HW1

1 Code

[1]:
```
import math
def factor(n):
    rlt = []
    input = n

    for i in range(2, int(math.sqrt(n)+1), 1):
        while n % i == 0:
            rlt.append(i)
            n = n / i
    if n > 2 and n != input:
        rlt.append(int(n))
    return rlt
```

2 Running time analysis

Step 1: The parameter that would impact the execution time of this algorithm is the size of number n, or equivalently the number of bits needed to represent n.

Step 2: The operation that would be executed most often in this algorithm is the condition in the while loop: **n % i == 0**.

Step 3: The number of times the basic operation **n % i == 0** gets executed varies with diffrent n. For example, if n itself is a prime number, then while condition will be checked for $\sqrt(n)$ times. However, if n is simply 2^x for some integer x, then the condition will be checked $log_2(n)$ times.

2.1 Case 1 : Multiplications and division (and additions) take constant time

In this case, the operation n % i takes constant time, denoted as c.

For the worst case, when n is a prime number, the condition **n % i == 0** will be run for each of the $i \in [2, 3, \ldots, \sqrt(n)]$. As the condition is **False** each time, no other opreations will be excuted expect for the final **if n > 2 and n != input** check which takes constant time. So $T(n) = \sum_{i=2}^{int(\sqrt(n)+1)}(c) = c * int(\sqrt(n)) = O(\sqrt(n))$.

2.2 Case 2: Multiplication and division of n-bit numbers take $O(n^2)$ time and additions and subtractions take $O(n)$ time

In this case, assume that n is a x decimal digits number. From math we know a given integer n consists of $\lfloor log_{10}(n) \rfloor + 1$ decimal digits, Thus, $x = \lfloor log_{10}(n) \rfloor + 1$. Since the decimal digits is directly proportional to the bits, we assume n is a x-bit number. Then, the operation n % i takes $O(x^2)$ time.

For the worst case as stated in Case 1, $T(n) = \sum_{i=2}^{int(\sqrt(n)+1)} (log(n)+1)^2 = int(\sqrt(n)) * (log(n)+1)^2 = O(\sqrt(n) * log(n) * log(n))$.

3 Table of $T(n)$ vs. n

Since we would compare the worst case running time $T(n)$ with n, here I would try to expriment on the prime numbers only.

3.1 The first expriment is testing for small primes n.

(Refer:https://primes.utm.edu/lists/small/)

[18]:
```
import time
primes = [7,11,937, 9907 , 90227, 404251, 944191,
          1299689,15485863,104395301,472882027,982451653]

runtime =[]
for num in primes:
    st = time.time()
    factor(num)
    et = time.time()
    runtime.append(et-st)
```

[31]:
```
import numpy as np
import pandas as pd
case1_T = [math.floor(math.sqrt(num)) for num in primes]
case2_T = [math.floor(math.sqrt(num) * (math.floor(math.log(num,10)))**2)
           for num in primes]
Tn_usec = [math.floor(t*1e6) for t in runtime]
data = {'T(n) in usec' : Tn_usec, 'n' : primes,
        'Case 1': case1_T,'Case 2': case2_T}
df = pd.DataFrame(data)
print(df)
```

```
    T(n) in usec          n  Case 1   Case 2
0              7          7       2        0
1              1         11       3        3
2              1        937      30      122
3              6       9907      99      895
4             17      90227     300     4806
```

5	44	404251	635	15895
6	69	944191	971	24292
7	82	1299689	1140	41041
8	307	15485863	3935	192825
9	810	104395301	10217	653913
10	1751	472882027	21745	1391734
11	3066	982451653	31344	2006021

In the table above: - 'T(n) in usec' is the actual running time in usec(microsecond) for a given prime n. - 'n' is given the prime number. - 'Case 1' and 'Case2' represent the "theoretical" running time calculated based on $T(n)$ derived in Question 2 under two cases.

To better see if the actual runnint time $T(n)$ is closely match one of the running time in 'Case 1' and 'Case 2', we condsider the ratio table below which takes ratio between the actual running time and the theoretical running time for two cases. If the ratio stays at a certain number, then we can have a conclusion that the actual running time $T(n)$ matches the theoretical running time up to a constant proportionally.

[39]:
```
ratio = {'Case 1' :df['T(n) in usec']/df['Case 1'],
         'Case 2' :df['T(n) in usec']/df['Case 2']}
ratio = pd.DataFrame(ratio)
print(ratio)
```

	Case 1	Case 2
0	3.500000	inf
1	0.333333	0.333333
2	0.033333	0.008197
3	0.060606	0.006704
4	0.056667	0.003537
5	0.069291	0.002768
6	0.071061	0.002840
7	0.071930	0.001998
8	0.078018	0.001592
9	0.079280	0.001239
10	0.080524	0.001258
11	0.097818	0.001528

From the table, it seems that for larger primes, the ratio stays around 0.07~0.09 for Case 1 and stays around 0.001~0.002 for Case 2. It is hard to tell which case better matches with the actual running time.

3.2 The second expriments is testing for large primes $2ˆx - k $, which is of x bits.

(Refer:https://primes.utm.edu/lists/2small/0bit.html)

We repeat the same procedures for a set of large primes.

