Introduction to Algorithms, 2021-2: Exam
Time limit: 180 minutes

This exam has 4 questions. Write all code in Python. For questions that do not involve code, write your answers either (a) in comments in a Python source file or (b) in a separate text file.

Important: Please read these instructions carefully.

During the exam, you **may** type your code into an editor and attempt to build and run it. You **may** refer to our quick reference page for Python, which is linked from the Programming 1 course page:

https://ksvi.mff.cuni.cz/~dingle/2021-2/prog_1/python_reference.html

You **may** also refer to any pages in the official Python documentation:

https://docs.python.org/

During the exam you **may not** visit other web pages, use Google or other search engines, refer to notes or files that you wrote before the exam, or consult in any way with other students or anyone else.

When you are done, send your solutions to adam.dingle@mff.cuni.cz. Good luck!

1.

a) Write a function `double_prime(n)` that takes an integer n ≥ 2 and returns `True` if every prime in the prime factorization of n occurs **at least** twice. For example:

- `double_prime(11)` = `False`, since 11 is prime
- `double_prime(72)` = `True`, since 72 = 2 · 2 · 2 · 3 · 3
- `double_prime(300)` = `False`, since 300 = 2 · 2 · 3 · 5 · 5, and the prime 3 appears only once in this factorization

b) What is your function's **worst-case** running time as a function of n? Assume that all primitive arithmetic operations (+, -, *, //, %) always run in O(1).

If your answer to (b) is O(n), your function is too slow – please optimize it to make it faster if you can.

~ Exam continues on next page ~

2.

In the lecture we learned about the **mergesort** algorithm.

You are given a function `merge(a, lo, mid, hi)` that can merge two adjacent sorted regions in an array a. (You do **not need to write** this function - assume that it is given to you.)

Before calling merge(), the regions `a[lo:mid]` and `a[mid:hi]` must both be sorted. It will merge them into a single sorted region `a[lo:hi]`. merge() always runs in O(N), where N = hi - lo.

For example:

```
a = [3, 5, 8, 10, 12,   6, 7, 11, 15, 18]  # 10 elements in 2 sorted regions

merge(a, 0, 5, 10)   # merge a[0:5] with a[5:10]
print(a)             # prints [3, 5, 6, 7, 8, 10, 11, 12, 15, 18]
```

a) Using this helper function, write a recursive function `mergesort(a)` that sorts an array.

b) Let T(N) be the time to run mergesort() on an array with N elements. Write a **recurrence relation** that expresses T(N) recursively, using big-O notation.

c) What is the best-case and worst-case big-O running time of mergesort(), as a function of N?

d) Suppose that we run mergesort() on an array that has N = 2^k elements. Exactly how many times will the merge() function be called as the sort runs?

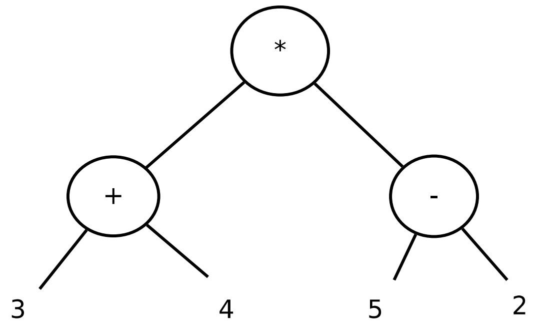
~ Exam continues on next page ~

3. Consider **arithmetic expressions** formed from integers and the operations '+', '-', '*', and '/', where '/' represents integer division. For example, ((3 + 4) * (5 - 2)) is one such expression.

In the lecture, we saw that we may represent an arithmetic expression using a **tree** of nodes in Python. We may use this class:

```
class OpExpr:
    def __init__(self, op, left, right):
        self.op = op     # operator character, e.g. '+'
        self.left = left
        self.right = right
```

In an expression tree, every interior node will be an instance of this class. Every leaf will be a Python integer. For example, the expression ((3 + 4) * (5 - 2)) is represented by this tree:



We may build this tree in Python like this:

```
>>> l = OpExpr('+', 3, 4)
>>> r = OpExpr('-', 5, 2)
>>> my_tree = OpExpr('*', l, r)
```

a) Write a function to_postfix(t) that takes the root of an expression tree and returns a string representing the tree in **postfix notation**, with a single space between numbers and operators:

```
>>> to_postfix(my_tree)
'3 4 + 5 2 - *'
```

b) Write a function eval(t) that takes the root of an expression tree and **evaluates** the expression, returning its value:

```
>>> eval(my_tree)
21
```

c) Write a function describe(t) that takes the root of an expression tree and returns a triple (n, a, b), where n is the **number of operations** in the tree, a is the **smallest integer** in the expression and b is the **largest integer** in the expression:

```
>>> describe(my_tree)
(3, 2, 5)
```

Hint: A node in the tree may be either an instance of OpExpr (an interior node) or a Python integer (a leaf). You can distinguish these cases using the built-in isinstance() function:

```
>>> isinstance(3, int)
True
>>> isinstance(my_tree, int)
False
>>> isinstance(my_tree, OpExpr)
True
```

~ Exam continues on next page ~

4.

The first line of standard input will contain two lowercase words W and X on a single line, separated by a space. The rest of the input will contain a list L of lowercase words, one per line. All words in the input will have the same length.

Write a program that reads the input and determines the shortest number of steps required to transform the word W into the word X, where (a) you may change only a **single letter** in each step and (b) all intermediate words in the sequence must appear in the list L. Print the resulting sequence of words, starting with W and ending with X. If more than one shortest sequence is possible, you may print any of them. If no such sequence is possible, print 'no path'.

Sample input:

```
cold warm
bold
bore
cord
core
hold
hole
load
lord
word
wore
worm
```

Output:

```
cold
cord
word
worm
warm
```

Hints:

This is a state space search problem. Use a graph search algorithm to find the shortest path. As you search, keep a dictionary that maps each word you encounter to the previous word in the search space. When you find the target word X, you can use this dictionary to construct a path to print out.