I'm very sorry. I'll type my homework next time.

1. Array, Linked list, and Recursion.

① The fastest way I can come up with is by using a Hashmap to record whether a number has appear before.

So I will use the number at the place as key, and the sign as value.
For example: array[k] = num, then we use the sign of array[|num|] as indicator ⇒ if array[|num|] > 0, change it to negative
if array[|num|] < 0, it means "num" already appeared once, so |num| is duplicate.

```
for index in 0 to n - 1
    value <— abs(array[index])
    if (array[value] > 0)
        array[value] *= (-1)
    else
        return value
```

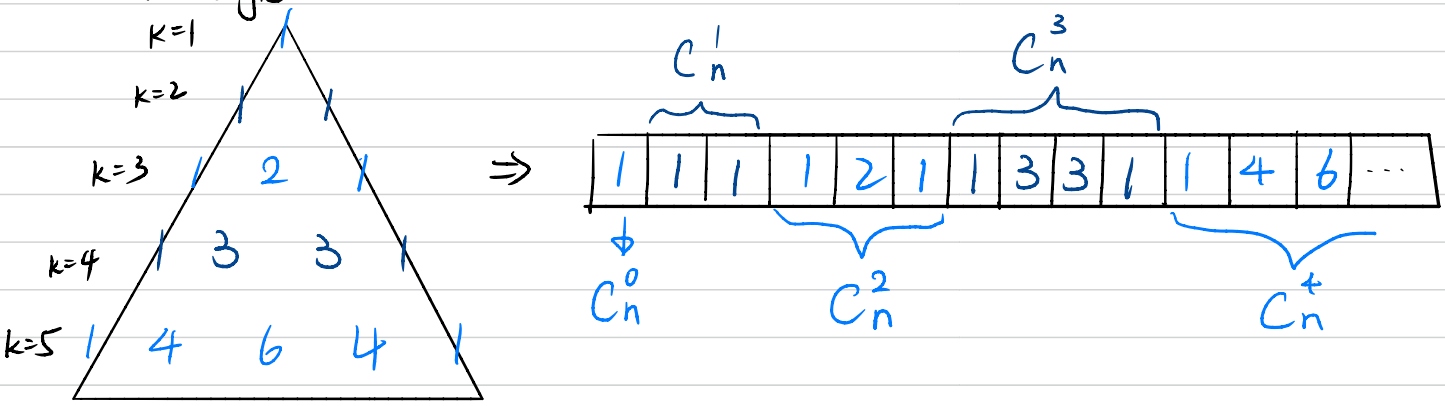
* value as key
sign of array[value] as indicator

* Time O(n), Space O(1)

∴ So if a > 0 appears once and b > 0 appears twice.

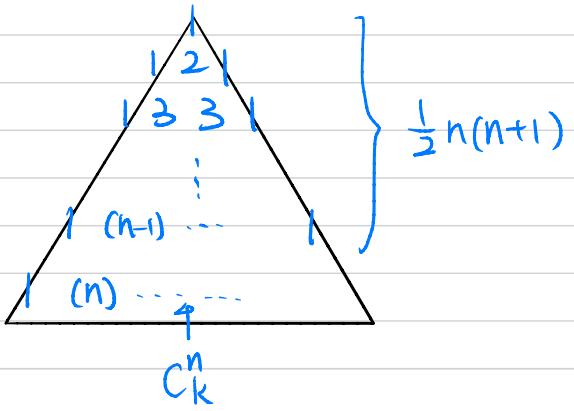
1. if a appears ⇒ change array[a] to negative (same as the first time b appears)
2. Since b appears the second time, you'll find array[b] < 0 ⇒ you know it appears twice so return it.

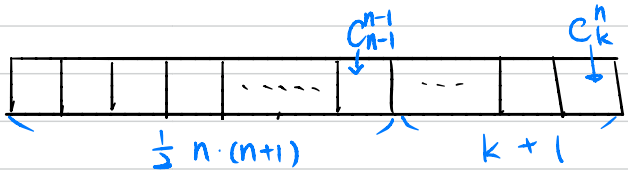
② Pascal Triangle



∴ for C_k^n, $0 \le k \le n$ ⇒ there is n+1 elements
⇒ So you store $C_n^0 \sim C_n^k$ consecutively with upper "k" and lower "n" increasing ($C_0^0, C_0^1, C_1^1, C_0^2, C_1^2, C_2^2 \cdots$)

To get C_k^n, you have to go through $(n-1)$ consecutive $C^{\square}$s.

