T1

思路与证明：题目让我们找到一个序列，使得按照这个序列介绍任务，可以让所有人完成任务的时间最短。利用贪心算法，我们想要让士兵们尽早开始任务，减少因简报浪费的时间，因此将士兵按照简报时间进行升序排列，优先对简报时间最短的士兵进行简报，这样可以得到最短响应时间，大家都可以尽可能早地投入到任务中去。但是如果有一个士兵，他的简报时间和任务时间都很长，导致他很晚开始，最后大家都在等他的任务结束，这种情况下确实不是最优解。

伪代码：

def minimize\_time(n, soldiers):

soldiers.sort(key=lambda x: x[0]) # 按照简报时间升序排序

total\_time = 0

max\_completion\_time = 0

for soldier in soldiers:

briefing\_time, completion\_time = soldier

total\_time += briefing\_time

max\_completion\_time = max(max\_completion\_time, total\_time + completion\_time)

return max\_completion\_time

def main():

case\_number = 1

while True:

n = int(input())

if n == 0:

break

soldiers = []

for \_ in range(n):

b, j = map(int, input().split())

soldiers.append((b, j))

result = minimize\_time(n, soldiers)

print(f"Case {case\_number}: {result}")

case\_number += 1

复杂度：对N名士兵进行排序，时间复杂度O(NlogN) 排序后对N名士兵的时间进行遍历，时间复杂度O(N)，总时间复杂度O(NlogN).只用到常数个数组存储排序结果、简报时间、任务时间等，空间复杂度O(N)

T2

思路与证明：和上一章动态规划比起来，这里的问题情形要简单一些，因为保证了输入的DNA序列都是等长的，不会有丢失和增添的情况，只需要比较对应位置的序列即可，这样也很容易得到贪心算法：想要得到最后共识误差最小的序列，只需要每个位置的误差最小即可，换句话说，就是找每个位置出现频率最高的字母。算法的正确性也是显而易见的。

伪代码：

def hamming\_distance(s1, s2):

return sum(c1 != c2 for c1, c2 in zip(s1, s2))

def consensus\_string(strings):

n = len(strings[0])

consensus = ""

for i in range(n):

counts = {'A': 0, 'C': 0, 'G': 0, 'T': 0} //这里注意用字典序排序

for s in strings:

counts[s[i]] += 1

consensus += max(counts, key=counts.get)

return consensus

def consensus\_error(strings, consensus):

return sum(hamming\_distance(consensus, s) for s in strings)

def main():

T = int(input("Enter the number of test cases: "))

for \_ in range(T):

m, n = map(int, input().split())

sequences = [input().strip() for \_ in range(m)]

cons\_str = consensus\_string(sequences)

cons\_err = consensus\_error(sequences, cons\_str)

print(cons\_str)

print(cons\_err)

复杂度：要确定最后输出的序列每个位置的字母，需要统计每个位置每个字母出现的频率，时间复杂度为O(mn)；需要将输入和输出的序列存下来，空间复杂度为O(mn)

T3

思路和证明：一共有n个敌人，d天，输入的格式为d行n列，也就是每一行代表了这一天里对手的出席情况。根据规则必须全员到场才能打败Arya，因此只需要检查这一行里面有没有0即可。

伪代码：

def max\_consecutive\_days(n, d, schedule):

consecutive\_days = 0

current\_consecutive\_days = 0

for day in schedule:

if '0' in day:

# If at least one opponent is absent, Arya beats all present opponents

current\_consecutive\_days += 1

else:

# If all opponents are present, reset the consecutive days counter

consecutive\_days = max(consecutive\_days, current\_consecutive\_days)

current\_consecutive\_days = 0

return max(consecutive\_days, current\_consecutive\_days)

def main():

n, d = map(int, input().split())

schedule = [input().strip() for \_ in range(d)]

result = max\_consecutive\_days(n, d, schedule)

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

复杂度：将输入的矩阵遍历一遍即可得到答案，时间与空间复杂度都是O(nd)

T4

思路与证明：想要得到符合要求的最小的数字，有这样几项规则：位数越小越好；位数相同时高位越小越好，也就是说，低位越大越好。因此贪心算法将目前可用的最大的数放在最低位即可。

伪代码：

def find\_minimum\_varied\_number(s):

# 初始化结果为一个空字符串

result = ""

# 从高位（9）开始递减，选择数字直到满足条件

current\_digit = 9

while s > 0:

# 如果当前数字未被使用，将其添加到结果中，并减去其值

if s >= current\_digit:

result = str(current\_digit) + result

s -= current\_digit

current\_digit -= 1

else:

result = str(s) + result

s = 0

return result

def main():

t = int(input()) # 读取测试用例的数量

for \_ in range(t):

s = int(input()) # 读取每个测试用例的 s 值

result = find\_minimum\_varied\_number(s)

print(result)

复杂度：由于题目限制各个位置上的数字不能相同，因此输入不会超过45，循环不超过9次，时间和空间复杂度都是O(1)，如果考虑到测试用例的数量，总复杂度O(t).

T5

思路与证明：不管如何操作，整个数组的乘积是固定的，并且，操作的次数是无限的，因此我们要做的就是如何分配这个乘积不同的乘数，使得乘数的和最大。对于一对整数xy来说，将乘数分为1和xy加和最大，利用贪心算法，当乘数有多个时，也是将乘数分为若干个1和一个大数最划算。

伪代码：

def max\_money():

res = 1

n = int(input())

for i in range(1, n+1):

x = int(input())

res \*= x

res = res + n - 1

print(res \* 2022)

复杂度：对于n个整数，需要遍历一次将它们乘起来，时间复杂度O(n)，空间复杂度O(1)