NATIONAL UNIVERSITY OF SINGAPORE

Department of Computer Science, School of Computing

IT5001—Software Development Fundamentals

Academic Year 2023/2024, Semester 2

PROBLEM SET 7

DIVIDE-AND-CONQUER & DYNAMIC PROGRAMMING

These exercises are optional and ungraded, and for your own practice only. Contact `yongqi@nus.edu.sg` for queries.

SHORT PROGRAMMING EXERCISES

Question 1. Implement the algorithms you've learnt in the following manner:

1. Implement linear search using recursion, and map, filter and reduce.
2. Implement bubble sort recursively.
3. The implementation of quicksort uses the first element of the list as the pivot. Try creating an implementation that uses the median-of-three as the pivot, which takes the median of the first, middle and last elements as the pivot.

Then, try to implement all the algorithms without looking at the slides.

Question 2. The asymptotic space complexity of an algorithm is concerned with the amount of extra space needed by the algorithm (for example, merge sort's space complexity is $O(n)$ due to the merge step). What is the time and space complexity of the following algorithm?

```
1: function SumPowOdd(ls, n)
2:     r ← []
3:     for each i ∈ ls do
4:         if i is odd then
5:             append i^n to r
6:     return ∑_{i=0}^{|r|-1} r[i]
```

Question 3. The following are lists of numbers that are midway through the sorting process. Assume that the list is meant to be sorted in ascending order. Among the algorithms that we learnt, which could have been used for each list?

1. `[4, 5, 2, 3, 7, 1, 6, 0, 8, 9]`
2. `[2, 4, 5, 8, 9, 0, 1, 3, 6, 7]`
3. `[5, 2, 1, 6, 0, 4, 3, 7, 9, 8]`
4. `[0, 1, 2, 3, 4, 9, 8, 6, 5, 7]`

Question 4. Due to a bug in Coursemology, the digits of the decimal system got reshuffled. For in-

stance, each 0 changed to 2, each 1 got changed to 3, each 2 got changed to 0. If the correct number is 021, the system shows 203; students grades have therefore become completely wrong, and everyone is up in arms! While the Coursemology team is working on a fix, you decide to obtain the correct ranking of students so that you can, at the very least, assign the correct letter grade to each student first.

Given the numbers that the program needs to sort, and the mapping of numbers given as a list (if `mapping[0] = 9`, then the 0s got changed to 9s), return a list of the jumbled numbers sorted by their correct decimal values, ascending. If multiple mapped values are equal, the values returned should be in the original order they were presented.

Note that the numbers to sort are presented in string-form, because 000 may be different from 00 depending on the mapping. Thus, all digits must be preserved.

Question 5. [Stolen from CS2040 Problem Set 2 AY18/19 Sem 1] As we all know, the Acme Corporation is a world leader in making people smile. (Motto: For fifty years, the leader in creative mayhem!) The Acme Corporation is well-known for its wide selection of anvils and birdseed, along with specialty items such as dehydrated boulders and earthquake pills. However, in recent days, Acme stock has been depressed, and their inventors have been hard at work coming up with new ways to make people happy. Their most recent invention is a robotic clown named Herbert. He is a fully automated clown, well equipped with all of your favorite Acme gizmos, and he can be hired to perform at your party! While Herbert is an excellent clown, he is not so good at math. (He was programmed with a very funny joke about how 1+1 = 3, and ever since he has had great difficulties.) Your job, in this problem, is to help Herbert with the financial aspects of his job.

Herbert is paid by the minute. Sometimes, he is hired for a very short performance (perhaps, only 1 minute long), and sometimes he is hired for a long engagement that may last for weeks. Herbert's salary, however, is very complicated: he is paid different amounts for different lengths of time. (An additional complication is that Herbert is payed in SmileBucks, which Herbert then may exchange for spare parts and electricity. Currency conversion and bartering techniques are outside the scope of this problem.)

Herbert has given you an employment log of his to analyze. The log is very long, containing data from the last several months. Each line of the log represents one minute of work. Hence a job that takes ten minutes is represented by ten lines in the log.

