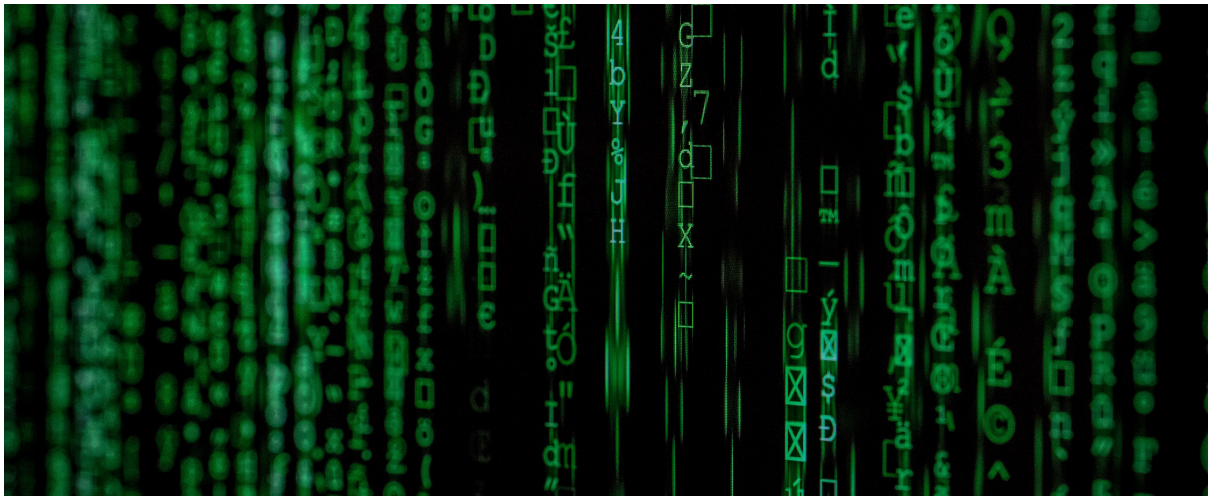


CSE321 - Introduction to Algorithm Design - HW04

Name Surname	Muhammed Oğuz
Number	1801042634

Table of Contents

General Explanation About Python

Typing Information

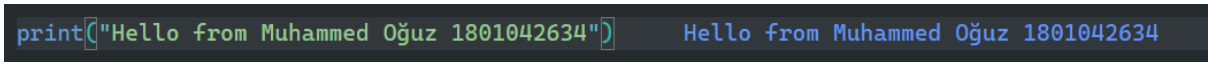
I use my python with typing module. It is a hobbit from my Java and C++ knowledge. So you will see an import and type notations.

```
from typing import List # a typing library from standart python3.8
def foo(variable : int) -> int: # I specify variable type and return type like this
```

Test Result Information

Additionally, I use an extension that directly shows output of the results in VSCode. It is called `Wolf`.

Here is an example



Left Side is the code and right side with blue color is the output. With help of that, I can show my input and output directly in one screenshot.

VS Marketplace Link: https://marketplace.visualstudio.com/items?itemName=traBpUkciP.wolf

First Problem - Cutting

Problem Approach

I noticed that, this problem is easily solvable with just taking $log2$ of n. Since this machine can cut multiple pieces at one, we can bring pieces top each other. Thanks to this, It will allow us to divide separately.

Solution and Analysis

```
def min_cut(n : int) -> int:
    if n == 1:                                  #  O(1)
        return 0                                #  O(1)
    else:
        return 1 + min_cut(math.ceil(n / 2))    # O(logn)
```

Complexity: $\theta(logn)$

Running Results

```
# test
print(min_cut(5))      3
print(min_cut(8))      3
print(min_cut(100))      7
```

Second Problem - The Worst and The Best

Problem Approach

This problem is simply, finding biggest number and smallest number in a List.

Solution and Analysis

```
def min_max(lst : List[int]) -> Tuple[int, int]:
    if len(lst) == 1:                                           # O(1)
        return (lst[0], lst[0])                                 # O(1)
    else:
        mid = len(lst) // 2                                     # O(1)
        left = min_max(lst[:mid])                               # O(log(n))
        right = min_max(lst[mid:])                              # O(log(n))
        return (min(left[0], right[0]), max(left[1], right[1]))     # O(1)
```

Complexity: $\theta(logn)$

Running Results

```
# test
print(min_max([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))     (1, 10)
print(min_max([10, 9, 8, 7, 6, 5, 4, 3, 2, 1]))     (1, 10)
print(min_max([15,6,4,33,4,5,12,5,6,7,8,3,5,2]))     (2, 33)
```

Third Problem - Meaningful

Problem Approach

This problem is actually finding kth biggest number in a list. I have done this before this homework. I will redo my older approach in here.

My approach was finding the smallest element in the list and deleting it. And do this process for `k - 1` times. And after return the smallest in the list.

