Aashish Adhikari

CS 519 Algorithms Design, Analysis, and Implementation

Fall 2019 Homework 1

1. **Write a function named *factoring* that returns all prime factors of an integer. For example, factors(12) returns [2,2,3]. If the input is a prime or 1 it returns an empty list. The factors should be listed in increasing order.**

   Solution:

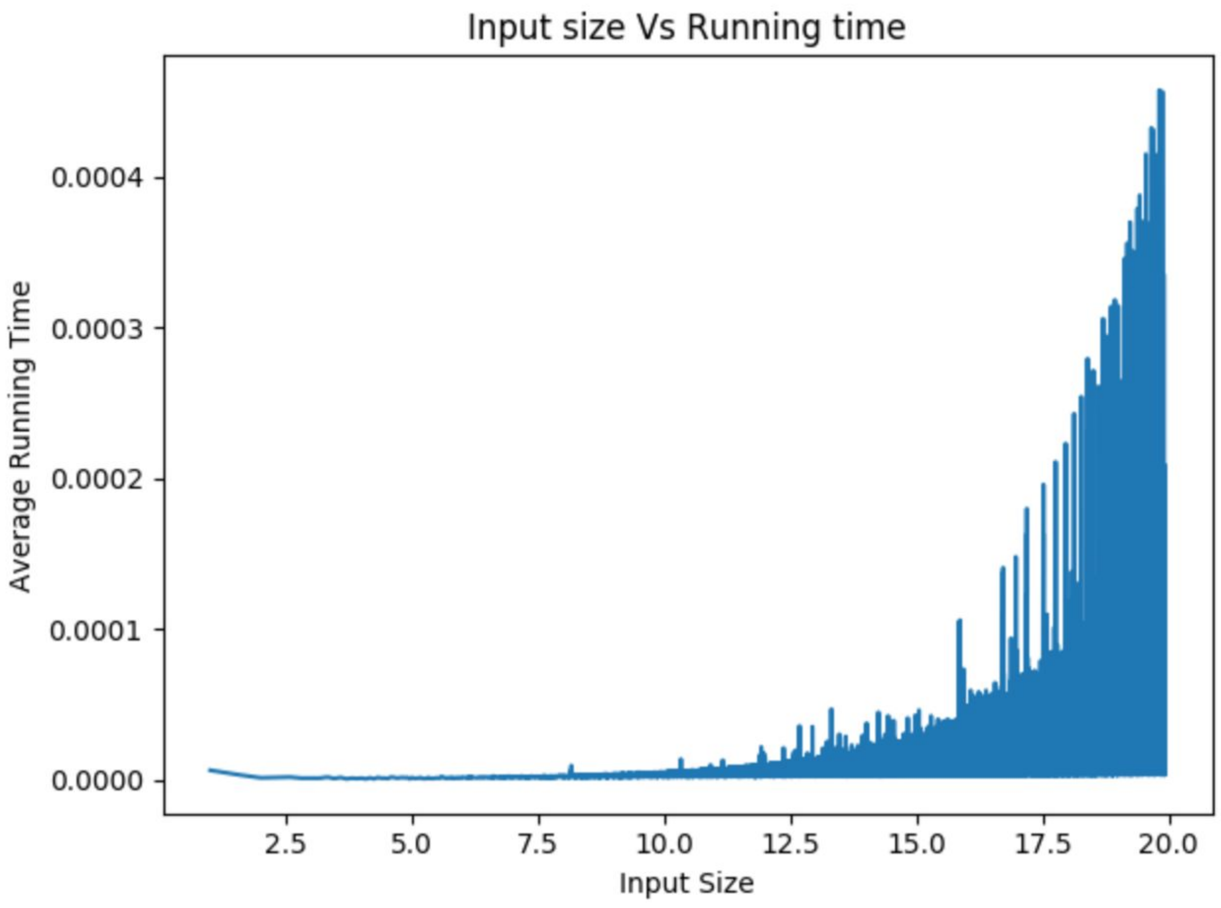
```
def factoring(original_inte):
  import math, time

  array_of_factors = []

  starting_time = time.time()

  if original_inte <= 0:
    print("Invalid input provided. Re-run with a positive integer input. Terminating the
program.")
    return -1

  '''First divisor is 2 which is also the only prime factor , thus divide until 2 is able to
divide to make the number odd'''

  while (original_inte % 2 == 0):
    array_of_factors.append(2)
    original_inte = original_inte / 2
    # print(str(original_inte) + "\t")

  # old code
  '''Now we will check only with odd numbers.'''
  odd_divisor = 3

  while original_inte > 1:
    while original_inte % odd_divisor == 0:
      array_of_factors.append(odd_divisor)
      original_inte /= odd_divisor

    '''After division fails, go to next odd number now.'''
    odd_divisor = odd_divisor + 2
    if odd_divisor > math.sqrt(original_inte):
      if original_inte > 1:
        array_of_factors.append(int(original_inte))
      break
```

```
if len(array_of_factors) == 1:
    return []
return array_of_factors
```