Each line in the log consists of two components: the name of the employer, and the total amount of SmileBucks (SB) that Herbert has been paid for the job, up to and including that minute. For example, consider the following two jobs where Humperdink hires Herbert for 5 minutes and Hortensia hires Herbert for 2 minutes:

```
Humperdink:7
Humperdink:10
```



```
Humperdink:12
Humperdink:14
Humperdink:16
Hortensia:10
Hortensia:11
```

Here, we see that Herbert was paid 7 SB by Humperdink for the first minute. He was paid an additional 3 SB for the second minute, yielding a total of 10 SB. In total, for the five minutes, he was paid 16 SB. Herbert then was paid 11 SB by Hortensia. You can assume that each employer only hires Herbert once, i.e., all the names of the employers are unique.

Your task is to define the `calculate_salary` function that takes in a list of records where each record is in a 2-tuple consisting of the name of the employer, and the cumulative amount paid, like in the above example, then returns the total amount of money received by Herbert. You may define other auxilliary functions to help you.

Note that the naive solution is to iterate through each record and add the highest salary corresponding to each name to the total. However, this is a very slow solution because some employers can hire Herbert for a very long time!

Question 6. Suppose you are given a continuous function $f(x)$, and you want find one root of f, i.e. to solve for when $f(x) = 0$. Further suppose that you are given a and b such that $f(a) \leq 0$ and $f(b) \geq 0$. Write a function `root(f, a, b)` that returns some x such that $f(x) \approx 0$. You can return some $\pm\epsilon$ where ϵ is a small number. Your function should run in $O(\log |a - b|)$.

Question 7. Work on three Kattis exercises. You must pass the test cases on Kattis—that means that your algorithms must not only be correct, they must be efficient as well.

Array of Discord: https://open.kattis.com/problems/arrayofdiscord
A Favourable Ending: https://open.kattis.com/problems/favourable
Accounting: https://open.kattis.com/problems/bokforing

JOHN WICK: CHAPTER 5 - VIOLENT ENDS

This is a contribution by former TA and M.Comp. (General Track) student, Shi Zhijian. Template files can be found after this section.

There is a big big name in the hitman world, ***John Wick***. People know him because of his beloved dog and his scary pencil.

Now times have changed, even killers needs efficiency. John now wants a digital butler ***J.A.R.V.I.S*** (Yes, exactly the AI system designed by Tony Stark) to help him improve his pre-task preparation.

Equipment Management

John usually use a tactical harness, which he uses to carry all his pistols, knives, magzines, etc. He always put his weapons randomly after he's done using them, which causes him a lot of trouble. Now he wants Jarvis to design an algorithm to group his gadgets efficiently.

A circular array h is used to represent the harness (a circular array is defined as an array where the first element and the last element are considered to be adjacent). For this, we will use the `Belt` class (see template file).

Slots with equipment are defined as 1 and empty slots are 0. For example, for some belt `h = Belt([1,0,1,1,0])`, the first, third and fourth slots have equipment in them, and the rest are empty. John can take one equipment out from its original slot and slot it to any empty slot.

Given a circular array `h` as an instance of `Belt`, return the minimum number of swaps required to group all weapons together at any location. Examples with explanations follow.

Example 1.

```
>>> h = Belt([0, 1, 0, 1, 1, 0, 0])
>>> min_swaps(h)
1
```

Explanation: Here are a few of the ways to group all the weapons together:
`Belt([0,0,1,1,1,0,0])` using 1 swap.
`Belt([0,1,1,1,0,0,0])` using 1 swap.
`Belt([1,1,0,0,0,0,1])` using 2 swaps (using the circular property of the belt).
There is no way to group all weapons together with 0 swaps.
Thus, the minimum number of swaps required is 1.

Example 2.


```
>>> h = Belt([1,1,0,0,1])
>>> min_swaps(h)
0
```

Explanation: All the weapons are already grouped together due to the circular property of the array. Thus, the minimum number of swaps required is 0.

How to use the Belt. John has kindly provided us mere civilians with a manual on how to use his tactical belt.

The belt constructor takes as a sole argument a regular Python list. Beyond that, it essentially behaves very similarly to a list. The main difference is that if the belt has `i` slots, then `belt[i + j]` is the same as `belt[j]` for some `j`, due to the circular property of the belt (as explained earlier, the first and last element are adjacent to each other).