Solution and Analysis

```
# find smallest in a list
def min_in_list(lst : List[int]) -> int:
    if len(lst) == 1:                   # O(1)
        return lst[0]                   # O(1)
    else:
        mid = len(lst) // 2             # O(1)
        left = min_in_list(lst[:mid])   # O(log n)
        right = min_in_list(lst[mid:])  # O(log n)
        return min(left, right)         # O(1)

# call min_in_list kth times and remove the smallest k times
def remove_kth_smallest(lst : List[int], k : int) -> List[int]:
    if k == 0:                                  # O(1)
        return lst                              # O(1)
    else:
        smallest = min_in_list(lst)            # O(log n)
        lst.remove(smallest)                    # O(n)
        return remove_kth_smallest(lst, k - 1) # O(log n)

# find kth biggest element
def kth_biggest(lst : List[int], k : int) -> int:
    remove_kth_smallest(lst, k - 1)             # O(log n)
    return min_in_list(lst)                     # O(log n)
```

Complexity: $\theta(logn)$

Running Result

```
# test
print(kth_biggest([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 5))     5
print(kth_biggest([10, 9, 8, 7, 6, 5, 4, 3, 2, 1], 5))     5
print(kth_biggest([15,6,4,33,4,5,12,5,6,7,8,3,5,2], 2))     3
```

Fourth Problem - Find Rop

This problem as a bit harder than others.

I notice that, it can be solvable with like merge sort algorithm. I wrote a merge sort algorithm.

It divides all elements like merge sort and checks if the next pair is reversed or not. If so, increment count. After this step, do the merge sort, since other pairs are already checked, we don’t have to check all them indivually. And it gives as reverse pair count.

Solution and Analysis

```
# divide function
def _merge_helper(lst : List[int], low : int, high : int) -> int:
    if (low >= high):                               # O(1)
        return 0                                    # O(1)
```

```
    mid = low + (high - low) // 2                  # O(1)

    left = _merge_helper(lst, low, mid)                    # O(nlogn)
    right = _merge_helper(lst, mid + 1, high)              # O(nlogn)

    mergeResult = merge_like(lst, low, mid, high)   # O(n)

    return left + right + mergeResult              # O(1)

def merge_like(lst : List[int], leftInd : int, midInd : int, rightInd : int) -> int:
    leftLst = lst[leftInd:midInd + 1]                           # O(n)
    rightLst = lst[midInd + 1: rightInd + 1]                    # O(n)

    i = 0
    j = 0
    k = leftInd
    count = 0

    while (i < len(leftLst) and j < len(rightLst)):             # O(n)
        if (leftLst[i] <= rightLst[j]):
            lst[k] = leftLst[i]
            i += 1
        else:
            lst[k] = rightLst[j]
            j += 1
            count += midInd + 1 - leftInd - i
        k += 1

    while (j < len(rightLst)):                                  # O(n)
        lst[k] = rightLst[j]
        j += 1
        k += 1

    while (i < len(leftLst)):                                   # O(n)
        lst[k] = leftLst[i]
        i += 1
        k += 1

    return count

def reversePairCount(lst : List[int]) -> int:
    return _merge_helper(lst, 0, len(lst) - 1)                        # O(nlogn)
```

Complexity: $\theta(nlogn)$

Running Results

```
# test
print(reversePairCount([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))     0
print(reversePairCount([5,1,2,3,4]))     4
print(reversePairCount([10, 9, 8, 7, 6, 5, 4, 3, 2, 1]))     45
print(reversePairCount([15,6,4,33,4,5,12,5,6,7,8,3,5,2]))     55
```

Fifth Problem - Exponential

Problem Approach

This is a very easy problem. It is basically taking a exponential of a given number. We need a base and a exponent.

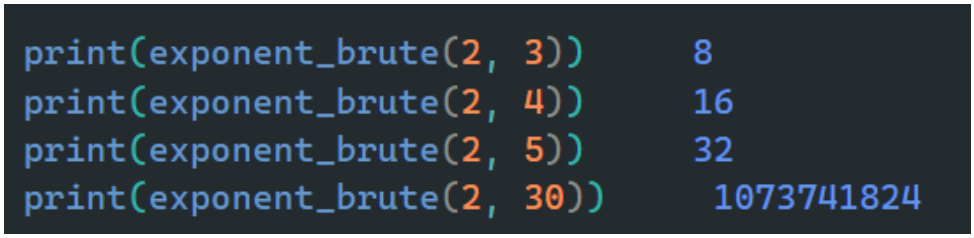
Brute Force

Solution and Analysis

```
# take exponent of a number with brute force
def exponent_brute(base : int, exp : int) -> int:
    if (exp == 0):                  # O(1)
        return 1                    # O(1)
    else:
        result = 1                  # O(1)
        for _ in range(exp):        # O(n)
            result *= base          # O(1)
        return result               # O(1)
```

Complexity: $\theta(n)$

Running Results



Divide and Conquer

Solution and Analysis

```
# take exponent of a number with divide and conquer
def exponent(base : int, exp : int) -> int:
    if (exp == 0):                                                          # O(1)
        return 1                                                            # O(1)
    elif (exp == 1):                                                        # O(1)
        return base                                                         # O(1)
    else:
        if (exp % 2 == 0):                                                  # O(1)
            return exponent(base, exp // 2) * exponent(base, exp // 2)      # O(logn)
        else:
            return base * exponent(base, (exp - 1) // 2) * exponent(base, (exp - 1) // 2)   # O(logn)
```

Complexity: $\theta(logn)$

Running Result

```
# test
print(exponent(2, 3))      8
print(exponent(2, 4))      16
print(exponent(2, 5))      32
print(exponent(2, 30))      1073741824
```

Have a nice day Sir!