1.

Group members:	Ziqi Liu	9364-4917
	Enyang Wang	6494-1124

Steps: 1. Enter root folder (project1).

2. using commend: **mix run proj1.exs arg1 arg2**

3. wait for the result (it may take a while)

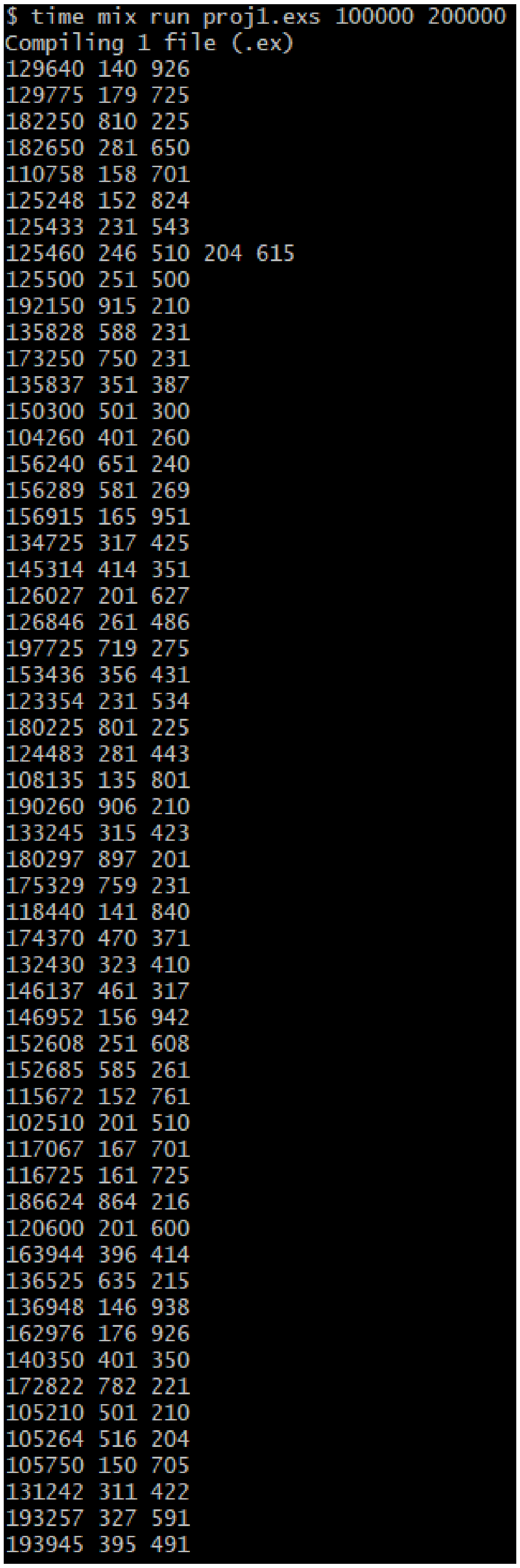
2.

We create 101 workers

3.

The range of work unit is 1000 numbers. We found 1000 numbers is the best choice after trying different ranges.

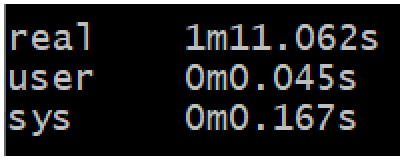
4.



```
$ time mix run proj1.exs 100000 200000
Compiling 1 file (.ex)
129640 140 926
129775 179 725
182250 810 225
182650 281 650
110758 158 701
125248 152 824
125433 231 543
125460 246 510 204 615
125500 251 500
192150 915 210
135828 588 231
173250 750 231
135837 351 387
150300 501 300
104260 401 260
156240 651 240
156289 581 269
156915 165 951
134725 317 425
145314 414 351
126027 201 627
126846 261 486
197725 719 275
153436 356 431
123354 231 534
180225 801 225
124483 281 443
108135 135 801
190260 906 210
133245 315 423
180297 897 201
175329 759 231
118440 141 840
174370 470 371
132430 323 410
146137 461 317
146952 156 942
152608 251 608
152685 585 261
115672 152 761
102510 201 510
117067 167 701
116725 161 725
186624 864 216
120600 201 600
163944 396 414
136525 635 215
136948 146 938
162976 176 926
140350 401 350
172822 782 221
105210 501 210
105264 516 204
105750 150 705
131242 311 422
193257 327 591
193945 395 491
```

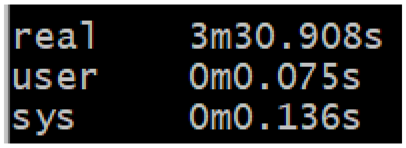
5.

Create 100 workers:


```
real	1m11.062s
user	0m0.045s
sys	0m0.167s
```

The ratio of CPU time to REAL time is (0.045+0.167)/71.062=0.003.

Create only 1 worker:


```
real	3m30.908s
user	0m0.075s
sys	0m0.136s
```

The ratio of CPU time to REAL time is (0.075+0.136)/210.908=0.001

Why our program has shorter CPU time than others?

Because the algorithm we use does not need too much compute. The detail of our algorithm is below.

I know our program takes a longer time than others. But as you can see, we are using parallelism compute since the real time is smaller and the ratio is bigger. Our algorithm is to check each number in the range whether it is vampire number or not. The way we judge a number is: firstly, find all permutations on each digit of this number. Then multiply the front half of numbers with the back half of numbers, and check if it is equal to the original number.

For example, check 1234 is a vampire number or not.

1. 1234 have 24 different permutations on each digit, 1234,1324,1243……
2. Check 12*34 = 1234? 13*24 = 1234?......
3. Check every number in the range

The reason why our program is slow is because we compute duplicate permutations. For example, we do not need to check 1324,1243, etc. anymore after checking 1234 is a vampire number or not. However, we cannot fix it since time limit.

6.

Find all permutations on each digit of a number. For example, all permutations of 1234 are 1234,1324,1243,1342,1423,1432,2134,2143,2341,2314,2413,2431,3124,3142,3241,3214,3412,3

421,4123,4132,4213,4231,4312,4321. The implementation is

```
def of([]) do
    [[]]
end

def of(list) do
    for h <- list, t <- of(list -- [h]), do: [h | t]
end
end
```