Example uses of the belt follows:

```
>>> b = Belt([1, 1, 0, 0, 1])
>>> b[0]
1
>>> b[5]
1
>>> b[6] = 0
>>> b
Belt([1, 0, 0, 0, 1])
>>> for i in b:
...     print(i)
1
1
0
0
1
>>> b.pop()
1
>>> b
Belt([1, 1, 0, 0])
>>> b[3:6]
Belt([0, 1, 1])
```

Time management

As a legendary hitman, John has been assigned numerous "impossible tasks" by his supervisor. Unfortunately, many of them overlap in time. John wants Jarvis to manage his time and help him organize these missions.

Missions are represented using the `Mission` class, consisting of a start and end date (inclusive). Dates are represented using the `Date` class, with its internal representation being a date in YYYYMMDD format, for example, 20220228 stands for 28 Feb 2022. A mission's start date is always strictly smaller than it's end date.

Given a list of missions, find the number of missions that need to be removed to make the rest of the missions non-overlapping.

Example 1.

```
>>> missions = [Mission(Date(20220101), Date(20220102)),
                Mission(Date(20220102), Date(20220103)),
                Mission(Date(20220103), Date(20220104)),
                Mission(Date(20220101), Date(20220103))]
>>> remove_overlaps(missions)
1
```

Explanation: The mission from 1 Jan 2022 to 3 Jan 2022 can be removed and the rest of the missions are non-overlapping.

Example 2

```
>>> missions = [Mission(Date(20220201), Date(20220202)),
                Mission(Date(20220302), Date(20220303))]
>>> remove_overlaps(missions)
0
```

Explanation: You don't need to remove any of the missions since they're already non-overlapping.

How to use the Date and Mission classes. Construct a date by passing in the date into its constructor in the form of YYYYMMDD. Dates can be compared; if for two dates a and b, a is earlier than b, then $a < b$. Missions have start and end dates. Example uses follow:

```
>>> a = Date(20220101)
>>> b = Date(20220203)
>>> m = Mission(a, b)
```



```
>>> m.start
1 Jan 2022
>>> m.end
3 Feb 2022
>>> m.start < m.end
True
```

Reward management

The High Table, which is the supreme authority of the hitman world, awards a certain amount of credits once a task is completed. To ensure quality of his "service", John does not want to do two tasks in a row.

Given a list of integers `rewards` representing the amount of credits provided for each task. e.g. `rewards = [2,7,9,3,1]`, return the maximum amount of credits John can get without performing two tasks consecutively.

Example 1.

```
>>> rewards = [1,2,3,1]
>>> max_reward(rewards)
4
```

Explanation: Perform task 1 (credits = 1) and then task 3 (credits = 3).
Total amount John can get: 1 + 3 = 4.

Example 2.

```
>>> rewards = [2,7,9,3,1]
>>> max_reward(rewards)
12
```

Explanation: Perform task 1 (credits = 2), task 3 (credits = 9) and task 5 (credits = 1).
Total amount John can get = 2 + 9 + 1 = 12.

TEMPLATE FILES

eqp_mgmt.py:

```
# Complete the min_swaps function below
def min_swaps(h):
    pass

# For testing, just run the python file
# You do not need to paste these into Coursemology
class Belt:
    """
    The Belt class acts as a circular list.
    """
    def __init__(self, ls):
        self._ls = ls

    def __iter__(self):
        """
        You can use a belt in a for loop.
        """
        return iter(self._ls)

    def __len__(self):
        return len(self._ls)

    def __getitem__(self, key):
        """
        You can get an element in the belt or a slice
        of the belt.
        """
        if isinstance(key, int):
            key %= len(self)
            return self._ls[key]
        res = []
        step = 1 if key.step is None else key.step
        start = key.start if key.start is not None else \
                0 if step > 0 else \
                len(self) - 1
        stop = key.stop if key.stop is not None else \
                len(self) if step > 0 else \
                -1
        for i in range(start, stop, step):
            res.append(self[i])
        return Belt(res)

    def append(self, item):
        self._ls.append(item)
```

```
    def extend(self, it):
        self._ls.extend(it)
    def pop(self, key = -1):
        key %= len(self)
        return self._ls.pop(key)
    def remove(self, item):
        self._ls.remove(item)
    def __contains__(self, item):
        return item in self._ls
    def count(self, item):
        return self._ls.count(item)
    def __repr__(self):
        return str(self)
    def __str__(self):
        return f'Belt({str(self._ls)})'

def test():
    print('=== Example 1 ===')
    input1 = Belt([0, 1, 0, 1, 1, 0, 0])
    print(f'input: {input1}')
    print('expected: 1')
    print(f'your output: {min_swaps(input1)}\n')
    print('=== Example 2 ===')
    input2 = Belt([1, 1, 0, 0, 1])
    print(f'input: {input2}')
    print('expected: 0')
    print(f'your output: {min_swaps(input2)}')

if __name__ == '__main__':
    test()
```

