Assignment 3 - Divide-n-Conquer Algorithms in Python

This assignment is a part of the course "Data Structures and Algorithms in Python".

In this assignment, you will implement an efficient algorithm for polynomial multiplication.

As you go through this notebook, you will find the symbol **???** in certain places. To complete this assignment, you must replace all the **???** with appropriate values, expressions or statements to ensure that the notebook runs properly end-to-end.

Guidelines

1. Make sure to run all the code cells, otherwise you may get errors like `NameError` for undefined variables.
2. Do not change variable names, delete cells or disturb other existing code. It may cause problems during evaluation.
3. In some cases, you may need to add some code cells or new statements before or after the line of code containing the **???**.
4. Since you'll be using a temporary online service for code execution, save your work by running `jovian.commit` at regular intervals.
5. Questions marked **(Optional)** will not be considered for evaluation, and can be skipped. They are for your learning.
6. If you are stuck, you can ask for help on the [community forum] (TODO - add link). Post errors or ask for hints, but **please don't ask for OR share the full working answer code** on the forum.
7. There are some tests included with this notebook to help you test your implementation. However, after submission your code will be tested with some hidden test cases. Make sure to test your code exhaustively to cover all edge cases.

Important Links

- Submit your work here: https://jovian.ai/learn/data-structures-and-algorithms-in-python/assignment/assignment-3-sorting-and-divide-conquer-practice
- Ask questions and get help: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/assignment-3/89
- Lesson 3 video for review: https://jovian.ai/learn/data-structures-and-algorithms-in-python/lesson/lesson-3-sorting-algorithms-and-divide-and-conquer
- Lesson 3 notebook for review: https://jovian.ai/aakashns/python-sorting-divide-and-conquer

How to Run the Code and Save Your Work

Option 1: Running using free online resources (1-click, recommended): Click the **Run** button at the top of this page and select **Run on Binder**. You can also select "Run on Colab" or "Run on Kaggle", but you'll need to create an account on Google Colab or Kaggle to use these platforms.

Option 2: Running on your computer locally: To run the code on your computer locally, you'll need to set up Python & Conda, download the notebook and install the required libraries. Click the **Run** button at the top of this page, select the **Run Locally** option, and follow the instructions.

Saving your work: You can save a snapshot of the assignment to your Jovian profile, so that you can access it later and continue your work. Keep saving your work by running `jovian.commit` from time to time.

```
project='python-divide-and-conquer-assignment'
```

```
!pip install jovian --upgrade --quiet
```

```
import jovian
jovian.commit(project=project, privacy='secret', environment=None)
```

```
[jovian] Updating notebook "megha-goyate/python-divide-and-conquer-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment
```

```
'https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment'
```

Problem Statement - Polynomial Multiplication

> Given two polynomials represented by two lists, write a function that efficiently multiplies given two polynomials. For example, the lists `[2, 0, 5, 7]` and `[3, 4, 2]` represent the polynomials $2 + 5x^2 + 7x^3$ and $3 + 4x + 2x^2$.
>
> Their product is
>
> $$(2 \times 3) + (2 \times 4 + 0 \times 3)x + (2 \times 2 + 3 \times 5 + 4 \times 0)x^2 + (7 \times 3 + 5 \times 4 + 0 \times 2)x^3 + (7 \times 4 + 5 \times 2)x^4 + (7 \times 2)x^5$$
> i.e.
>
> $$6 + 8x + 19x^2 + 41x^3 + 38x^4 + 14x^5$$
>
> It can be represented by the list `[6, 8, 19, 41, 38, 14]`.

The Method

Here's the systematic strategy we'll apply for solving problems:

1. State the problem clearly. Identify the input & output formats.
2. Come up with some example inputs & outputs. Try to cover all edge cases.
3. Come up with a correct solution for the problem. State it in plain English.
4. Implement the solution and test it using example inputs. Fix bugs, if any.
5. Analyze the algorithm's complexity and identify inefficiencies, if any.
6. Apply the right technique to overcome the inefficiency. Repeat steps 3 to 6.

This approach is explained in detail in Lesson 1 of the course. Let's apply this approach step-by-step.

