Final Exam Preparation

15th January 2021 – 16th January 2021

Haven't got the key yet?

Admin 1: HURTGLMQJBSL

Admin 2: VGLRBLLSWGQ

Admin 3: LGCFMJSRRBTGMLM

Admin 4: EQBEMQGURBKKSLUBJFBLQY

Title		**Author**	**Editorialist**
A.	Literally Sorting	Admin 4	Admin 4
B.	Quadbonacci	Admin 3	Admin 4
C.	Pokemon Flag	Admin 2	Admin 2
D.	Thou Shalt Not Passeth	Admin 4	Admin 4
E.	Identical Substring	Admin 4	Admin 4
F.	Afterschool Candy	Admin 3	Admin 1
G.	Unown Apocalypse	Admin 2	Admin 4
H.	Pokemon Journey Continue	Admin 2	Admin 2
I.	Beauty Of A String	Admin 2	Admin 2 and Admin 4
J.	New Era Of Team Rocket	Admin 2	Admin 2
Bonus Problem			
K.	A Trip with Friends	Admin 1	Admin 2 and Admin 4
L.	Late for Work	Admin 4	Admin 4
M.	The Path to Sunibland	Admin 3	Admin 2
N.	Sherbet Equation	Admin 2	Admin 4
O.	Angry Alchemist	Admin 2	Admin 4

A. Literally Sorting

Notice that for basic sorting algorithms such as bubble sort or selection sort, your program will have a time complexity of $O(TN^2)$ due to multiple test cases. This means, for the worst case, your program will run roughly $T \times N \times N = 3000 \times 1000 \times 1000 = 3 \times 10^9$ operations. As a rule of thumb, an online judge can run 10^8 operations in a second. Thus, this kind of program will run for 30 seconds, which is far exceeding the time limit of 1 second.

To solve this problem, we need to use an advanced sorting algorithm that has a time complexity of $O(N \log N)$. With this, our program has a complexity of $O(TNlogN)$, running $T \times N \times \log_2 N \approx 3 \times 10^6$ operations, or about 0.03 seconds.

Time Complexity: $O(TNlogN)$

Source Code:
https://ideone.com/UbSWSo

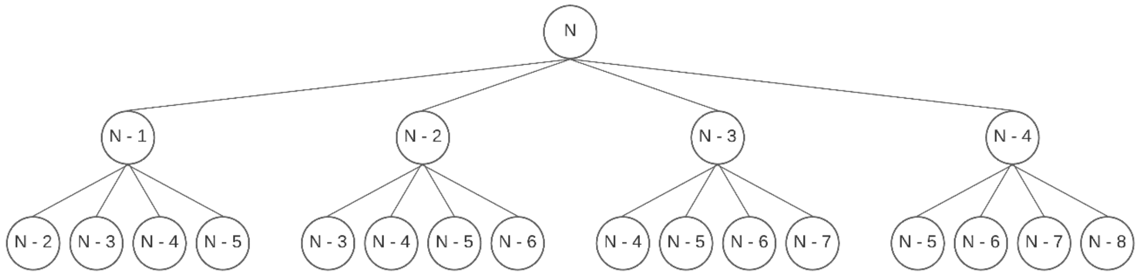
B. Quadbonacci

The first naïve solution is obviously recursion. Upon first observation, it can be noticed that the formula for the N-th term of quadbonacci ($N > 4$) is

$$F(N) = F(N-1) + F(N-2) + F(N-3) + F(N-4).$$

Thus, the easiest to code solution probably looks like this:
https://ideone.com/VIJBqz

However, this code is inefficient. Look at a part of the recursion process below.



Notice that several duplicate recursions are being executed, such as $F(N-2)$, $F(N-5)$, and more. These duplicate recursions are unneeded and causing the program to run significantly slower.

To solve this problem, we can store our calculated value in an array. So, when we need to find the value of $F(x)$ for some x, we can simply access the value stored in $quad[x]$.

