M1911 Neat Decimals

Author : Alex Tung

2/3/2019

Problem Statment

- Given K, N, B
- Find a solution with K integers $a_1, a_2, \ldots, a_k$ such that…
  - $0 <= a_1 <= \ldots <= a_k$
  - $\text{sum}(a_i) = N$
  - a_i / N is a terminating decimal when written in base K
  - $a_K - a_1$ is minimized

Problem Statment

- Given K, N, B
- Find a solution with K integers $a_1, a_2, \ldots, a_k$ such that…
  - $0 <= a_1 <= \ldots <= a_k$
  - $\text{sum}(a_i) = N$
  - ~~a_i / N is a terminating decimal when written in base K~~
  - $a_K - a_1$ is minimized

Idea

- Trivial
- Minimize $a_K - a_1$
- Distribute N evenly on a1, a2, … , aK
- N / K, …, N / K, N / K + 1, … , N / K + 1

Idea

```
target = N
for (int i = 1; i <= k; i++) {
    a[i] = target / k;
    if (target % k >= k - i + 1) a[i]++;
}
```

Idea

- When will a_i / N become a terminating decimal in base B?
- Hint : a_i / N is terminating in base B if and only if a_i / N * B^l is an integer

Subtask 1

- $B = 2$
- $a_i / N * 2^l$ is an integer
- $a_i / N = t / 2^m$, where t and m are non-negative integers
- $a_i = t * (N / 2^m)$

Subtask 1

- $a_i = t * (N / 2^m)$
- a_i is a multiple of $N / 2^m$!
- Let $x = N / 2^m$, for some non-negative integer m
- What value of x should we choose?

Subtask 1

- We want to minimize $a_K - a_1$ => minimize x

- E.g. K = 3, N = 12, B = 2
- if x = 12 / 2 = 6, solution = {0, 6, 6}, diff = 6
- if x = 12 / 2^2 = 3, solution = {3, 3, 6}, diff = 3 => better

Subtask 1

```
x = N; while (x % 2 == 0) x /= 2;

target = tempN / x;

//Distribute target to k value

for (int i = 1; i <= k; i++) a[i] *= x;
```

- Time Complexity = $O(k + \log_2 N)$

Subtask 2

- B = 10
- a_i / N * 10^l is an integer
- a_i / N = t / $(2^m * 5^r)$, where t, m and r are non-negative integers
- a_i = t * (N / $(2^m * 5^r)$)

Subtask 2

```
while (x % 2 == 0) x /= 2;
while (x % 5 == 0) x /= 5;
```

- Add one line :)

Subtask 3

- ~~B = 2, B = 10~~ B <= 10^6
- a_i / N * B^l is an integer
- a_i = t * (N / ($p_1^{c1} * p_2^{c2} * \dots * p_r^{cr}$))
- p_1, p_2, …, p_r are the prime factors of B
- c1, c2, …, cr are some non-negative integers

Subtask 3

- x = N after removing powers of (prime factors of B)
- Need to find the prime factorization of B :)
- prime factorization of B in O(B)
    - pass this subtask but not subtask 4, 5, 6
- Time complexity = O(B)

Subtask 4, 5

- B <= 10^9
- TLE with O(B) prime factorization => O(sqrt(B)) prime factorization
  - check out Math in OI (I)
- If you distribute wrongly / don't know how to distribute evenly
- You may only pass subtask4 (K = 2)

Subtask 6

- $B <= 10^{18}$
- TLE with O(sqrt(B)) prime factorization => faster prime factorization
  - You may use pollard rho algorithm to find the prime factorization
- There is a easier way

Subtask 6

- We actually don’t need to find out the prime factorization
- Need to remove prime factors of B from N

- Repeat N’ = N / gcd(N, B), until gcd(N, B) = 1 -> x = N’

Subtask 6

- Repeat N’ = N / gcd(N, B), until gcd(N, B) = 1 -> x = N’
- Everytime you remove some powers of prime factors of B
- When gcd(N, B) = 1, N no longer contains prime factors of B
- Time complexity = $O((\log_2 N)^2)$

M1912 Visiting Museums

Charlie Li
2019/03/02

Problem

- You given a_i, b_i and c_i of N museums.
- You have M dollar.
- To visit museum i, you need to pay c_i dollar.
- Suppose museum i is the x^th museum that you are visiting, then the interesting value of visiting it is a_i-(x-1)*b_i
- Find a sequence to visit some of the museums such that the total interesting value is maximized.