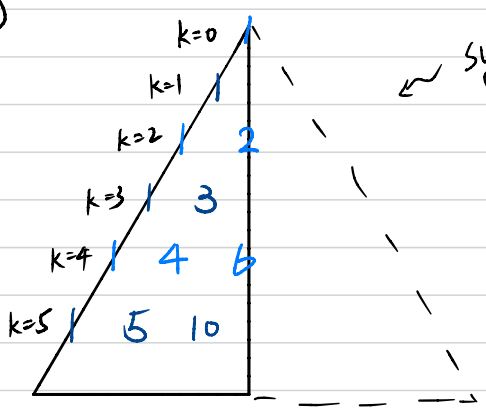


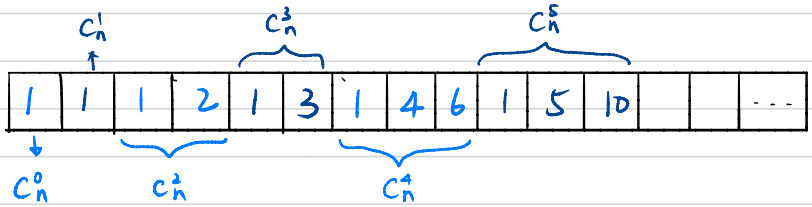



```
int getCombination(int n, int k){
    return array[n * (n + 1) / 2 + k];
}
```

∴ So C_k^n is the $[\frac{1}{2}(n)(n+1) + k + 1]$ number, but you have to minus one since array start from 0 $\Rightarrow C_k^n = array[\frac{1}{2}n\cdot(n+1) + k]$

③





$(C_k^n = C_{n-k}^n)$

∴ same as previous but only store half of the Pascal Triangle.

for every n, we store first $\lfloor \frac{n}{2} \rfloor + 1$ elements of $C_0^n \sim C_n^n$

```
int getCombination(int n, int k){   ← getCombination(n, k) = C_k^n
    if (2 * k > n)                  ← C_k^n = C_{n-k}^n
        k = n - k;
    int prev_num = 0;
    for(int i = 0; i < n; i++)      ← calculate elements of C_□^0 ~ C_□^{n-1}
        prev_num += (i / 2 + 1);
    return array[prev_num + k];
}
```

④ The fastest way I can come up with is using two pointers method. (turtle - rabbit algorithm).

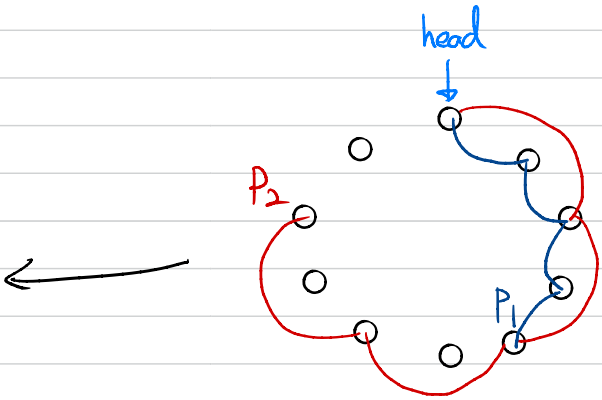
So you basic create two pointers P_1 and P_2 while P_1 moves one step at once and P_2 moves two.

Thus, if P_2 will get back to head at the next step ($P_2 \to next \to next = head$), P_1 is at the end of first sub-ring.

Then the rest is simple, point $P_2 \to next \to next$ to $P_1 \to next$.

And $P_1 \to next$ to head.

```
P1 = P2 = head;
while(1){
    if(P2->next->next == head){
        P2->next->next = P1->next;
        P1->next = head;
        break;
    }
    P2 = P2->next->next;
    P1 = P1->next;
}
```



⑤ The same concept as bubble sort, but swap if odd precede even.

```
void bubble_recursive(int array[], int length){

    if(length == 1) return;

    for(int k = 0; k  < length - 1; k++){
        if(array[k] % 2 == 1 && array[k + 1] % 2 == 0){
            int tmp = array[k];
            array[k] = array[k + 1];
            array[k + 1] = tmp;
        }
    }

    return bubble_recursive(array, length - 1);

}


bubble_recursive(array, sizeof(array) / sizeof(int));
```

terminating condition

if odd, even ⇒ swap

next layer

the number of elements of array

(b) The method I come up with is using two pointer and find the odd number who is nearest to the begin and even to the end then swap them. Repeat it until two pointer crossed.

```
int p1 = 0, p2 = sizeof(array) / sizeof(int) - 1;

while(p1 < p2){

    if(array[p1] % 2 == 0) p1++;
    if(array[p2] % 2 == 1) p2 --;

    if(array[p1] % 2 == 1 && array[p2] % 2 == 0){
        int tmp = array[p1];
        array[p1] = array[p2];
        array[p2] = tmp;
    }

}
```

continue to search odd ... even.

2. Analysis Tool

① Suppose one of a scenario that $f(n) = g(n)$, so let $d(n) = a \cdot f(n)$, $e(n) = b \cdot g(n)$

⇒ if $a \neq b$, $d(n) - e(n) = (a-b) \cdot f(n) = O(f(n))$ — ①

and $O(f(n) - g(n)) = O(1)$ — ②

∴ if $f(n)$ has a higher degree (or is different) than 1

From ①, ② $d(n) - e(n) = O(f(n)) \neq O(f(n) - g(n)) = O(1)$

Thus, it's proved that for $d(n) = O(f(n))$, $e(n) = O(g(n))$

→ $d(n) - e(n)$ not necessarily equals to $O(f(n) - g(n))$.

