linear(in_features=120, out_features=84)
18        # 30. 定义5的全连接层，输入为84，输出为10
19        self.fc3 = nn.Linear(in_features=84, out_features=10)
20
21    def forward(self, x):
22        # 31. 完成Input-S2: 先卷积+relu，再2x2下采样
23        x = F.max_pool2d(F.relu(self.conv1(x)), kernel_size=(2, 2))
24        # 32. 完成S2-S4: 先卷积+relu，再2x2下采样
25        x = F.max_pool2d(F.relu(self.conv2(x)), kernel_size=2) # 卷积核为方形时，可以只写一个维度
26        # 33. 将特征向量展平成行向量
27        x = x.view(-1, 16 * 5 * 5)
28        # 34. 应用fc1+relu

```

运行 定义网络 ×

```

D:\python\python.exe D:\pythonProject\定义网络.py
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)

```

练习 16:

```

scikit-image.py
Scipy.py
初识PyTorch.py
定义网络.py
张量的自动微分.py
梯度.py

36
37 net = Net()
38 print(net)
39
40 params = list(net.parameters())
41 # print(params)
42 print(len(params))
43 print(params[0].size())

运行 定义网络 x
:
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
10
torch.Size([6, 3, 5, 5])

```

练习 17:

```

张量的自动微分.py
梯度.py

44
45 input1 = torch.randn(1, 3, 32, 32)
46 out = net(input1)
47 print(out)

运行 定义网络 x
:
(conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
(fc1): Linear(in_features=400, out_features=120, bias=True)
(fc2): Linear(in_features=120, out_features=84, bias=True)
(fc3): Linear(in_features=84, out_features=10, bias=True)
)
10
torch.Size([6, 3, 5, 5])
tensor([[ 0.0049,  0.0633,  0.1846,  0.0730,  0.0493,  0.0895,  0.1843,  0.0295,
          -0.0326, -0.0743]], grad_fn=<AddmmBackward0>)

进程已结束。退出代码为 0

```

练习 18:

```

net.zero_grad()
50 put.backward(torch.randn(1, 10))

运行 定义网络 ×

:
(conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
(fc1): Linear(in_features=400, out_features=120, bias=True)
(fc2): Linear(in_features=120, out_features=84, bias=True)
(fc3): Linear(in_features=84, out_features=10, bias=True)
)
10
torch.Size([6, 3, 5, 5])
tensor([[ 0.0488, -0.0068,  0.0162,  0.0167,  0.0004, -0.0245,  0.0098,  0.0783,
          0.0614,  0.0882]], grad_fn=<AddmmBackward0>)

进程已结束，退出代码为 0

```

练习 19:

```

26 # 32, 32或32-34, 左卷积+relu, 由2x2下采样
27 x = F.max_pool2d(F.relu(conv2(x)), [kernel_size, 2]) # 卷积核为2, 步长为2, 可以只写一个维度
28 x = x.view(-1, 16 * 5 * 5)
29 # 34, 使用fc1+relu
30 x = F.relu(self.fc1(x))
31 # 35, 使用fc2+relu
32 x = F.relu(self.fc2(x))
33 # 36, 使用fc3
34 x = self.fc3(x)
35 return x

36 net = Net()
37 criterion = nn.MSELoss()
38
39 # 准备输入和目标
40 input = torch.randn(1, 3, 32, 32)
41 target = torch.randn(10).view(1, -1)
42
43 # 返回预测值和损失
44 output = net(input)
45 loss = criterion(output, target)
46 print(loss)

```

运行 损失函数 ×

```

D:\python\python.exe D:\pythonProject\损失函数.py
tensor(0.6473, grad_fn=<MseLossBackward0>)

```

练习 20:

```

22 # 36, 使用fc3
23 x = self.fc3(x)
24 return x

25 net = Net()
26 criterion = nn.MSELoss()
27
28 # 准备输入和目标
29 input = torch.randn(1, 3, 32, 32)
30 target = torch.randn(10).view(1, -1)
31
32 # 返回预测值和损失
33 output = net(input)
34 loss = criterion(output, target)
35 print(loss)
36
37 loss.backward()
38
39 print('conv1.bias.grad after backward')
40 print([net.conv1.bias.grad])

```

运行 损失函数 ×

```

D:\python\python.exe D:\pythonProject\损失函数.py
tensor(1.2936, grad_fn=<MseLossBackward0>)
conv1.bias.grad after backward
tensor([-0.0074,  0.0151,  0.0082,  0.0025,  0.0230,  0.0185])

```

3 体会与收获

通过本次实验，我深刻体会到了系统工具与性能分析在软件开发中的重要性，同时也对 PyTorch 深度学习框架有了更深入的理解。

我学会了使用系统命令来监控系统状态和进程管理，如通过 `last` 命令查看用户登录信息，使用 `lsof` 或 `netstat` 查看端口监听情况。通过 `stress` 和 `taskset` 命令，我能看到 CPU 资源的分配，懂得了资源合理分配的重要性。

通过 `cProfile` 和 `lineprofiler` 对排序算法进行性能分析，我能够直观地比较插入排序和快速排序的时间效率差异。

在神经网络部分，我从张量基础操作开始，逐步实现了完整的网络定义、前向传播、反向传播和损失计算。

在实验过程中我遇到最大的问题是我完全没看懂元编程是什么，根据练习题我把它理解为了和 GIT 一样是便于开发的工具，但实际上元编程是一种能够编写其他程序的一种程序，是一种写程序的程序。其次是我本次实验的很多地方由于翻译问题，题目的信息捕获不全面，导致很多地方卡壳了，最后是看的原文自己翻译得到了正确的操作步骤。

github 地址:<https://github.com/HollowWarlock/ouc-xitonggongjukaifa>.git