Subtasks

- $1 \le N,M \le 500$
  $1 \le a_i \le 10^9$
  $0 \le b_i \le 10^9$
  $0 \le c_i \le M$
- Subtask 1: $N \le 10$
- Subtask 2: $N \le 20$
- Subtask 3: $N \le 100$
- Subtask 4: All $b_i = 0$
- Subtask 5: All b_i are equal.

Subtask 1

- N <= 10 is small here, so we can just use exhaustion to check all the possible choices for the sequence.

- O(n!)

- How?

Subtask 1

- You can use std::next_permutation in C++ to help you.

- Or you can write a recursion yourself using the knowledge from “recursion, divide and conquer”

* Remember using 64-bit integer

Subtask 2

- N <= 20
- This time N is larger but not too large.
- Therefore, we can use bitwise DP (appear in DP(II)) this time

Subtask 2

- Define the state
- dp[i] = maximum total interesting value obtain by visiting the museums encoded by i in any order, if you do not have enough money then set dp[i] = 0.
- What does it mean by the museums encoded by i?

- Let's see it using examples:
  i = 5 = 101_2 => visit museum 1 and 3
  i = 6 = 110_2 => visit museum 2 and 3

Subtask 2

- Transition is

  if (i & (1 << j)) = 0 and cost(i|(1 << j)) <= M then
  dp[i|(1 << j)] = max(dp[i|(1 << j)], dp[i] + a[j] – count_ones(i) * b[j])

Subtask 2

- Time Complexity O(N * 2^N)

Subtask 4

- This is exactly the same as the knapsack problem appears in DP(I)
- Take a look to the powerpoint of DP(I) if you want to know more

Observations

- Upon looking at subtask 4, you may find this problem is similar to Knapsack.

- In fact, you can use Knapsack to solve this problem but with one more dimension (to record how many museums visited before)

Full solution?

```
for (int i = 1; i <= n; i++) {
  for (int j = 0; j <= m; j++) {
    for (int k = 1; k <= i; k++) {
      dp[i][j][k] = max(dp[i-1][j][k], dp[i-1][j-c[i]][k-1] + a[i] – (k - 1) * b[i]);
    }
  }
}
```

This cannot pass sample 5, giving v_max = 40

Full solution?

- Why this is not true?
  - Because the order for knapsack is not important
  - Knapsack just follow the increasing order of index
  - But here going to museums in ascending order of index may not be optimal

- What order should we choose?
  - Suppose we have decided to go to both museum i and j.
  - We need to pay the same amount no matter what order.
  - However the interesting value may not be the same
  - $a_i + a_j – b_j$ <? >? $a_j + a_i – b_i$

Full solution?

- We can see that to maximize the total interesting value, we must go to the museum with larger b (since we will lose more if we go there later)

- It is easy to see that this relation is transitive.
  i.e. $b_i > b_j$ and $b_j > b_k$ implies $b_i > b_k$

- So, we can use greedy to find the order.

Full solution?

- We first sort the museums in descending b’s.
- And then run the previous knapsack with path tracing.

- O(N^2M)

- Then done?

Full solution?

