CS566HW1-2024

September 4, 2024

0.1 Homework 1. Introduction to algorithms

This is first Homework for CS 566.

0.2 Task 1. Solve the problem "Valid Parenthesis" from https://leetcode.com/problems/valid-parentheses/ using Python3

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[43]:
```
class Solution:
    def isValid(self, s: str) -> bool:
        # if the length of string is not even, then we automatically know it's␣
↪not balanced and return false early
        if len(s) %2 != 0:
            return False

        # try with stack, approach is to push each opening parentheses (,{,[ to␣
↪stack and when we encounter closing
        stack = []


        for char in s:
            # check for opening parentheses
            if char in ['{', '(', '[']:
                stack.append(char)
            else:
                # if char is closing parentheses with empty stack, false
                if stack is None:
                    return False
                # take top element
                stack_top = stack.pop()
                # check end of stack to compare open and close, if not matching␣
↪then false
                if stack_top == '(' and char != ')':
                    return False
                if stack_top == '{' and char != '}':
                    return False
```

```
                if stack_top == '[' and char != ']':
                    return False

        # if not all elements popped, that means there is unclosed parentheses
        if len(stack) != 0:
          return False

        # otherwise it is valid
        return True
```

0.2.1 Do not modify the testing code below. If you get message "Mistake in test case #", it means that you algorithm is incorrect.

[44]:
```
#test_case_1
expected, s = True, "()"
actual = Solution().isValid(s)
assert expected==actual, "Mistake in test case 1"

#test_case_2
expected, s = True, "()[]{}"
actual = Solution().isValid(s)
assert expected==actual, "Mistake in test case 2"

#test_case_3
expected, s = False, "(]"
actual = Solution().isValid(s)
assert expected==actual, "Mistake in test case 3"
print('All tests pass!')
```

```
All tests pass!
```

0.2.2 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n) - Pushing char to stack worst case is n

Time Analysis: O(n) - For loop to process each char in s

0.3 Task 2. Solve the problem "Nim Game" from https://leetcode.com/problems/nim-game/description/ using Python3

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[45]:
```
class Solution:
    def canWinNim(self, n: int) -> bool:
        # if n is divisible by 4, it will always go down to the last 4 stones␣
 ↪since both play optimally and opponent will always win
```

```
        # otherwise I should be able to win
        # eg. n = 8, Whatever I take he takes something and next round it will␣
↪have 4 stones and I will lose
        return n%4!=0
```

[46]:
```
#test_case_1
expected, n = False, 4
actual = Solution().canWinNim(n)
assert expected==actual, "Mistake in test case 1"

# #test_case_2
expected, n = True, 2
actual = Solution().canWinNim(n)
assert expected==actual, "Mistake in test case 2"

# #test_case_3
expected, n = True, 1
actual = Solution().canWinNim(n)
assert expected==actual, "Mistake in test case 3"
print('All tests pass!')
```

```
All tests pass!
```

0.3.1 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(1)

Time Analysis: O(1)

0.4 Task 3. Theoretical problem.

There are 8 algorithms with complexities: - $log(n)$ - $\sqrt{n}$ - n^3 - n - $nlog(n)$ - n^2 - 2^n - $n!$

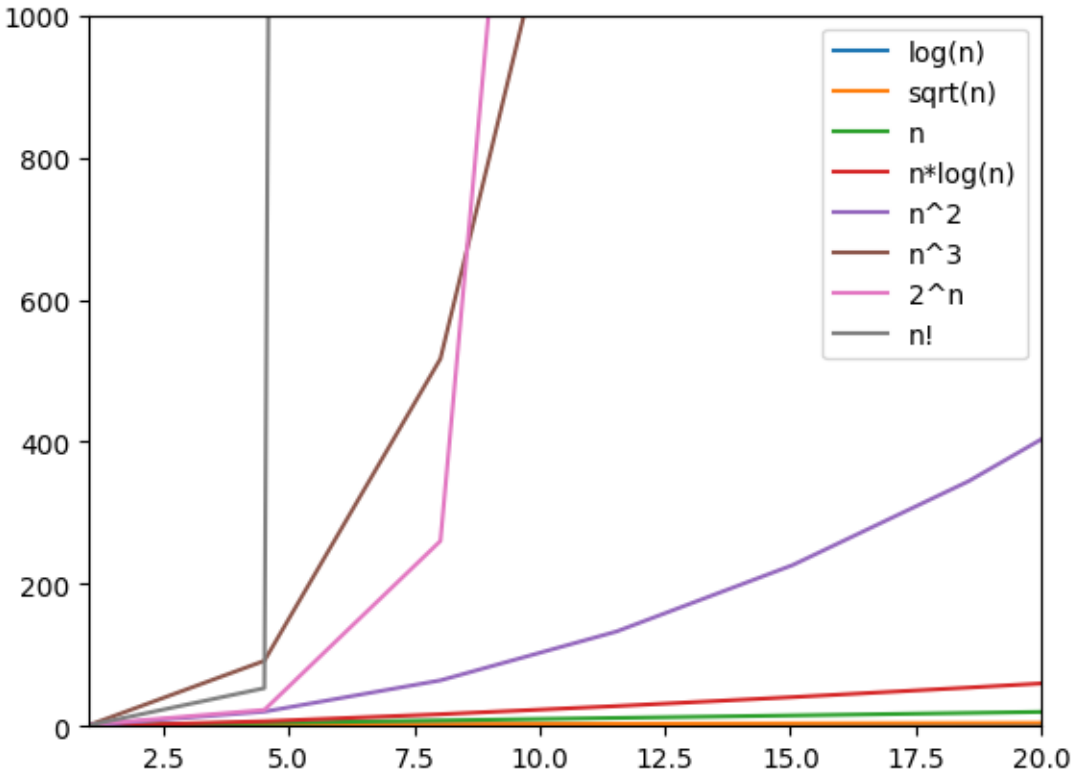
Order them in the order from fastest to slowest (5 points)

Answer:

- $log(n)$
- $\sqrt{n}$
- n
- $nlog(n)$
- n^2
- n^3
- 2^n
- $n!$

[51]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import factorial
```

```
n = np.linspace(1,700,200)

functions = [np.log(n), np.sqrt(n), n, n*np.log(n), n**2, np.square(n), np.
 ↪power(n,3), 2**n]

functions = {
    'log(n)': np.log(n),
    'sqrt(n)': np.sqrt(n),
    'n': n,
    'n*log(n)': n*np.log(n),
    'n^2': np.square(n),
    'n^3': np.power(n,3),
    '2^n': 2**n,
    'n!': factorial(n)

}

for label, func in functions.items():
  plt.plot(n,func, label=label)

plt.ylim(0, 1000)
plt.xlim(1, 20)
plt.legend()

plt.show()
```


1000
800
600
400
200
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
log(n)
sqrt(n)
n
n*log(n)
n^2
n^3
2^n
n!