Solution

1. State the problem clearly. Identify the input & output formats.

While this problem is stated clearly enough, it's always useful to try and express in your own words, in a way that makes it most clear for you.

Problem

> I would say it is Polynomial Multiplication Problem.

Input

1. [5, 0, 10, 6]
2. [1, 2, 4]

Output

1. [5, 10, 30, 36, 52]

Based on the above, we can now create a signature of our function:

```
def multiply_basics(a, b):
    len_a = len(a)
    len_b = len(b)
    product= []
    product_len = len_a + len_b - 1
    for i in range(product_len):
        product.append(0)
    for i in range(len_a):
        for j in range(len_b):
            product[i+j] += a[i] * b[j]
    return product


multiply_basics([1],[0])
```

```
[0]
```

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/python-divide-and-conquer-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment
```

```
'https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment'
```

2. Come up with some example inputs & outputs. Try to cover all edge cases.

Our function should be able to handle any set of valid inputs we pass into it. List a few scenarios here:

1. [2, 3, 5, 7], [3, 4, 2]
2. [5, 0, 10, 6], [1, 2, 4]
3. [0], [1]
4. [1], [0]
5. [0], [0]

(add more if required)

Create a test case of each of the above scenarios. We'll express our test cases as dictionaries, to test them easily. Each dictionary will contain 2 keys: `input` (a dictionary itself containing one key for each argument to the function and `output` (the expected result from the function).

```
test0 = {
    'input': {
        'poly1': [2, 0, 5, 7],
        'poly2': [3, 4, 2]
    },
    'output': [6, 8, 19, 41, 38, 14]
}
```

```
test1 = {
    'input': {
        'poly1': [5, 0, 10, 6],
        'poly2': [1, 2, 4]
    },
    'output': [5, 10, 30, 36, 52, 24]
}
```

```
test2 = {
    'input': {
        'poly1': [0],
        'poly2': [1]
    },
    'output': [0]
}
```

```
test3 = {
    'input': {
        'poly1': [1],
        'poly2': [0]
    },
    'output': [0]
}
```

```
test4 = {
    'input': {
        'poly1': [0],
        'poly2': [0]
    },
    'output': [0]
}
```

```

```

```
# add more if required
```

```

```

```

```

Let's store all the test cases in a list, for easier automated testing.

```
tests = [test0,test1, test2, test3, test4]
```

```

```

```

```

```

```

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/python-divide-and-conquer-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment
```

```
'https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment'
```

3. Come up with a correct solution for the problem. State it in plain English.

Our first goal should always be to come up with a *correct* solution to the problem, which may not necessarily be the most *efficient* solution.

Here's the simplest solution: If you have lists `poly1` and `poly2` representing polynomials of length m and n respectively, the highest degree of the exponents are $m-1$ and $n-1$ respectively. Their product has the degree

$(m-1)+(n-1)$ i.e $m+n-2$. The list representing the product has the length $m+n-1$. So, we can create a list `result` of length $m+n-1$, and set

`result[k]` = Sum of all the pairs `poly1[i] * poly2[j]` where `i+j = k`

Example:

$(2+5x^2+7x^3)\times(3+4x+2x^2)$

$=(2\times 3)+(2\times 4+0\times 3)x+(2\times 2+3\times 5+4\times 0)x^2+(7\times 3+5\times 4+0\times 2)x^3+(7\times 4+5\times +(7\times 2)x^5$

$=6+8x+19x^2+41x^3+38x^4+14x^5$

Explain this solution in your own words below:

1. **m is the len(poly1)**
2. **n is the len(poly2)**
3. **m-1 is the highest degree for poly1**
4. **n-1 is the highest degree for poly2**
5. **len of the resultant list will be m+n-1**

(add more steps if required)

Let's save and upload our work before continuing.

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/python-divide-and-conquer-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment
```

```
'https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment'
```

4. Implement the solution and test it using example inputs. Fix bugs, if any.

Implement the solution

```
def multiply_basic(a, b):
    len_a = len(a)
    len_b = len(b)
    product= []
    product_len = len_a + len_b - 1
    for i in range(product_len):
        product.append(0)
    for i in range(len_a):
        for j in range(len_b):
```

```
            product[i+j] += a[i] * b[j]
    return product
```

Test your solution using the test cases you've defined above.