Time Complexity: $O(N)$

Source Code:

Recursive	: https://ideone.com/JuytbO
Iterative	: https://ideone.com/47DKTR

C. Pokemon Flag

To solve this problem, we can look at the sample given. Look at the value of N in:

- Sample 0. If $N = 3$, the number pattern printed is: [1 2 1] 3 [1 2 1]
- Sample 1. If $N = 4$, the number pattern printed is: [1 2 1 3 1 2 1] 4 [1 2 1 3 1 2 1]
- Sample 2. If $N = 2$, the number pattern printed is: [1] 2 [1]

It means that if we want to draw a flag with a value $N \geq 2$, it must begin with a flag with pattern $N - 1$, then a number N, then end with a flag with pattern $N - 1$.

Then we can look at each pattern. If a section is numbered with X, it will make a stairs pattern with height L. And the width of the stairs is X times larger. As an example, if $X = 3$ and $L = 3$, the stairs pattern will be like this:

Stairs Pattern $X = 3$, height $L = 3$ with normal width:

```
3
3 3
3 3 3
```

Stairs Pattern $X = 3$, height $L = 3$ with X times width:

```
3 3 3
3 3 3 3 3 3
3 3 3 3 3 3 3 3 3
```

Then, we can print the flag pattern using recursion.

Time Complexity: $O(2^N N^2 L)$

Source Code:
https://ideone.com/nP2Hwp

D. Thou Shalt Not Passeth

The idea is straightforward. Given the list of participants, you only need to sort all participants based on the largest number and then the most preferred color. To simplify the color comparison, we can assign an integer value for each color. That way, we can compare each person's color easier than comparing color using strings.

Also, notice that the constraints are relatively small. Because of this, we can simply use a basic sorting algorithm (e.g., bubble sort, selection sort).

Time Complexity: $O(N^2)$ or $O(N \log N)$, depending on the sorting algorithm.

Source Code:
https://ideone.com/qq0XzT

E. Identical Substring

First Method:

Iterate through the string S and count the number of consecutive identical characters. If the current character is the same as the previous one, add 1 to the counter. Otherwise, reset the counter back to 1. If the current counter is equal to N, print the correct index, and stop the process.

Example:
S = “aabbccc”
N = 3

Position	Counter
aabbccc	1
a**a**bbccc	2
aa**b**bccc	1
aab**b**ccc	2
aabb**c**cc	1
aabbc**c**c	2
aabbcc**c**	3 (Finished)

Alternative Method:

Check all substrings of S with the size N. The first substring that satisfies the condition will be the answer.

Example:
S = “aabbccc”
N = 3

Substring	Is the answer?
aabbccc	No
a**abb**ccc	No
aa**bbc**cc	No
aab**bcc**c	No
aabb**ccc**	Yes (Finished)

Time Complexity:

First Method	: $O(\|S\|)$
Second Method	: $O(\|S\|N)$

Source Code:

First Method	: https://ideone.com/MV4jzS
Second Method	: https://ideone.com/9Kh2Ea

F. Afterschool Candy

To solve this problem, we must sort all candy types based on their price. Do not forget that when you sort the candy prices, you must also include the stock in the sorting process so they won't get mixed up.

After that, we just need to iterate through all candy types and find the maximum number of candy that Jay and Max can buy. If the number of candy Jay and Max can buy is at least N, print **"YES"**, else print **"NO"**.

Do notice that we cannot manually pick candies one by one, as the constraint for B_i is 10^9. We can check the number of candies we can buy for the i-th type of candy by finding $\min\left(\frac{K}{price[i]}, stock[i]\right)$. This formula determines whether you can buy all the stock for the current candy, or you can only buy some of it with your remaining money.

Time Complexity: $O(TM\ logM)$.

Source Code:
https://ideone.com/3Gx1tK

G. Unown Apocalypse

To solve this problem, we need to decrypt the word in the text back to normal, if the word is not decrypted or still the same as before, then it means it is a tricky word or it means that do not print the word.

We can make an array of int that can be used like a dictionary. This array saves the ASCII value of a character. When we input the pairs, do not forget that there are lowercase letters too.

