Mini Comp 1 Editorial

Credits: LMH, Sampson, Steven, Theo

WiFi Security

Setter: LMH

Prepared By: LMH

Beware of Pascal `string` length limit (255)

Basic IO and Data Processing

WiFi Security

Subtask 1, 3:

- Count frequency of `"a"`, `"b"`
- Case 1 - `B = "ab"`
  - Change the majority of `A` to `"a"`, others to `"b"`
- Case 2 - `B = "ba"`
  - Change the majority of `A` to `"b"`, others to `"a"`

WiFi Security

Subtask 2, 4 - Solution 1

- Count `A` letters frequency in an array
- Create a mirror string `C`
- Read `B` char by char `b`
  - Find the letter ch with highest frequency
  - For each char in `A`, such that `Ai = ch`
    - Map `Ci` to `b`
  - Reset `ch`'s frequency to be `0`
- Output `C`

Expected Score: `100`

WiFi Security

Subtask 2, 4 - Solution 2

- Count `A`'s letter frequency in an array
  - Packed data type for storing frequency and letter
- Create a mirror string `C`
- Sort frequency array decreasingly by its frequency
- Create a mapping between original char and encrypted char
  - `i-th` char of sorted frequency array -> `i-th` char of B
- Map each char of `A` to `C` using the above mapping
- Output `C`

Expected Score: `100`

2048 Game

Setter: Sampson

Prepared By: Theo

Observation 1: All numbers should be in form of 2^x, where x is a positive integer. If some numbers are not in this form, it is obviously impossible

2048 Game

Observation 2: Consider the largest number be 2^x. If we form 2^x from its elementary form (2^(x-1) * “2”s), it takes at least (x-1) turns

Reason: In first turn we form 2^(x-1) * “2”s into 2^(x-2) * “4”s, and second turn into 2^(x-3) * “8”s. This process continues until we reach 2^(0) * “2^x”s. As the power is reduced x-1 times, it takes at least (x-1) turns

2048 Game

Wrong solution 1: check if the set of numbers contain at least (x-1) * “2”s as produced by the (x-1) turns, if it is not true then return Impossible

Observation 3: Consider the first “extra” 2. It has 2^(x-2) turns idling, therefore it can exist in forms of 2^2,

Snow White's Shuffling

Setter: Sampson

Prepared By: Sampson

Snow White's Shuffling

Solution 1: Naive

Store the deck by an array d[0..p]

For every shuffle operation, transform d[p - k], d[p - k + 1], …, d[p] one by one.

If k == 4,

index	p - 4	p - 3	p - 2	p - 1
d[]	10	4	7	6
new d[]	4	7	6	10

Time Complexity: O(MK)

Snow White's Shuffling

Observation 1:

A shuffling consists of

- removing the $(p - k)^{th}$ element
- shifting the top $(k - 1)$ elements to the left by 1 index
- placing the removed element on the top

The shifting costs the most time.

Snow White's Shuffling

Solution 2: Preventing Shifting

For every shuffle operation,

- remove the $(p - k)^{th}$ element, mark the position as null
- the top (k - 1) elements remain in place
- placing the removed element on a new cell representing the top

Snow White's Shuffling

Now a shuffling costs O(1) time

How to find the x^{th} top card?

- Go down from the top of the deck
- Count how many numbers (excluding NULL) have been gone through
- Stop when x numbers have been found

index	p - 4	p - 3	p - 2	p - 1	p (new)
d[]	10	4	7	6	
new d[]	NULL	4	7	6	10

Snow White's Shuffling

Finding the x^{th} card costs O(x) naively

It can be speeded up by a data structure such as BIT or segment tree

The cost of finding the x^{th} card is optimized to O(lg p)

Total time complexity: O(M lg M)

Snow White's Shuffling

After shuffling, the order of the top (k - 1) numbers do not change.

The only thing is that the $(p - k)^{th}$ element is put at the back.

We can use a queue to store the top k numbers.

Snow White's Shuffling

d[]: Elements under the top $(p - k)^{th}$ card

q[]: The top k elements

If k == 3

d[]	10	4	7	6
q[]	5	3	1	

Adding a new card 8,

d[]	10	4	7	6	5
q[]		3	1	8	

Snow White's Shuffling

Shuffling

d[]	10	4	7	6
q[]	5	3	1	

becomes

d[]	10	4	7	6
q[]		3	1	5

Snow White Shuffling

Finding the x^{th} element

If x <= k, find the answer in q[]
else find the answer in d[]

Each operation takes O(1) now
Total time complexity: O(M)

Advanced Optimization Coding

Setter: Sampson

Prepared By: Steven

*AOC originally stands for Age of Empires - The Conquerors, a computer game popular among coders.

Advanced Optimization Coding

Let's start with brute force.

There are 2^N states, namely 0,1,2, .. , 2^N-1.

For each state in binary form, e.g. 10000011,

the i^{th} bit from right

= "1" means accept the i^{th} contestant,

= "0" means reject.

Here, the 1^{st}, 2^{nd}, 8^{th} contestant are accepted. Total number of contestants is 3. If no contestant complains, record the answer.