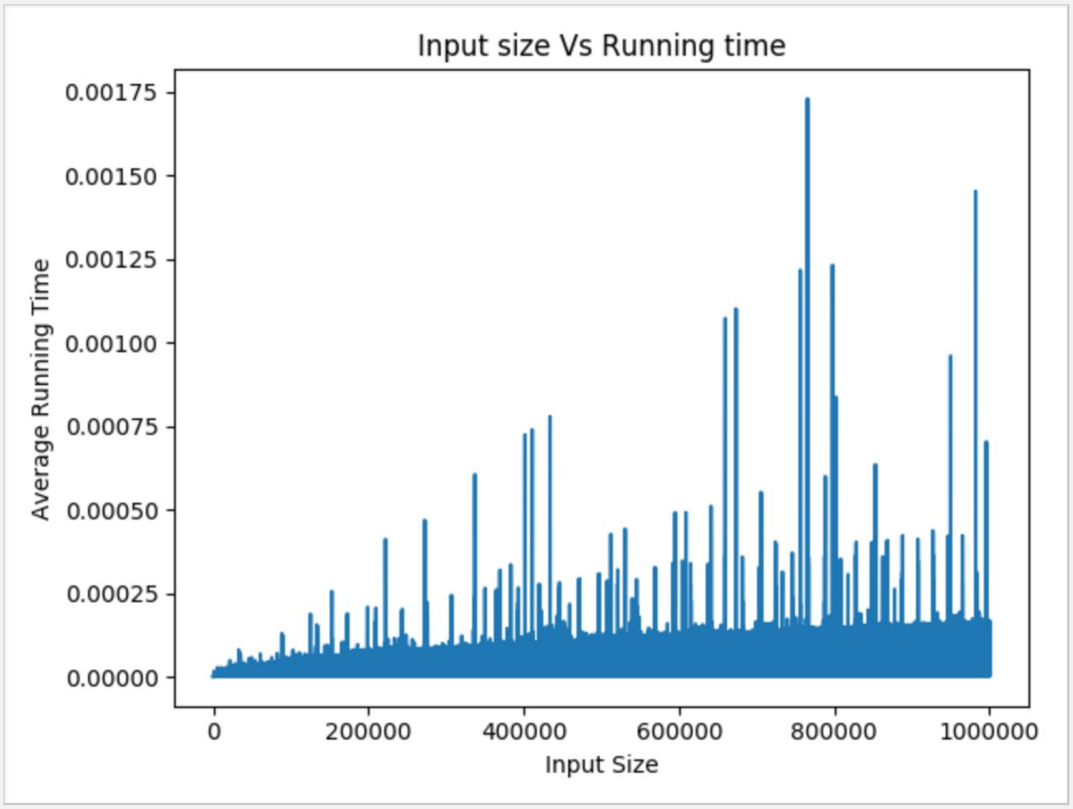
2. **Derive the asymptotic complexity of the running time of your algorithm. You can assume that all primitive Python functions take constant time.**

   Solution:
   We will ignore simple statements that take constant time.

   Case 1: When the input is considered as the number of bits used to represent the input value, the time taken shows an exponential increase as below.