One of the solutions is to split the string into words by using strtok function from <string.h> library. Each word is then decrypted back to normal and we can use a parameter that determines whether the word has been decrypted back or stay the same as before. Once it is done, if it has already been decrypted back, print the word.

Time Complexity: $O(L)$

Source code:

https://ideone.com/zE72Vh

H. Pokemon Journey Continue

In this problem, we need to find how many islands there are on the map. It is said that "Each island is not touching or adjacent to each other horizontally, vertically, or diagonally". This means two lands are still part of the same island if both of them touch diagonally with each other.
As an example:

```
3 5
#.#.#
.#.#.
#.#.#
```

This is still called as 1 island since the lands are touching diagonally. To solve this, we can use the flood fill algorithm. When we spot a land, it means that we found an island.
We recursively visit every adjacent cell in 8 directions and check whether that cell is part of the same island the current cell is. After visiting a cell, change that cell into '.' so that we do not revisit the same cell twice. Do this until all cells become '.'.

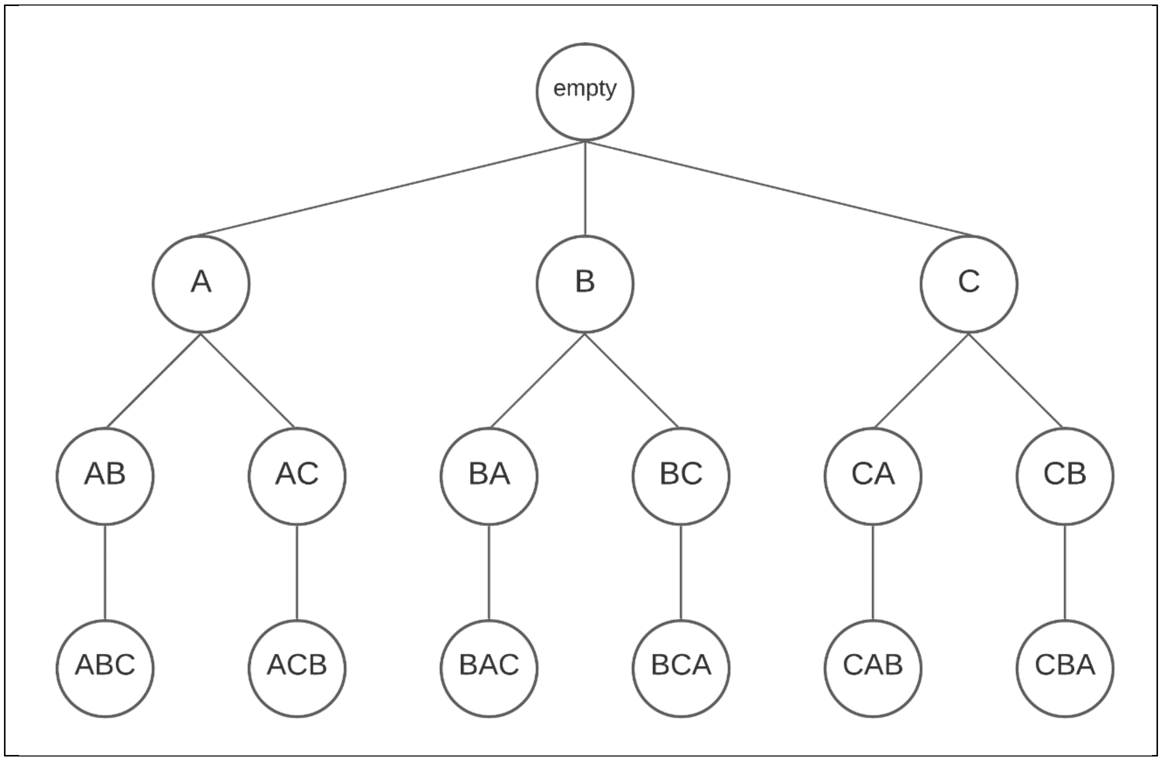
Time Complexity: $O(NM)$

Source Code:
https://ideone.com/f1B7JR

I. Beauty of A String

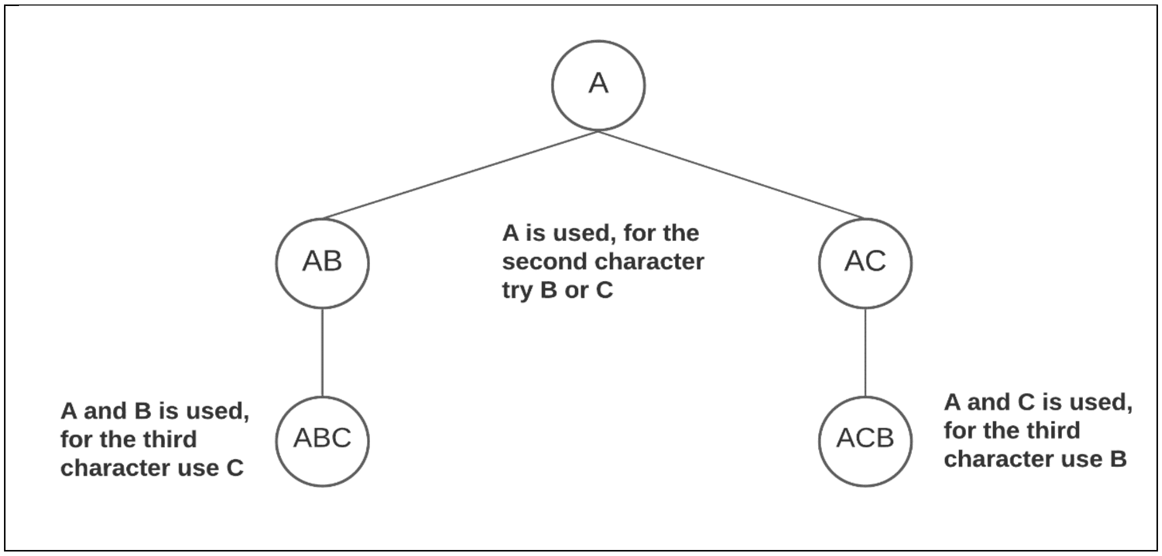
To solve this problem, Initially, we make a list of numbers for the value of beauty that will be obtained later in the process. Then we can generate all possible strings that can be made by permutation. In each possible permutation, we calculate the beauty values and marked them on the beauty values list. After the permutation process has been done, then we can just search, how many beauty values has been marked and print the answer.

To understand the process of permutation, check the diagram below:



For every character that we want to add, check whether the character has been used or not, temporarily mark the mentioned character as taken, then proceed to continue the recursion for more characters.

Example:


A
AB
A is used, for the second character try B or C
AC
A and B is used, for the third character use C
ABC
ACB
A and C is used, for the third character use B

Time Complexity: $O(N!)$

Source code:

https://ideone.com/XbOVcL

J. New Era of Team Rocket

In this problem, we can solve just by doing exactly what the problem said or implementation. We list the contestant in a struct array with the identity: name, total vote, and total cities won. After listing the contestant, we go to the next part where it is the votes from each city given.

In this part, we can use an array of strings to list the winner, then move it into the contestant list. Do not forget to count the number of votes each contestant gets in each city.

If the calculation and list of each city have been done, then we sort the contestant list based on their total votes and total city win. We want it to be sorted by total votes in descending order, or if the total votes are the same, then sorted by total city won in descending order.

Then we have our winner on the first index of the list. But we need to check again whether the first and the second index information (total votes and total cities won) are the same or not. If same, then the election must be repeated. Otherwise, we have the winner.

