Lab 10 – Algorithms and Time Complexity

Summission

- `Lab10_Time_Complexity_Analysis.pdf`
- `Lab10_<your_name>_<SID>_Q1.py`
- `Lab10_<your_name>_<SID>_Q2.py`
- `Lab10_<your_name>_<SID>_Q3.py`

1. (3 Pts) Learn to analyze the time complexity of programs

Answer the questions in the Word file of "Lab10_Time_Complexity_Analysis", and submit pdf.

2. Program Assignment

1. (2pts) Perfect Square

A **perfect square** is an integer that is the square of an integer. Write a Python program that given a positive integer *num*, return *true* if *num* is a perfect square or *false* otherwise.

Example: *num = 16* should return *true*; *num = 14* should return *false*

Note:

- You must not use any built-in library function, such as *sqrt*.
- The time complexity of your program should be log(n). (Hint: use binary search)

2. (2Pts) Power x^n

Implement a Python function to calculates x (x>0) to the power of n (i.e., x^n).

Note:

- You must not use any built-in functions, such as *pow*
- n is an arbitrary integer (can be negative or 0)
- Please specify the time complexity in the Θ-notation in the comment.
- **Optinal (extra 1pt):** come up with an approach with log(N) time complexity.

```
# Example
print(my_power(2.0, 10))                # 1024.0000
print(my_power(2.0, -2))                # 0.2500
print(my_power(2.1, 3))                 # 9.2610
```

3. (2 pts) Check Anagram

An **anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, using all the original letters exactly once.
Write a python function that given two strings *s* and *t*, return *true* if *t* is an anagram of *s*, and *false* otherwise.

Note: Please specify the time complexity of each approach in the Θ-notation in the comment.

```
def is_anagram(s, t):
    # function body

print(is_anagram(“anagram”, “nagaram”))          # True
print(is_anagram(“anagrm”, “nagaram”))           # False
print(is_anagram(“rat”, “car”))                  # False
```