Case 2: When the input is considered as the base 10 integer passed, the following is the trend observed. For my code, the worst case is when the input is a prime number and whenever a prime is passed, the time taken shows a sudden rise as shown below.



Let us consider case 2 to answer this question. However, we will also convert it into case 1 and see the answer.

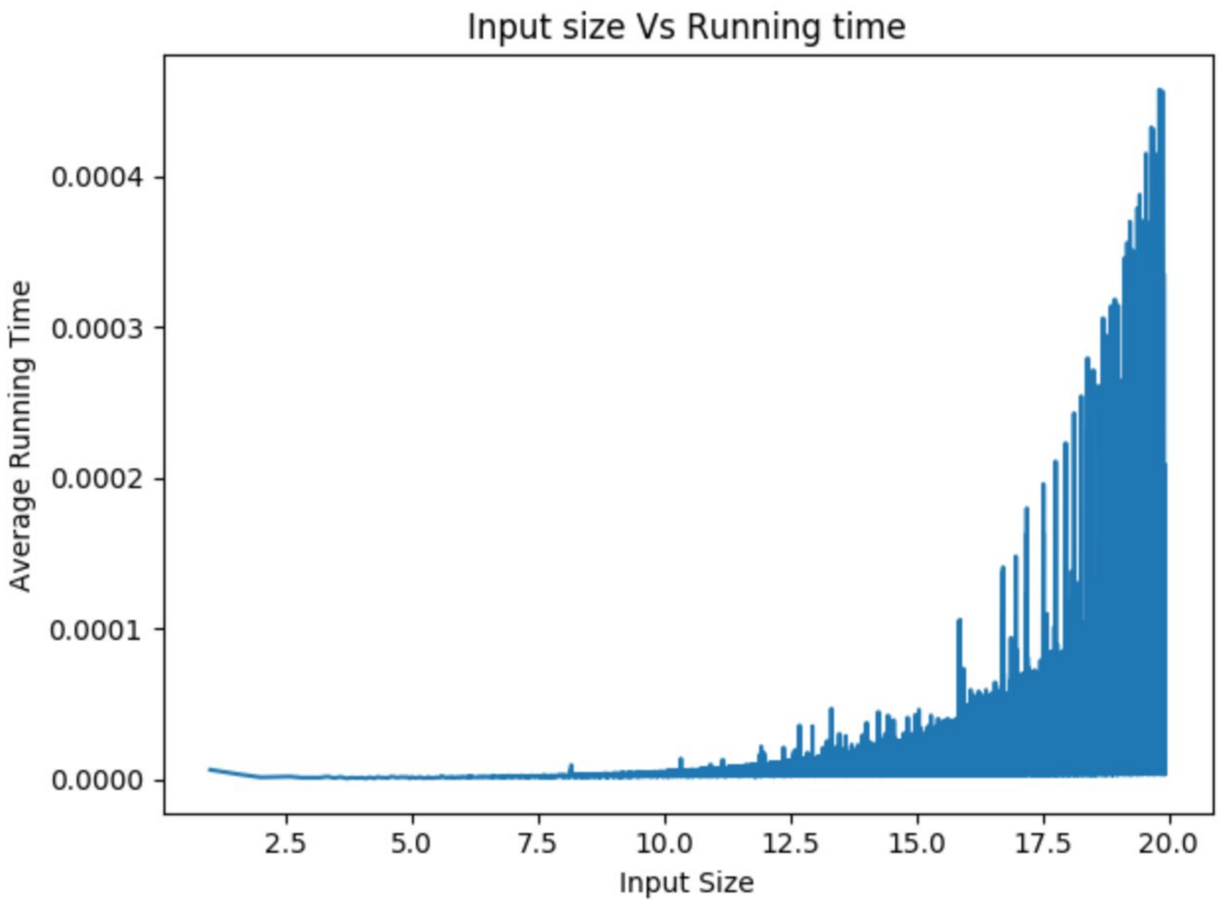
For case 2, for the loop that calculates the number of 2's that fit into the input, the complexity is O($\log_2 n$). This will be added to the nested while loops' time complexity.