time_mgmt.py:

```
# Complete the remove_overlaps function below
def remove_overlaps(missions):
    pass

# For testing, just run the python file
# There is no need to paste the following into Coursemology
class Date:
    MONTHS = [None, 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
        'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    def __init__(self, yyyymmdd):
        self._int = yyyymmdd
    def __repr__(self):
        return str(self)
    def __str__(self):
        return f'{self._int % 100} '\
```

```
            f'{Date.MONTHS[self._int // 100 % 100]}'\
            f' {self._int // 10000}'
    def __eq__(self, obj):
        return isinstance(obj, Date) and \
                self._int == obj._int
    def __lt__(self, obj):
        if not isinstance(obj, Date):
            raise TypeError
        return self._int < obj._int
    def __gt__(self, obj):
        if not isinstance(obj, Date):
            raise TypeError
        return self._int > obj._int
    def __ne__(self, obj):
        return not (self == obj)
    def __le__(self, obj):
        return self == obj or self < obj
    def __ge__(self, obj):
        return self == obj or self > obj

class Mission:
    def __init__(self, start, end):
        self.start = start
        self.end = end
    def __getitem__(self, key):
        return [self.start, self.end][key]
    def __str__(self):
        return f'[{str(self.start)}, {str(self.end)}]'
    def __repr__(self):
        return str(self)

def test():
    print('=== Example 1 ===')
    input1 = [Mission(Date(20220101), Date(20220102)),
              Mission(Date(20220102), Date(20220103)),
              Mission(Date(20220103), Date(20220104)),
              Mission(Date(20220101), Date(20220103))]
    print(f'input: {input1}')
    print('expected: 1')
    print(f'your output: {remove_overlaps(input1)}\n')
    print('=== Example 2 ===')
    input2 = [Mission(Date(20220201), Date(20220202)),
              Mission(Date(20220302), Date(20220303))]
    print(f'input: {input2}')
    print('expected: 0')
    print(f'your output: {remove_overlaps(input2)}')

if __name__ == '__main__':
    test()
```

rwd_mgmt.py:

```python
# Complete the max_reward function below
def max_reward(task_rewards):
    pass

# For testing, just run the python file
# There is no need to paste the following into Coursemology
def test():
    print('=== Example 1 ===')
    input1 = [1, 2, 3, 1]
    print(f'input: {input1}')
    print('expected: 4')
    print(f'your output: {max_reward(input1)}\n')
    print('=== Example 2 ===')
    input2 = [2, 7, 9, 3, 1]
    print(f'input: {input2}')
    print('expected: 12')
    print(f'your output: {max_reward(input2)}')

if __name__ == '__main__':
    test()
```

SOLUTIONS

Question 1. Only solutions to the first portion will be provided; for implementations of the other algorithms you can just refer to the lecture slides or online materials.

