# Alice, Bob and Circuit

### Task Review

- Original task
  - *n* persons, each with a name and a number.
  - *m* letters, each specified by a sender and a recipient's names, with the content of the sender's number.
  - For each person, calculate the sum of the received letter contents (modulo  $2^{16}$ )
  - $n \le 700, m \le 1000$
- Communication style
  - Alice knows the information of n persons, and Bob knows that of m letters. Each needs to send a binary string within  $10^5$  bits to Circuit.
  - Circuit has to use at most  $2 \times 10^7$  binary logic gates to get the results.

- n = 1, m = 0
- No letters are sent. The result is simply 0.
  - 0 can be generated by a gate of operation=0.
  - Or, 0 can be sent from Alice to Circuit.
- Expected score: 4 points.

- $n = 1, 0 \le m \le 1$
- If m = 1, the answer is the number. Else, the answer should be 0.
- Alice sends the number (16 bits).
- Bob sends m (1 bit).
- Circuit performs 16 AND operations, and then output.
- Expected score: 8 points.

- $n = 1,0 \le m \le 1000$
- Answer is the number multiplies m.
  - It is unnecessary to implement multiplication in circuit. Conducting addition for multiple times is enough for this subtask.
- Alice sends the number (16 bits).
- Bob sends m bits (contents does not matter, only to let circuit() directly know m)
- Circuit performs m-1 times of addition
  - Assuming each addition needs 80 gates, there are 80000 gates in total.
- Expected score: 12 points.

## How to implement addition?

- 1-bit adder
  - Input 3 bits: a, b, c
  - Output 2 bits: s1, s0 (the binary form of A+B+C is S1 S0)
  - S0 = a XOR b XOR c
  - S1 = (a AND b) or ((a XOR b) AND c)
  - 5 gates
- 16-bit adder
  - Cascading 16 1-bit adders
  - For the three inputs of the 1-bit adders, two of them come from input numbers, and the other is the carry bit from the lower position.
  - The carry bit is initially 0, and the overflowed carry bit is discarded (equivalent to modulo 2<sup>16</sup>)
  - At most 80 gates (with optimization chances)

- n = 26, names appearing in order, no duplicate letters
  - Means that Bob's input can be arranged as an  $n \times n$  0/1-matrix
- Alice sends the 26 numbers in the order from a to z (26×16 bits)
- Bob sends the 0/1-matrix (26×26 bits)
- Circuit performs AND and addition according to the matrix
  - Needs  $26 \times 26 \times 16$  AND-s and  $26 \times 26$  additions, which is far below the limit of  $10^7$  gates.
- Expected score: 24 points.
  - Can obtain 36 points combined with Subtasks 1~3.

- Based on Subtask 4, but the names may not be ordered.
- Nevertheless, Bob knows all names.
  - His input can still be arranged as an  $n \times n$  0-1 matrix.
- If Alice and Bob follow the same order (e.g., lexicographical), numbers sent from Alice can still correspond to Bob's matrix.
  - However, this order may be different from the order of answer output.
- Alice sends an extra 26×26 0/1-matrix (where there are exactly one element 1 from each row and from each column), so that Circuit can rearrange the order of the answers.
- Expected score: 48 points.
  - Can obtain 60 points combined with Subtasks 1~3.

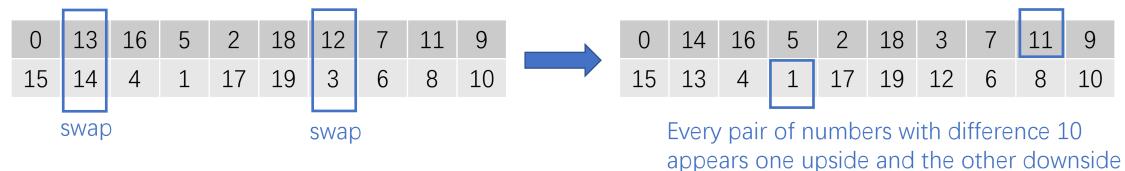
- *n* is not restricted to equal 26, and may be any value no more than 30.
- There is nothing special. You should pass this subtask unless your solution fails strangely.
  - The original purpose to place a subtask of n = 26 is to facilitate debugging.
- Expected score: 54 points.
  - Can obtain 66 points combined with Subtasks 1~3, which is the maximal score achieved during the competition.
- There also exists another approach of  $16 \times 19 \times n \times m$ , where the names are sent from Alice and Bob to Circuit. This can also get 66 points.

#### Subtasks 7~9

- Subtask 7:  $n = 676, m \le 1000$ , names are in order.
- Subtask 8: names may be out of order.
- Assuming we can solve 7, then how to do 8?
  - We need a method to adjust the output order.
  - It is too inefficient to adopt an  $n \times n$  0/1-matrix!
- Considering there are n! possible permutations for n elements, the lower bound is to use  $\log_2 n! = O(n \log n)$  bits to encode a permutation, to be performed by Circuit
- We now provide a construction

## How to implement an $O(n \log n)$ permutation?

- For example, we need to rearrange 20 elements to the ordered state.
- Split the array into the upper and the lower halves, each with 10 elements.
- Swap the elements up and down, so that every pair of numbers with difference 10 appears one upside and the other downside.
  - It can be proved that it is always possible to do so.