[35]:
```
primes2 = [2**10-3, 2**20-3, 2**30-35, 2**40 - 87, 2**50-27, 2**60 - 93]
```

```
runtime2 =[]
for num in primes2:
    st = time.time()
    factor(num)
    et = time.time()
    runtime2.append(et-st)
```

[36]:
```
case1_T2 = [math.floor(math.sqrt(num)) for num in primes2]
case2_T2 = [math.floor(math.sqrt(num) * (math.floor(math.log(num,10)))**2)
            for num in primes2]
Tn_usec2 = [math.floor(t*1e6) for t in runtime2]
data2 = {'T(n) in usec' : Tn_usec2, 'n' : primes2,
         'Case 1': case1_T2,'Case 2': case2_T2}
df2 = pd.DataFrame(data2)
print(df2)
```

```
   T(n) in usec                    n       Case 1        Case 2
0             9                 1021           31           287
1            93              1048573         1023         36863
2          3096           1073741789        32767       2654207
3        134025        1099511627689      1048575     150994943
4       3038723     1125899906842597     33554431    7549747199
5      94094946  1152921504606846883   1073741823  347892350975
```

[40]:
```
ratio2 = {'Case 1' :df2['T(n) in usec']/df2['Case 1'],
          'Case 2' :df2['T(n) in usec']/df2['Case 2']}
ratio2 = pd.DataFrame(ratio2)
print(ratio2)
```

```
     Case 1    Case 2
0  0.290323  0.031359
1  0.090909  0.002523
2  0.094485  0.001166
3  0.127816  0.000888
4  0.090561  0.000402
5  0.087633  0.000270
```

From the ratio table, it seems that for Case 1, the ratio stays around 0.08~0.09, which is similar to the result we see in the first expriment. However, the ratio for Case 2 keeps changing.

Based on the limited expriments above, I would see Case 1(assuming that multiplications and division (and additions) take constant time) better matches the actual running time.

3.3 How large can n be so that is approximately 5 minutes. What if is 5 hours? 5 days?

Based on the above expriments, the ratio: **'T(n) in usec' / 'Case 1'**, or equivalently **T(n) * 1e6** / $O(\sqrt(n))$ is about 0.09. Given that 5 min = 3e+8 usec, we can derive that n can be $(3e + 8/0.09)^2 = 1.11e + 19$ which is around 20 decimal digits.

For 5 hours, n can be of 20 * 60 = 1200 decimal digits, and for 5 days, n can be of 20 * 24 * 60 = 28800 decimal digits.

4 Correctness proof

4.1 Invariant of the loop

Outer loop: Before the start of each loop, for any j, 1 < j < i, j is not a factor of n, and the products of all the elements saved in the list 'rlt' and n equals to the original input number(saved as 'input').

Inner loop: Each element i saved in the list rlt is a prime factor.

4.2 Proof of correctness

4.2.1 Initialization

Before the start of the first iteration of the outer loop, i = 2, n = input. For 1 < j < i=2 is an empty set, and rlt is an empty list. So, the outer loop invariant is ture.

4.2.2 Maintenance

Suppose i = k and n = m in the begining of an outer iteration. Assume that the outer loop invariant holds, then we have for any j, 1 < j < i=k, j is not a factor of n = m. And the products of all the elements saved in the current list 'rlt' and n=m equals to the original input.

- At the start of the first iteration of the inner loop, we have i = k and n % i == 0. Since from the outer loop invariant we know n=m is a factor of the original input and here k is a factor of n, thus k is also a factor. We also have k is a prime number, this is because for any j, 1 < j < k, j is not a factor of n. Otherwise, if k is not prime, there should be a j < k that divides k, which would then be a factor of n. This contradictes the outer loop invariant. Thus, the inner loop invariant holds intially.
- Suppose n = q in the begining of an iteration of the inner loop, we have i = k and q % i ==0. Follow the same logic above, we have k is a prime factor.
- When the inner loop terminates, the new n is not divisible by k any more. We prove that each i saved in the list rlt is a prime factor.

When this outer iteration ends, the new n is not divisible by k. And for any j, 1 < j < k+1, j is not a factor of the new n. Otherwise, j would also be a factor of the m since the new n is a factor of m, which contradicts the outer loop invariance assumption.

During this outer iteration, when a k is saved in the list rlt, it means k is a factor of n. Thus, the new n times all of the k saved in this iteration would be m. Since the outer loop invariant holds before the start of this iteration, we know the product of m and all the elements saved before this iteration equals to the orignal input. Thus, we have the product of the new n and all the element in the rlt equals to the original input when this outer iteration ends.

We prove that the outer loop invariant holds at the end of iteration of the outer loop.

4.2.3 Termination

When the outer loop derminates, i = int($\sqrt(n)$) + 1. For any j, 1 < j < $\sqrt(n) + 1$, j is not a factor of n, then n must be a prime number or 1. (This is because any composite number must has at least one prime factor less than or equal to the squre root of itself). Also, loop invariant makes sure that n times all the elements in the list rlt equals to the original input.

After the outer loop ends, if n > 1 and n is not equal to the orignal number, we add n to the rlt list, which then contains all the prime factors of the original input. Otherwise, when n is 1 or n equals to the input number, the list rlt should be an empty list(since the input is not divided by any number).