Finally, output the maximum.

Advanced Optimization Coding

Number of states: 2^N

Check if anyone will complain: O(N)

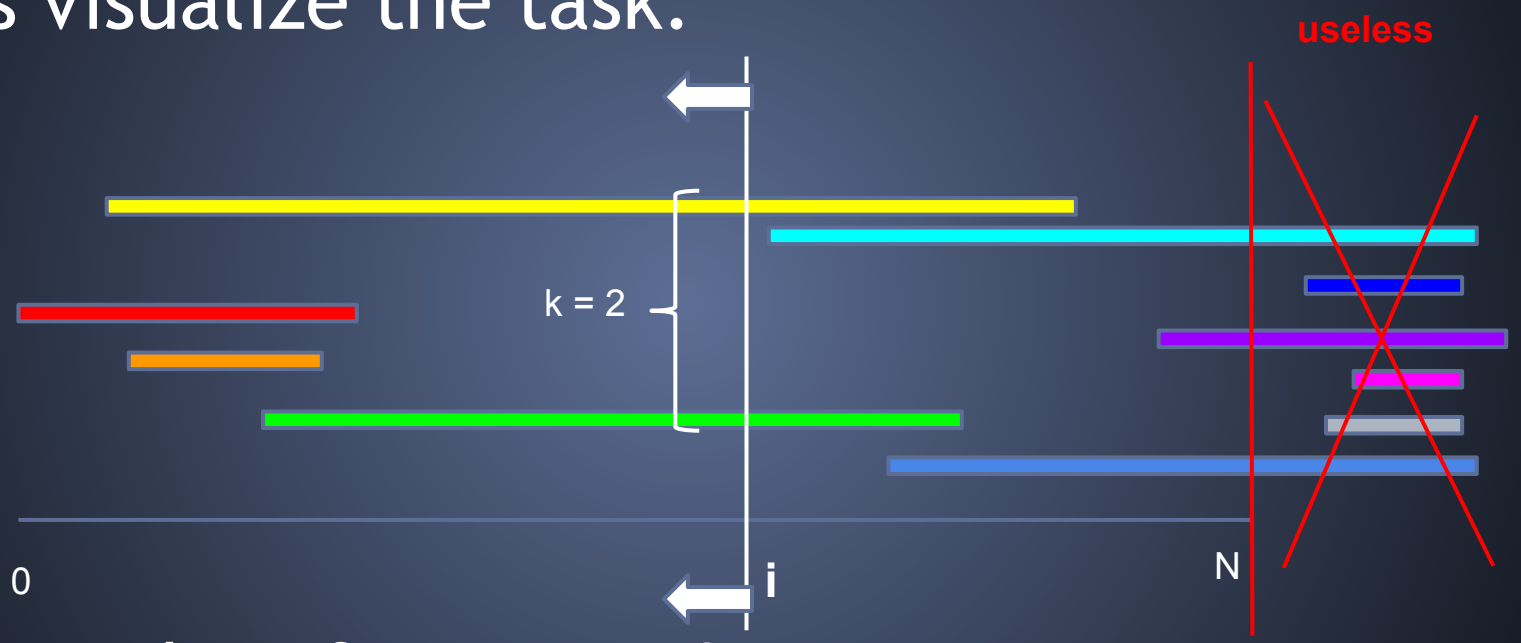
Overall time complexity: $O(N*2^N)$

Enough for Subtask 1: 1 <= N <= 10

With such an easy solution, you get 50 points!

Advanced Optimization Coding

Let’s visualize the task:



Sweep a line from N to 0.

k is the # of intersections, i is current index.

When k >= i, output i.

Advanced Optimization coding

Implementation 1:

Sort all the 2N endpoints and then scan from right to left.

Maintain a counter **k**, storing the current number of intersections.

On encountering a right endpoint, **k++**; for a left endpoint, **k--**.

Time complexity: O(Nlog(N))

Enough for Subtask 2: 1 <= N <= 100000

Advanced Optimization Coding

Implementation 1.9 (beta):

Maintain an array a[0..N]. Initialize as zeros.

[0000000000000000000000000]

For each L_i and R_i,

increment $a[L_i]$, $a[L_i+1]$, …, $a[R_i]$ by 1.

Final result:

[0111222223333111444332210]

Scan from right to left. If a[i] >= i, output i.

Time complexity: $O(N^2)$

Advanced Optimization Coding

Bottleneck of implementation 1.9 (beta):

For each L_i and R_i,
increment $a[L_i]$, $a[L_i+1]$, …, $a[R_i]$ by 1.

Advanced Optimization Coding

Rather, we do

For each L_i and R_i,

$a[L_i] = a[L_i] + 1$

$a[R_i + 1] = a[R_i + 1] - 1$

Finally,

For i from 2 to N,

$a[i] = a[i] + a[i - 1]$

Final result:

[01112222333311144433221O] - the same!

Advanced Optimization Coding

The technique is called “difference array”.

Implementation 2.0: O(N)
Full score.