```
multiply_basics([2, 0, 5, 7], [3, 4, 2])
```

```
[6, 8, 19, 41, 38, 14]
```

```
multiply_basics([5, 0, 10, 6], [1, 2, 4])
```

```
[5, 10, 30, 26, 52, 24]
```

```
multiply_basics([0], [1])
```

```
[0]
```

```
multiply_basics([1], [0])
```

```
[0]
```

```
multiply_basics([0], [0])
```

```
[0]
```

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/python-divide-and-conquer-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment
```

```
'https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment'
```

5. Analyze the algorithm's complexity and identify inefficiencies, if any.

Can you analyze the time and space complexity of this algorithm?

```
multiply_basic_time_complexity = 'O(n^2)'
```

```
multiply_basic_space_complexity = 'O(n^2)'
```

```

```

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/python-divide-and-conquer-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment
```

```
'https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment'
```

6. Apply the right technique to overcome the inefficiency. Repeat steps 3 to 6.

We can apply the divide and conquer technique to solve this problem more efficiently. Given two polynomials `A` and `B`, we can express each of them as a sum of two polynomials as follows:

The Divide Step: Define

$$\begin{aligned} A_0(x) &= a_0 + a_1 x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1}, \\ A_1(x) &= a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1} x + \cdots + a_n x^{n - \lfloor \frac{n}{2} \rfloor}. \end{aligned}$$

Then $A(x) = A_0(x) + A_1(x) x^{\lfloor \frac{n}{2} \rfloor}$.

Similarly we define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\lfloor \frac{n}{2} \rfloor}.$$

Then

$$\begin{aligned} A(x)B(x) &= A_0(x)B_0(x) + A_0(x)B_1(x)x^{\lfloor \frac{n}{2} \rfloor} + \\ &\quad A_1(x)B_0(x)x^{\lfloor \frac{n}{2} \rfloor} + A_1(x)B_1(x)x^{2\lfloor \frac{n}{2} \rfloor}. \end{aligned}$$

We need to compute the terms `A0 * B0`, `A1 * B0 + A0 * B1` and `A1 * B1`. This can obviously be done using 4 multiplications, but here's a way of doing it with just three multiplications:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0 B_0 \\ Z &= A_1 B_1 \end{aligned}$$

U and Z are what we originally wanted and

$$A_0 B_1 + A_1 B_0 = Y - U - Z.$$

Each of the products can themselves be computed recursively. For a more detailed explanation of this approach see http://www.cse.ust.hk/~dekai/271/notes/L03/L03.pdf .

Need help? Discuss and ask questions on the forum: https://jovian.ai/forum/c/data-structures-and-algorithms-in-python/assignment-3/89

7. Come up with a correct solution for the problem. State it in plain English.

Explain the approach described above in your own words below:

1. **Divivde: Divide a problem into subproblem**
2. **Conquer: Solve each problem directly or recurssively**
3. **Combine the solutions of the subproblem into a global solution**

(add more steps if required)

Let's save and upload our work before continuing.

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/python-divide-and-conquer-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment
```

```
'https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment'
```

8. Implement the solution and test it using example inputs. Fix bugs, if any.

We are now ready to implement the solution. You may find the following functions `add`, `split` and `increase_exponent` useful.

```
def add(poly1, poly2):
    """Add two polynomials"""
    result = [0] * max(len(poly1), len(poly2))
    for i in range(len(result)):
        if i < len(poly1):
            result[i] += poly1[i]
        if i < len(poly2):
            result[i] += poly2[i]
    return result
```

```
add([1, 2, 3, 4], [0, 4, 3])
```

```
[1, 6, 6, 4]
```

```
def split(poly1, poly2):
    """Split each polynomial into two smaller polynomials"""
    mid = max(len(poly1), len(poly2)) // 2
    return  (poly1[:mid], poly1[mid:]), (poly2[:mid], poly2[mid:])
```

```
split([1, 2, 3, 4], [0, 4, 3, 6, 7, 8, 2])
```

```
(([1, 2, 3], [4]), ([0, 4, 3], [6, 7, 8, 2]))
```

```
def increase_exponent(poly, n):
    """Multiply poly1 by x^n"""
    return [0] * n + poly
```

```
increase_exponent([1, 2, 3, 4], 3)
```

```
[0, 0, 0, 1, 2, 3, 4]
```

Implement the optimized multiplication algorithm below. You may use the some or all of the helper functions defined above.