(1.1) Both are self-explanatory. Recursion:

```
def linear_search(ls, key, i = 0):
    if i >= len(ls):
        return None
    if ls[i] == key:
        return i
    return linear_search(ls, key, i + 1)
```

Using map, filter and reduce

```
from functools import reduce
def linear_search(ls, key):
    def check(a, b):
        if isinstance(a, int):
            return a
        return b
    return reduce(check, map(lambda i: i if ls[i] == key else None, range(len(ls))))
```

(1.2) Similarly, self-explanatory

```
def swap(ls, i, j):
    ls[i], ls[j] = ls[j], ls[i]

def is_not_sorted(ls):
    return any(ls[i] > ls[i + 1] for i in range(len(ls) - 1))

def bubble(ls, n):
    if n >= len(ls) - 1: return
    if ls[n] > ls[n + 1]: swap(ls, n, n + 1)
    bubble(ls, n + 1)

def bubblesort(ls):
    if is_not_sorted(ls):
        bubble(ls, 0)
        bubblesort(ls)
```

(1.3) Instead of assigning the pivot to be the start of the list, just use the median-of-three strategy to obtain the index of the pivot, then swap that element to the start so that you can quicksort as per normal.

```python
def median_of_three(ls, start, end):
    # If only two possible elements, let the median be the start
    if end - start <= 2:
        return start
    # Find mid
    mid = (end + start - 1) // 2
    # Find median of start, mid and end
    print(start, mid, end - 1)
    return sorted((start, mid, end - 1), key=lambda x: ls[x])[1]

def partition(ls, start, end):
    # Get the median of three
    pivot_idx = median_of_three(ls, start, end)
    # Move that pivot element to the front
    ls[pivot_idx], ls[start] = ls[start], ls[pivot_idx]
    # Continue as per normal
    pivot = ls[start]
    partition_idx = start + 1
    for i in range(start + 1, end):
        if ls[i] <= pivot:
            ls[i], ls[partition_idx] = ls[partition_idx], ls[i]
            partition_idx += 1
    ls[start], ls[partition_idx - 1] = ls[partition_idx - 1], ls[start]
    return partition_idx - 1

# Same as before
def quicksort(ls, start = 0, end = None):
    end = len(ls) if end is None else end
    if start + 1 >= end: return
    partition_idx = partition(ls, start, end)
    quicksort(ls, start, partition_idx)
    quicksort(ls, partition_idx + 1, end)
```

Question 2. Assume line 2 runs in some constant time. Then, we observe that the loop runs for $|ls|$ iterations. In each iteration, we append i^n to r. Computing i^n takes $O(\log n)$ time, assuming that multiplication is of some constant time. Therefore, the loop runs in $O(|ls| \log n)$ time. The return statement runs in $O(|ls|)$ time since we are summing the elements in the list r which in the worst case, contains all the elements in ls, so the overall time complexity of the algorithm is in $O(|ls| \log n + |ls|) = O(|ls| \log n)$. The space complexity is obviously in $O(|ls|)$ to accomodate list r.

Question 3.

(3.1) Bubblesort, Quicksort and Selection Sort. Bubblesort because the last two elements have been bubbled. Quicksort is also possible because if the pivot chosen is the maximum element, quicksort is just a glorified bubblesort. Selection sort is also possible because we could have sorted the maximum elements first, and pushed them all to the right, instead of our usual selection scheme of starting with the minimum elements and pushing them to the left.

(3.2) Mergesort. The left half and right halves of the list are sorted.

(3.3) Quicksort. The pivot 7 was chosen and the list was partitioned based on this pivot.

(3.4) Bubblesort, Quicksort and Selection Sort. Bubblesort because we could have bubbled from right to left and bubbled the minimum elements first, instead of the usual bubbling scheme of bubbling the maximum elements from left to right. Quicksort is also possible because the worst pivots of 0, 1, 2, 3 and 4 could have been chosen first. Selection sort is also obviously possible.

Question 4. The idea is to sort as per normal, but to use the sorting key which maps each number in the list to its intended uncorrupted value. You may use any other sorting function like the ones we've learnt; but for simplicity's sake we shall use python's built-in sorting function.

```python
def strange_sort(numbers, mapping):
    '''
    Sorts the numbers by using the mapping as the ordering of the elements.
    Numbers is a list of numbers, but each number is represented using strings
    (this is because '000' might be different to '00' after the mapping).
    '''
    # Base case.
    if not numbers:
        return numbers

    # Convert mapping into a dictionary for fast mapping of digits.
    mapping = { str(value): str(index) for index, value in enumerate(mapping) }

    # This function takes the jumbled number, and uses mapping to restore it
    # to what it was meant to be.
    def restore_original(n: str) -> int:
        return int(''.join(map(lambda i: mapping[i], n)))

    # Sort the numbers using the original number as the key.
    return sorted(numbers, key = restore_original)
```

Question 5. The solution is something like binary search. Let the searching region be the entire list first. Let the first element in the searching region be the name to look for, i.e., suppose the name in the leftmost record in the list is 'Alfred'. Then, we want to look for the record with the highest cumulative payout with the name 'Alfred' as well. Conduct a binary search for this record, and once done, move one step to the right from this record for the next name, and continue from there.