Source code:

https://ideone.com/e0g08F

One of the admins reading this problem:

The creator:

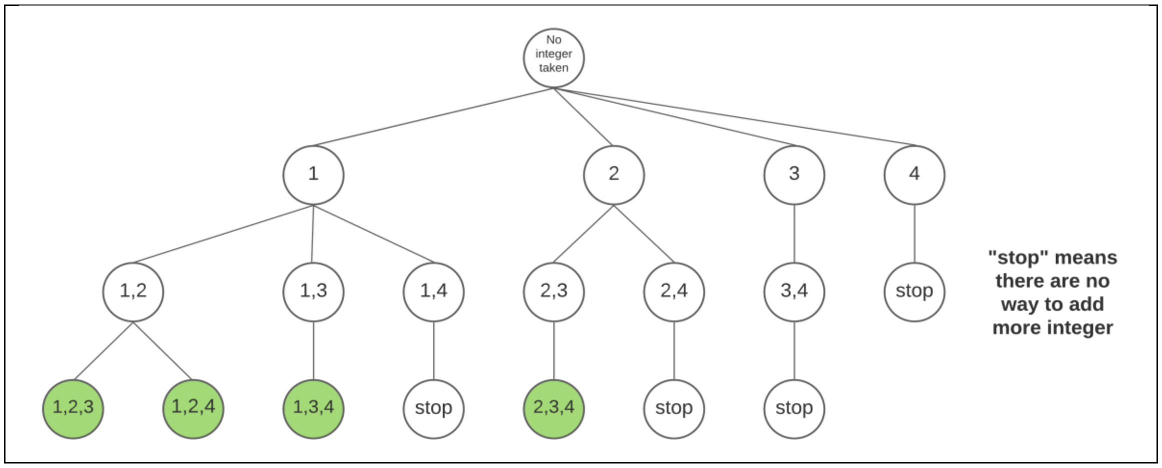
K. A Trip with Friends

In this problem, we can solve it by generating all possible combinations of the group. The method is quite similar to how we generate all permutation in the problem I, Beauty of A String. To guarantee that the combination is sorted, we always pick an integer that is larger than the integer we pick previously. With this, we also do not need to check whether an integer is taken or not.

For example, check the diagram below:

Let $N = 4, K = 3$.

All possible combinations:



In each possible combination, check whether there are fail pairs (a pair of students who do not like each other). One of the solutions is we can use a 2D matrix. If x and y are a pair of students who do not like each other then we mark matrix[x][y] and matrix[y][x]. So that, when we check whether there are fail pairs or not, we can just look from the marked matrix.

Time Complexity: $O(\frac{N!}{K!(N-K)!}K^2)$

Source Code:

https://ideone.com/5OLMsN

L. Late for Work

Observe that to maximize the number of presents Santa can deliver, we need to pick children starting with the lowest value of naughtiness first. So, the first step is to sort all naughtiness in ascending order. Since the constraint for N is large, we need a sorting algorithm with $O(N \log N)$ time complexity.

The second problem is the query. You cannot count the answer simply by picking a child while the average value is less than K_i, as the time complexity would be $O(NQ)$ for the worst case. A faster solution is needed.

Notice that if we try to pick X children, then we will always pick X children with the smallest value. Thus, we can compute the sum of the first X children using the prefix sum.

Prefix sum, more formally, can be expressed as the formula below.

$$Sum[X] = \sum_{i=1}^{X} A[i]$$

Where $A[i]$ is the i-th smallest naughtiness value.

Notice that the prefix sum is always sorted in ascending order since $A[i] > 0$ for all i and the array is sorted. Because of this, we can do a binary search to determine the maximum number of children to pick.

Example:
Look at the sample test case. First, sort all values in ascending order.

index	1	2	3	4	5
Arr[index]	2	3	4	7	9

Then, calculate the prefix sum.

index	0	1	2	3	4	5
Arr[index]		2	3	4	7	9
Sum[index]	0	2	5	9	16	25

Finally, do a binary search for the queries. Below is the process for $K = 3$.

Left	Mid	Right	Average	Next Action
0	3	5	$\frac{Sum[3]}{3} = \frac{9}{3} = 3$	Average <= K, try higher value.
3	4	5	$\frac{Sum[4]}{4} = \frac{16}{4} = 4$	Average> K, try lower value.
3	4	4	$\frac{Sum[4]}{4} = \frac{16}{4} = 4$	Average> K, try lower value.