Test your solution using the empty cells below.

```
# Standardizes the polynomial (Gets rid of trailing 0s)
def standardize(lst):
    lst = list(lst) # make a copy
    if lst[len(lst)-1] == 0:
        lst.pop()
    return lst

# Finds if the polynomial is 0
def isZeroPoly(lst):
    if lst == []:
        return True
    else:
        return False

# Adds 2 polynomials ([2,3],[4,5] => [6,8])
def addPoly(lst1,lst2):
    result = []
    if len(lst1) > len(lst2):
        for x in range(0,len(lst1)):
            if len(lst2)-1 < x:
                result.append(lst1[x])
            else:
                result.append(lst1[x] + lst2[x])
    else:
        for x in range(0,len(lst2)):
            if len(lst1)-1 < x:
                result.append(lst2[x])
            else:
```

```
                result.append(lst1[x] + lst2[x])
    result = standardize(result)
    return result

# Scales the polynomial by the scale variable ((3,[1,2,3]) => [3,6,9])
def scalePoly(scale,lst):
    lst = list(lst)
    for x in range(0,len(lst)):
        lst[x] = (lst[x] * scale)
    lst = standardize(lst)
    return lst

# Gives the constant in the polynomial ([2,3,4] => [2])
def constCoef(lst):
    constant = lst[0]
    return constant

# Shifts the polynomial left ([2,3,4] => [0,2,3,4])
def shiftLeft(lst):
    newlst = []
    newlst.append(0)
    for x in range(0,len(lst)):
        newlst.append(lst[x])
    return newlst

# Shifts the polynomial right ([2,3,4] => [3,4])
def shiftRight(lst):
    #lst.remove(lst[0])
    return lst[1:]

def multiply_optimized(lst1, lst2):
    print(lst1, lst2)
    if isZeroPoly(lst1):
        return []
    else:
        return addPoly(
            multiply_optimized(shiftRight(lst1), shiftLeft(lst2)),
            scalePoly(constCoef(lst1), lst2)
        )

P = [1, 2, 3]
Q = [4, 5, 6]
multiply_optimized([2, 0, 5, 7], [3, 4, 2])
```

```
[2, 0, 5, 7] [3, 4, 2]
[0, 5, 7] [0, 3, 4, 2]
[5, 7] [0, 0, 3, 4, 2]
[7] [0, 0, 0, 3, 4, 2]
[] [0, 0, 0, 0, 3, 4, 2]
```

```
[6, 8, 19, 41, 38, 14]
```

```
multiply_optimized([2, 0, 5, 7], [3, 4, 2])
```

```
[2, 0, 5, 7] [3, 4, 2]
[0, 5, 7] [0, 3, 4, 2]
[5, 7] [0, 0, 3, 4, 2]
[7] [0, 0, 0, 3, 4, 2]
[] [0, 0, 0, 0, 3, 4, 2]
```

```
[6, 8, 19, 41, 38, 14]
```

```
multiply_basics([0], [1])
```

```
[0]
```

```

```

```

```

```

```

```
import jovian
```

```
jovian.commit()
```

```
[jovian] Updating notebook "megha-goyate/python-divide-and-conquer-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment
```

```
'https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment'
```

Make a Submission

Congrats! You have now implemented hash tables from scratch. The rest of this assignment is optional.

