NYU, Tandon School of Engineering

CS-UY 1134 Data Structures and Algorithms — Fall 2023

Homework #2
Due by Thursday 10/5, 11:59pm

Submission instructions:

1. You should turn in 6 files:
    - A '.pdf' file for the first two questions.
      Name your file: 'YourNetID_hw2_q1to2.pdf'
    - 5 '.py' files, one for each question 3-7.
      Name your files: 'YourNetID_hw2_q3.py' and 'YourNetID_hw2_q4.py', etc.

   Note: your netID follows an abc123 pattern, not N12345678.

2. You should submit your homework via Gradescope. For Gradescope's autograding feature to work:
    a. Name all classes, functions and methods **exactly as they are in the assignment specifications**.
    b. Make sure there are **no print statements** in your code. If you have tester code, please put it in a "main" function and **do not call it**.

Question 1:

Use the definitions of O and of Θ in order to show the following:

a. $5n^3 + 2n^2 + 3n = O(n^3)$

b. $\sqrt{7n^2 + 2n - 8} = \Theta(n)$

c. Show that if $d(n)=O(f(n))$ and $e(n)=O(g(n))$, then the product $d(n)e(n)$ is $O(f(n)g(n))$.

Question 2:

Give a θ characterization, in terms of *n*, of the running time of the following four functions:

$\boxed{\Theta(n^2)}$

```
def example1(lst):
 """Return the sum of the prefix sums of sequence S."""
     n = len(lst)|
     total = 0 |
     for j in range(n):n
         for k in range(1+j):(n+1)/2
             total += lst[k] |
     return total
```

$\boxed{\Theta(n)}$

```
def example2(lst):
"""Return the sum of the prefix sums of sequence S."""
     n = len(lst) |
     prefix = 0 |
     total = 0 |
     for j in range(n):n
         prefix += lst[j] |
         total += prefix |
     return total
```

$\Theta(\log(n))$

```
def example3(n):
    i = 1   |
    sum = 0 |
    while (i < n*n): n
        i *= 2  |
        sum += i |
    return sum
```

$\Theta(n)$

```
def example4(n):
    i = n   |
    sum = 0 |
    while (i > 1):log_2(n)
        for j in range(i):
            sum += i*j |
        i //= 2
    return sum
```

$n + \frac{n}{2} \ldots = 2n$

Question 3:
Implement a function `def factors(num)`. This function is given a positive integer `num`, and returns a generator, that when iterated over, it will have all of `num`'s divisors in an **ascending order**.

For Example, if we execute the following code:
```
for curr_factor in factors(100):
   print(curr_factor)
```
The expected output is:
```
1 2 4 5 10 20 25 50 100
```

Implementation requirement: Pay attention to the running time of your implementation. The **for** loop like the above, should run in a total cost of $\Theta(\sqrt{num})$.

Question 4:
The number ***e*** is an important mathematical constant that is the base of the natural logarithm. ***e*** also arises in the study of compound interest, and in many other applications.
Background of ***e***: https://en.wikipedia.org/wiki/E_(mathematical_constant)

e can be calculated as the sum of the infinite series:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

The value of ***e*** is approximately equal to 2.71828. We can get an approximate value of ***e***, by calculating only a partial sum of the infinite sum above (the more addends we add, the better approximation we get).

Implement the function `def e_approx(n)`. This function is given a positive integer `n`, and returns an approximation of ***e***, calculated by the sum of the first (`n`+1) addends of the infinite sum above.

To test your function, use the following main:
```
def main():
    for n in range(15):
        curr_approx = e_approx(n)
        approx_str = "{:.15f}".format(curr_approx)
        print("n =", n, "Approximation:", approx_str)
```

Note: Pay attention to the running time of `e_approx`. By calculating the factorials over for each addend, your running time could get inefficient. An efficient implementation would run in $\Theta(n)$.

Question 5:

Implement the function **`def`** `split_parity(lst)`. That takes `lst`, a list of integers. When called, the function changes the order of numbers in `lst` so that all the odd numbers will appear first, and all the even numbers will appear last. Note that the inner order of the odd numbers and the inner order of the even numbers don't matter.

For example, if `lst` is a list containing [1, 2, 3, 4], after calling `split_parity`, `lst` could look like: [3, 1, 2, 4].

Implementation requirements:

1. You are **not allowed** to use an auxiliary list (a temporary local list).
2. Pay attention to the running time of your implementation. An efficient implementation would run in a linear time. That is, if n is the length of `lst`, the running time should be $\Theta(n)$.

Question 6:

Implement the function **`def`** `two_sum(srt_lst, target)`. This function is given:

- `srt_lst` - a list of integers arranged in a **sorted** order
- `target` - an integer

When called, it returns two indices (collected in a tuple), such that the elements in their positions add up to `target`. If there are no such indices, the function should return `None`.

For example, if `srt_lst=[-2, 7, 11, 15, 20, 21]`, and `target=22`, your function would return (1, 3) because `srt_lst[1]` + `srt_lst[3]`=7+15=22

Note: Pay attention to the running time of your function. Aim for a linear time algorithm.

Question 7:

Implement the function **`def`** `findChange(lst01)`. This function is given `lst01`, a list of integers containing a sequence of 0s followed by a sequence of 1s.
When called, it returns the index of the first 1 in `lst01`.

For example, if `lst01` is a list containing `[0, 0, 0, 0, 0, 1, 1, 1]`, calling `findChange(lst01)` will return 5.

Note: Pay attention to the running time of your function. If `lst01` is a list of size n, an efficient implementation would run in logarithmic time (that is $\Theta(log_2(n))$).