3	3	3	Answer is found.

So, the maximum number of children Santa can give presents to for $K = 3$ is 3.

Time Complexity: $O\big(T(NlogN + QlogN)\big) = O(TlogN(N + Q))$

Source Code:
https://ideone.com/MONQ1f

M. The Path to Sunibland

In this problem, we can use DFS or Depth-First Search algorithm (more commonly known as floodfill) on finding the path. To start, we need to find the start point (x and y). Then we can recursively go to 4 possible paths:

- $x - 1$ and y
- $x + 1$ and y
- x and $y - 1$
- x and $y + 1$

We continue each of that part again repeatedly until the point out of the border or reach the destination point, do not forget that we need to list the movement too (U, R, D, and L). Remember that we cannot pass the same cell twice. It means that we can use the permutation concept, where we temporarily mark all passed points and erase those marks when a path or answer has been found. When we reach the destination point, do not forget to keep the string in an array of strings.

After the depth-first search has been done, then we can sort the array and output the answer. It is recommended to sort the array in $O(N \log N)$ time complexity to avoid getting Time Limit Exceeded verdict.

Source Code:

https://ideone.com/hzXbU0

N. Sherbet Equation

There are two things that need to be solved for an accepted solution.

The first one is calculating 2^X for $S(X,0)$. Doing this iteratively will cause your program to run slower, as each $S(X,0)$ will have a time complexity of $O(X)$.

The issue is, there might be several $S(X,0)$ being executed. As an example, executing $S(12,2)$ will execute both $S(12,1)$ and $S(6,1)$. $S(12,1)$ will execute $S(3,0)$ and $S(2,0)$. $S(6,1)$ also executes $S(3,0)$ and $S(2,0)$. Notice that both function $S(3,0)$ and $S(2,0)$ run twice, which is unnecessary.

To counter this problem, we can save the value of $2^X \ mod \ 10^9+7$ in a separate array, namely, $calc$. Then, simply return the value of $calc[X]$ whenever we call $S(X,0)$.

The second one is the factorization process. Again, doing the factorization process in $S(X,b)$ will have a time complexity of $O(X)$. To optimize this, notice that if an integer i is a factor of X, then $\frac{X}{i}$ is also a factor of X. With this, we can reduce the looping to only $\sqrt{X}$ instead of X, as the greatest integer i where $i \le \frac{X}{i}$ is $\sqrt{X}$. Any factor beyond $\sqrt{X}$ has already been checked when we check i, where $i \le \sqrt{X}$.

Source Code:
https://ideone.com/Bfy3t5

Note:
Alternatively, you can also use binary exponentation for calculating 2^X.

O. Angry Alchemist

Idea 1: Brute force

Try every possible combination of A_i, B_j, C_k, D_l. If $A_i + B_j + C_k + D_l = K$, add the answer by 1.

The time complexity is $O(N^4)$, which is not fast enough and will end up getting a Time Limit Exceeded verdict.

Idea 2: Slightly faster brute force

Sort the array D. Try every possible combination of A_i, B_j, C_k. Observe that if $A_i, +B_j + C_k + X = K$, then $X = K - (A_i + B_j + \ C_k)$. Search for the value X in array D using lower bound binary search.

The time complexity is $O(N^3 logN)$, still not enough and will end up getting a Time Limit Exceeded verdict.

Idea 3: Even faster brute force (Expected Solution)

For every possible combination of A_i and B_j, store the value of $A_i + B_j$ in a separate array, namely, $sumAB$. Do the similar thing for array C and D. Store the value of $C_i + D_j$ in an array, $sumCD$. Sort the array $sumCD$. Now, for every value in $sumAB$, find the number of values in $sumCD$ that is equal to $K - sumAB_i$ using lower bound binary search. Add the number to the sum.

The time complexity is $O(N^2 \log(N^2))$

Source Code:

https://ideone.com/oavVTI