You can make a submission on this page: https://jovian.ai/learn/data-structures-and-algorithms-in-python/assignment/assignment-3-sorting-and-divide-conquer-practice

Submit the link to your Jovian notebook (the output of the previous cell). You can also make a direct submission by executing the following cell:

```
jovian.submit(assignment="pythondsa-assignment3")
```

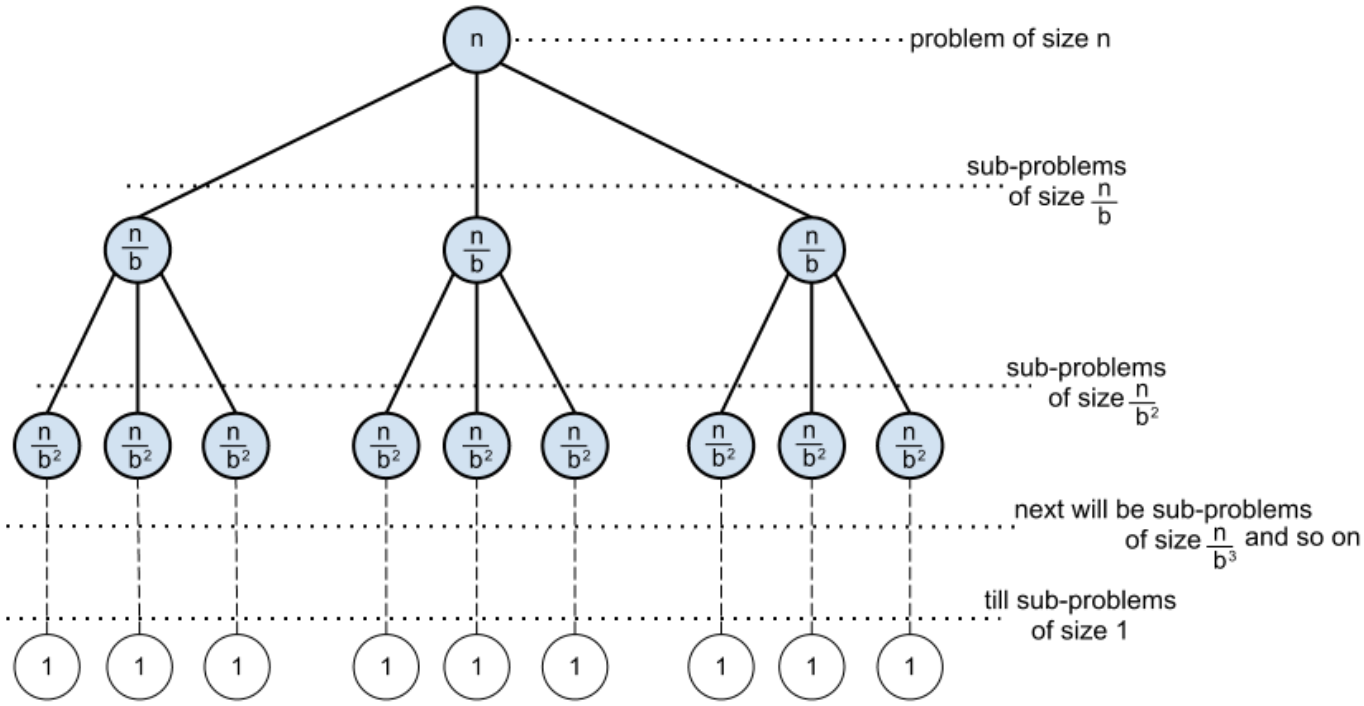
```
[jovian] Updating notebook "megha-goyate/python-divide-and-conquer-assignment" on https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/megha-goyate/python-divide-and-conquer-assignment
```

```
[jovian] Submitting assignment..
[jovian] Verify your submission at https://jovian.ai/learn/data-structures-and-algorithms-in-python/assignment/assignment-3-sorting-and-divide-conquer-practice
```

(Optional) 9. Analyze the algorithm's complexity and identify inefficiencies, if any.

Can you analyze the time and space complexity of this algorithm?

Hint: See the tree of subproblems below (source). Substitute the right values for `n` and `b` to determine the time complexity.



The height of the tree is answer to the following question:

How many times we divide problem of size n by b until we get down to problem of size 1? The other way of asking same question:

when $\frac{n}{b^x} = 1$ *[in binary tree b = 2]*

i.e. $n = b^x$ which is $\log_b n$ *[by definition of logarithm]*

```
import jovian
```

```
jovian.commit()
```