② Claim: if $f(n) + g(n) = O(\max\{f, g\})$ and $\max\{f, g\} = O(f(n) + g(n))$

Proof: Suppose $f, g > 0$ and without loss of generality: $f > g$

1° $\max\{f(n), g(n)\} = f(n) \leq c \cdot (f(x) + g(x))$, if $c = 1$, then for n_0 that makes $f(n_0), g(n_0) > 0$, it holds ∴ $n_0 \in R$

Thus: $\max\{f(n), g(n)\} = O(f(n) + g(n))$.

2° $f(n) + g(n) \leq c \cdot \max\{f(n), g(n)\} = c \cdot f(n)$, if $c = 2$, then for n_0 that makes $f(n_0), g(n_0) > 0$, it holds ∴ $n_0 \in R$

Thus: $f(n) + g(n) = O(\max\{f(n), g(n)\})$

From 1° & 2° ⇒ $O(f(n) + g(n)) = O(\max\{f(n) + g(n)\})$, QED

③ Suppose a function $f(n) = \begin{cases} n^2, & n \in R \\ 1, & n \notin R \end{cases}$

1° consider $f(n) \leq c \cdot n$, for $n \in R$, there isn't any $c > 0$ s.t. $n^2 \leq c \cdot n$ for $n > n_0$ ∴ $n^2 \geq c \cdot n$

Thus $f(n) \neq O(n)$

2° consider $f(n) \geq C \cdot (n)$, for $n \notin \mathbb{R}$, there isn't any $C > 0$, st
$1 \geq C \cdot n$ for $n > n_0$ $(c > 0)$

Thus $f(n) \neq \Omega(n)$

From 1°, 2°, for $f(n) = \begin{cases} n^2, & n \in R \\ 1, & n \notin R \end{cases}$, $f(n) \neq O(n)$ ∧ $f(n) \neq \Omega(n)$, QED.

⊕ iff exist $C = 1$, n_0 st $\sum_{i=1}^{n} \lceil \log_2 i \rceil \leq C \cdot \sum_{i=1}^{n} (\log_2 n + 1) = C \cdot n \cdot (\log_2 n + 1)$

∴ for $n_0 \geq 1$, $\sum_{i=1}^{n} \lceil \log_2 i \rceil \leq n + n \log_2 n = O(n + n \log n) = O(n \log n)$

$\left[\lim_{n \to \infty} \frac{n + n \log n}{n \log n} = \lim_{n \to \infty} \frac{n+1}{n} = 1 \right]$

Thus $\sum_{i=1}^{n} \lceil \log_2 i \rceil = O(n \log n)$

3. Play with Big Data

I read input first and store appeared user_id, product_id in different unordered_set and time in set. By checking the store u,p,t, I could determine wether to put certain peices of information into the data structure thus save some space.

(1) get(u,p,t) / purchased(u): (stored user appeared in get, purchased query)

I use unordered_map<[user_id], unordered_map<[time], unordered_map<[product_id], vector<[User's Data]>>>>

to stimulate a 3D vector with better average time complexity.

For get(u,p,t), I need an average: O(1) and worst O(n)·O(p)·O(q) with 3D-hashmap.

But I'll need O(log n)·O(log p)·O(log q) if I choose to implement in vector.

For purchase(u), I just simply use the same data structure as "get" does. Then I iterated throug all data[u] and put those sale = 1 in a vector and sort them.

An average O(1)·O(p')·O(q') in getting all data (p' ⇒ user→p, q' ⇒ user→q) and O(k log k) in sorting (k, the number of data in answer field).

(2) click(p₁, p₂): (stored user who click p which p in click's query).

I use unordered_map<[product_id], unordered_set<[user_id]>> to implement. Also, the reason I choose unordered_map over map is for its average time complexity O(1) has a better performance than map's O(log p) O(find user_id). And unordered_set also has an average time complexity O(1) so I choosed not to use a vector. Then I use set intersection to get the same element with time complexity O(min{size(p₁), size(p₂)}. Then sort it with O(k log k).

(3) profit(time, theta)

First, when reading input, I put "profit" query's time in a set and remember the minimum "profit" query time.

Then I store profit query in a vector and sort it by largest time to smallest time.

Later, when reading the file, I put those time ≥ minimum_profit_time in a set map<[time], unordered_map<[user_id], [click and purchase number]>> and

iterate it from the largest time → smallest time. So I only need to run through the whole table once. Then I sum up the click / purchase time in set < [user_id], [click, purchase time] >, during every query, I search through first 10 valid user (which in case the smallest in lexicographic order) since map will automatically sort by its key.

And I consider two case with theta = 0 (!= 0) which is also trivial but worth mentioned.

All in all, I use O(n) for going through all user, O(tlogt) for sorting and O(logt)·O(1) for iterating through it.