The following pseudocode and Python code should explain.

1: **function** CALCULATESALARY(*records*, *name*, *searchLow*, *searchHigh*)
2: Let the default parameter values be $name \in records[searchLow]$, $searchLow \leftarrow 0$, $searchHigh \leftarrow |records| - 1$
3: **if** $name \in records[-1]$ **then**
4: **return** $salary \in records[-1]$
5: $mid \leftarrow \frac{searchLow+searchHigh}{2}$
6: **if** $name \in records[mid]$ and $name \notin records[mid+1]$ **then**
7: $maxSalary \leftarrow salary \in records[mid]$
8: **return** $maxSalary +$ CALCULATESALARY$(records, name \in records[mid+1], mid+1)$
9: **if** $name \notin records[mid]$ **then**
10: **return** CALCULATESALARY$(records, name, searchLow, mid-1)$
11: **return** CALCULATESALARY$(records, name, mid+1, searchHigh)$

```python
def calculate_salary(records,
        name: str = None,
        search_low: int = 0,
        search_high: int = None) -> int:
    '''
    Overall program logic:
    1) choose the first element in the searching region as the name to look for
    2) binary search the record that contains the highest salary corresponding
       to name
    3) get the salary from 2) then continue with another name, until you've
       reached the end of the list.
    '''
    # Initialize defaults
    if name == None:
        # Set the default name to be the name in the first record
        name = records[search_low][0]

    if search_high == None:
        # Set the default rightmost index in the searching region to be
        # the end of the list
        search_high = len(records) - 1

    # Base case: name to look for matches the last element.
    if name == records[-1][0]:
        return records[-1][1]

    # See if the midpoint might contain the highest salary corresponding
    # to the name.
    mid = (search_low + search_high) // 2

    if records[mid][0] == name and records[mid + 1][0] != name:
        # This means that the record at index mid contains the highest
        # salary corresponding to name
        res = records[mid][1]
```

```
        # You still need to look at the next fella.
        return res + calculate_salary(records, records[mid + 1][0], mid + 1)

    if records[mid][0] != name:
        # Too far right. Try reducing mid
        return calculate_salary(records, name, search_low, mid - 1)

    # Too far left. Try increasing mid
    return calculate_salary(records, name, mid + 1, search_high)
```

Question 6. If $f(a) \leq 0$ and $f(b) \geq 0$, then clearly there must exist some $a \leq x \leq b$ where $f(x) = 0$ because f is continuous. We can therefore do a binary search to obtain this x. This is called the bisection method.

1: **function** ROOT(f, a, b)
2: **if** $f(a) = 0$ **then return** a
3: **if** $f(b) = 0$ **then return** b
4: $x \leftarrow \frac{a+b}{2}$
5: **if** $|f(x)| \leq \epsilon$ **then return** x
6: **if** $f(x) < 0$ **then return** ROOT(f, x, b)
7: **return** ROOT(f, a, x)

```
def root(f, a, b):
    if f(a) == 0: return a
    if f(b) == 0: return b
    x = (a + b) / 2
    if abs(f(x)) <= 0.00001:
        return x
    if f(x) < 0: return root(f, x, b)
    return root(f, a, x)
```