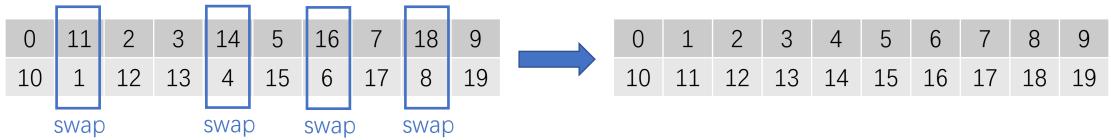
# How to implement an $O(n \log n)$ permutation?

• Recursively process the upper half and the lower half, so that the ones-place becomes all correct.

0	14	16	5	2	18	3	7	11	9	0	11	2	3	14	5	16	7	18	9
15	13	4	1	17	19	12	6	8	10	10	1	12	13	4	15	6	17	8	19

Every pair of numbers with difference 10 appears one upside and the other downside

• Finally, swap the elements up and down, so that all elements become correct.



## How to implement an $O(n \log n)$ permutation?

- There are n/2 chances of optional swapping before recursion and after recursion, respectively. We can use n/2 bits each, to record whether to swap.
- It is similar when *n* is odd. There are  $\left\lfloor \frac{n}{2} \right\rfloor$  chances of optional swapping before and after recursion, respectively. Not going into details here.
- Let T(n) denote the number of bits to encode a permutation of n elements. Then  $T(n) = 2\left\lfloor \frac{n}{2} \right\rfloor + T\left( \left\lfloor \frac{n}{2} \right\rfloor \right) + T\left( \left\lfloor \frac{n}{2} \right\rfloor \right)$ . According to the master theorem,  $T(n) = O(n \log n)$ .

## Full-Score Algorithm

- Convert each person and each letter to a 4-tuple (U, V, W, T).
  - A person with name X and number Y converts to (X, X, Y, 0).
  - A letter with sender P and receiver Q converts to (P, Q, 0, 1).
  - (Assuming each string is converted to a 19-bit integer.)
- Sort all tuples by U first then V.
  - The sorting algorithm will be introduced later.
- Consider all tuples in order, maintaining a temporary variable C:
  - When meeting an element of T=0 (person), assign: C <- (W of the current element)
  - When meeting an element of T=1 (letter), assign: (W of the current element) <- C
- At this point, all letters are assigned with the sender's number.

## Full-Score Algorithm

- Next, sort all tuples by V first then T (descending).
- Consider all tuples in order, maintaining a temporary variable S (initially 0):
  - At an element of T=1 (letter), assign: S < -S + (W of the current element)
  - At an element of T=0 (person), assign: (W of the current element) <- S; S <- 0
- At this point, all persons are assigned with the correct computation result.
- Sort all tuples by T first then U.
  - Now the first *n* elements are persons.
- Finally, adopt the  $O(n \log n)$  permutation to rearrange the persons into the order of Alice's input, and output the result.

## How to implement sorting?

- Conventional O(n log n) sorting (e.g., quicksort, mergesort, heapsort) cannot be directly implemented in circuits.
- But we can find that:
- Alice and Bob can sort their own elements in advance, respectively.
- Circuit only needs to merge two sorted arrays.
  - We can use  $O(n \log n)$  in-place merge.
  - Note that the conventional O(n) merging is still unimplementable in circuits.

# Final Algorithm

- Alice sends to Circuit:
  - n 4-tuples of persons, sorted by name U.
  - The bits encoding the permutation which can rearrange *n* persons from ordered by U into ordered by Alice's input (i.e., output order).
- Bob sends to Circuit:
  - m 4-tuples of letters, sorted by sender U.
  - The bits encoding the permutation which can rearrange m letters from ordered by U into ordered by V.

# Final Algorithm

- Circuit's computation:
  - After receiving the 4-tuples, merge two sorted arrays (first U then V).
  - Perform the first sequential scan, so that the letters are assigned with senders' numbers.
  - Undo the previous merge (which can be implemented by recording whether each comparison leads to a swap).
  - Apply Bob's permutation, to rearrange the letters from ordered by U into ordered by V.
  - Merge two sorted arrays again (first U then T descending).
  - Perform the second sequential scan, so that each person gets the correct result.
  - Undo the previous merge.
  - Apply Alice's permutation, to rearrange the persons into Alice's input order.
  - Output the answer.
- Expected score: 100 points.

## Possible but Unnecessary Optimizations

- Constant propagation: e.g., 0 AND x = 0, 0 OR x = x
- "Undefined" propagation: e.g., undefined XOR x = undefined
- Dead circuit optimization: If the result of a computation gate is never used (directly or indirectly) by any output, the gate can be eliminated.
  - Enable less thinking when writing code.
- Duplicate circuit optimization: Merge two computation gates with the same operation type and dependency gates.
  - Help check for problems in the code.
- Reference answer needs about  $10^7$  gates, or about  $8 \times 10^6$  after optimization.

# The End