The second while loop now checks for odd prime factors by incrementing the divisor by 2 in each iteration, starting at 3. If the number is divisible by 3 multiple times, its executed with the worst case of $\log_k n$ where k == 3. However, this is not the worst case. Rather, if the input to this “odd input section” is a (large) prime number, the second while loop is called n/2 times and the outer while loop only 1 time. Hence considering the worst case of a prime number input, **the time complexity is O (n/2) → Also, O(n)**

If we want to convert it into Case 1, consider input = $\log_2 n$
$\log_2 t(n) = O(\log_2 n)$
$\log_2 t(n) = O(m)$
$t(n) = O(2^m)$

3. **Plot the running time of your algorithm with respect to the *size* of the input. How do you measure the size of the input? What mathematical function of the form derived in question 2 best characterizes the running time *T(n)*?**
   Solution

PLOT: Note that the input size is here in terms of ($\log_2$ORIGINAL_INPUT)



Size of the input: We take the number of bits required to represent the original input N. Here, size of the input = $\log_2 N$.

Mathematical function of the form derived in question 2 best characterizes the running time *T(n):*

The form derived there is best described in 2 ways for 2 cases: **Case 1 the exponential function 2 ^m and Case 2 Linear function n/2** . However, we are using the log of the input to plot above.

4. **Give a table *T(n)* vs. *n.* How large can *n* be so that *T(n)* is approximately *5* minutes. What if *T(n)* is *5* hours? *5* days? Factoring is a fundamental crypto-primitive that underlies modern cryptography. What size of *n* makes it practically impossible to factorize*,* e.g.*, T(n) > 100* years.**

TABLE

Original Input	Log of input	Runtime
1	1.0	5.0067901611328125e-06
10	3.4594316186372978	1.1920928955078125e-06
100	6.6582114827517955	2.1457672119140625e-06
1000	9.967226258835993	3.0994415283203125e-06
10000	13.287856641840545	1.1920928955078125e-05
100000	16.60965490131509	2.0503997802734375e-05

I used curve fitting to calculate the time required. For example: a portion of used data looked like below:

Input	**Time**
1.1920928955078125e-06	1
2.8371810913085938e-05	100001
3.933906555175781e-05	200001
1.1682510375976562e-05	300001
5.2928924560546875e-05	400001
9.083747863769531e-05	500001
8.821487426757812e-06	600001
0.00012874603271484375	700001
2.5272369384765625e-05	800001
0.00015282630920410156	900001
1.5735626220703125e-05	1000001
9.059906005859375e-06	1100001

```
1.0967254638671875e-05 1200001
2.6464462280273438e-05 1300001
2.3126602172851562e-05 1400001
8.749961853027344e-05 1500001
7.724761962890625e-05 1600001
2.002716064453125e-05 1700001
2.193450927734375e-05 1800001
3.7670135498046875e-05 1900001
0.00018310546875 2000001
0.00023484230041503906 2100001
3.314018249511719e-05 2200001
5.888938903808594e-05 2300001
0.00024056434631347656 2400001
3.123283386230469e-05 2500001
2.288818359375e-05 2600001
5.6743621826171875e-05 2700001
0.00028133392333984375 2800001
1.4543533325195312e-05 2900001
```

For t= 5 minutes:

5 × 60 × 2^20 / 0.0015⟶38 bits

t = 5hrs → 44 bits
t = 5 days → 49 bits
t = 100 years and more → 61 bits

5. **How long did you take for the following tasks?**

   Think of an algorithm and write code for it.

It was easy to think of a code at first but then it did not pass the time limit check for case 8. It took about 30 minutes to finish up. Then it took another 2 hours to come up with another code that passed that test as well.

   Get the code to run

I think this is answered above as well. So writing code was easy I suppose but making it pass all the cases took some time.

Answer the questions 2-4 above.

About a few hours since I was confused about what to use as the input size at first and had to talk to the instructor as well as discuss with friends on if I should use the original value of input or the log value. Also, me and my friends felt like the wording of the questions were very confusing and could have been written more clearly.