Question 7. **Array of Discord**. Look through each pair. If the leading digit in the left number can be replaced with 9 such that the pair is unsorted, then we have a solution. Likewise, if the leading digit in the right number can be replaced with 0 or 1 such that the pair is unsorted, then we have a solution. Otherwise, it is not possible.

```
def solve(arr):
    arr = arr[:]
    for i in range(len(arr) - 1):
        # if already unsorted, return arr
        if arr[i] > arr[i + 1]:
            return arr
        solution = replace(str(arr[i]), str(arr[i + 1]))
        if solution:
```

```
            arr[i:i + 2] = map(int, solution)
            return ' '.join(map(str, arr))
    return 'impossible'

def replace(x, y):
    if len(x) < len(y):
        return False
    smallest_digit = '1' if len(x) > 1 else '0'
    # Try replacing leading digit in x with 9
    if y[0] != '9' or x[1:] > y[1:]:
        return '9' + x[1:], y
    # Try replacing leading digit in y with 0 or 1
    if x[0] != smallest_digit:
        return x, smallest_digit + y[1:]
    return False

def main():
    input()
    print(solve(list(map(int, input().split()))))

if __name__ == '__main__':
    main()
```

A Favourable Ending. My solution for this uses top-down dynamic programming with memoization. Suppose we can get to *favourably* from pages a, b and c. Then the number of ways to get to *favourably* is the sum of the number of ways to get to a, b and c.

```
def ways_to_get_there(elem, prevs, memo):
    if elem == 1:
        return 1
    if elem in memo:
        return memo[elem]
    memo[elem] = sum([ways_to_get_there(i, prevs, memo) for i in prevs[elem]])
    return memo[elem]

def favourable(book: list) -> int:
    # Build prevs
    prevs = {}
    for section in book:
        prev = section[0]
        for next_section in section[1:]:
            if next_section not in prevs:
                prevs[next_section] = []
            prevs[next_section].append(prev)
    # return prevs
    return 0 if 'favourably' not in prevs else \
            ways_to_get_there('favourably', prevs, {})
```

```
# Just for input/output
def main():
    for _ in range(int(input())):
        print(favourable([
            list(map(lambda x: x if x == 'favourably' or x == 'catastrophically'
            else int(x), input().split(' ')))
            for n in range(int(input()))]))

if __name__ == '__main__':
    main()
```

Accounting. Maintain a dictionary containing all the new values of particular people are when SET commands happen. Maintain a single variable containing the amount of money that everyone else has.

```
def run(n, q):
    sets = {}
    everyone = 0
    for _ in range(q):
        i = input().split()
        if i[0] == 'SET':
            person, val = map(int, i[1:])
            sets[person] = val
        elif i[0] == 'PRINT':
            person = int(i[1])
            if person in sets:
                print(sets[person])
            else:
                print(everyone)
        else:
            everyone = int(i[1])
            sets = {}

def main():
    n, q = tuple(map(int, input().split()))
    run(n, q)

if __name__ == '__main__':
    main()
```

John Wick. **Equipment Management**.

1. The length of consecutive 1s is the number of 1s in the array.
2. We use a moving window, which has a size of the number of 1s in the entire array, to record the maximum number of 1s within that window.
3. The minimum swap is the number of 1s in the array minus the maximum number of 1s within the window.

```python
def min_swaps(h):
    window = sum(h)
    curr = sum(h[:window])
    res = window - curr
    for i in range(len(h)):
        res = min(res, window - curr)
        curr += h[i + window] - h[i]
    return res
```

Time Management.

1. Get the minimum number of missions to remove is equivalent to get the maximum number of non-overlapping missions.
2. Missions with smaller end date should be checked first.
3. Start point of current mission should be greater than end point of the last selected mission.

```python
def remove_overlaps(missions):
    missions.sort(key = lambda x: x.end)
    valid_missions = 0
    last_mission = None
    for mission in missions:
        if last_mission and mission.start < last_mission.end:
            continue
        valid_missions += 1
        last_mission = mission
    return len(missions) - valid_missions
```

Rewards Management.

1. There are 2 options for i^{th} mission, *perform* or *not perform*.
2. The maximum reward of "*not perform* i^{th} mission" is the maximum rewards to be obtained for all missions up to and including the $(i-1)^{\text{th}}$ mission.
3. The maximum reward of "*perform* i^{th} mission" is the maximum rewards to be obtained for all missions up to and including the $(i-2)^{\text{th}}$ mission, plus the reward of the i^{th} mission.

```python
def max_reward(task_rewards):
    n = len(task_rewards)
    table = [0] * n
    table[0] = task_rewards[0]
    for k in range(1, n):
        table[k] = max(table[k - 1], task_rewards[k] + table[k - 2])
    return table[-1]
```

– End of Problem Set 7 –