M1912 - Visiting Museums *2019 Mini Competition I*	C++11	Runtime Error	55 *Score*

Full Solution

- The size of a 500*500*500 array of long long =500*500*500*8/1024/1024 = 953MB > 256MB

- Use rolling array please ☺

* Rolling array is just used for the calculation part (the array of long long), to trace the part, you can still use a 500*500*500 Boolean array

M1913 Electricity

Charlie Li

2019/03/02

Problem

- Given a rooted tree (why is it a tree? What is its root?) with 2 kinds of nodes.
  - Electrical appliances: Charge some current on its ancestors
  - Power strip: Can afford certain current to be charged by its subtree

- Select a maximized subset of Electrical appliances such that on every power strip node, the total current charged by its subtree does not exceed its capacity.

Subtasks

- $1 \leq N,M \leq 500000$
  $1 \leq c_i \leq 10^{18}$
  $1 \leq r_i \leq 10^9$
  $0 \leq f_i \leq N$
- Subtask 1: $N,M \leq 20$
- Subtask 2: $N,M \leq 500$
- Subtask 3: $N,M \leq 5000$
- Subtask 4: The structure of the tree is random
- Subtask 5: No additional constraints

Subtask 1

- It is good to think of (or even write some) naïve solutions first.
  - This can help you to test on small test cases.

- Here we can use a brute force solution to find the answer.
- Just enumerate every subset of those electrical appliances and then use a dfs to check if that subset is valid.

- O((N+M) * 2^M)

Observations

- Suppose every electrical appliances is plugged into a power strip which is plugged into the mains, how can we find the answer?

- Easy, only turn on the electrical appliances which uses the least current until it reach the capacity.
- In other words, we can consider the electrical appliances we have not used (which are with high current) being unplugged or disconnected or destroyed or …
- They will never be turned on since we can always find a better substitute for it.

Observation

- After we have deleted the electrical appliances that we do not want, we can then also remove the power strip and directly plug the remaining electrical appliances one level up.

- We can repeat this process until we reach the mains (i.e. root of the tree)

Subtask 3

- Actually, the previous described algorithm is a greedy algorithm. (see more on “Greedy algorithms”).

- A simple way to implement it is to store a sorted list of electrical appliances that plugged into the power strip for every power strip.
- Once we move one level up, we sort it once.
- And then take the electrical appliance with low current one level up until reach the capacity, and so on…

Subtask 3

- We will do at most M sorting, and every sorting is O(N lg N)
- So the overall time complexity of this algorithm is O(MN lg N).
- A bonus to this algorithm is that it will also pass subtask 4.
- The reason is that in subtask 4, the structure of the tree is random and so the tree is not tall, expected depth is O(lg N).
  - In this case, the above described algorithm runs in O(N lg^2 (M+N)) time.

Full solution

- Actually, at any time, what we need to do is just extract_min and merge some sorted arrays.
- We can replace the sorted arrays by priority queues!
- Though, direct substitution gives us the same time complexity (O(MN lg N) for subtask 3 and O(N lg^2 (M+N)))
- There is a trick which can improve this a lot.

Full solution

- When merging a priority queue A to another priority queue B, the naïve way is to repeatly pop an element from A and push it into B.

- This is find if B is larger than A but this is waste of time if A is larger than B (or even B has no element).
- In the later case, actually we can first swap two priority queues and then perform the naïve merging.

Full solution

- Base on the above algorithm, once an element x is being popped and pushed into another heap, the new heap must be at least twice as large as the original one.

- Therefore we have x will only be popped and pushed O(lg N) times
- Every time when it is pushed or popped it is O(lg N)
- There are total O(N) elements.
- So the overall time complexity would be O(M + N lg^2 N).

Full solution 2

- You can implement a binomial heap yourself and improve the merging time to O(lg N), then the total time complexity would be O((M+N) lg N)

* You can find more information on Wikipedia
* Don’t waste your time during contest to do this unless it is necessary.

Full solution 3

- We can also use segment tree (appears in DS(III)) to solve the same problem in O((M+N) lg N), we can replacing the tree by the array of nodes in pre-order, than a subtree of a node would be a continuous segment.

- To remove an element, we can do a range query and than a point update.
- And we don’t need to do the merging.
- You may find out more information on DS(III)

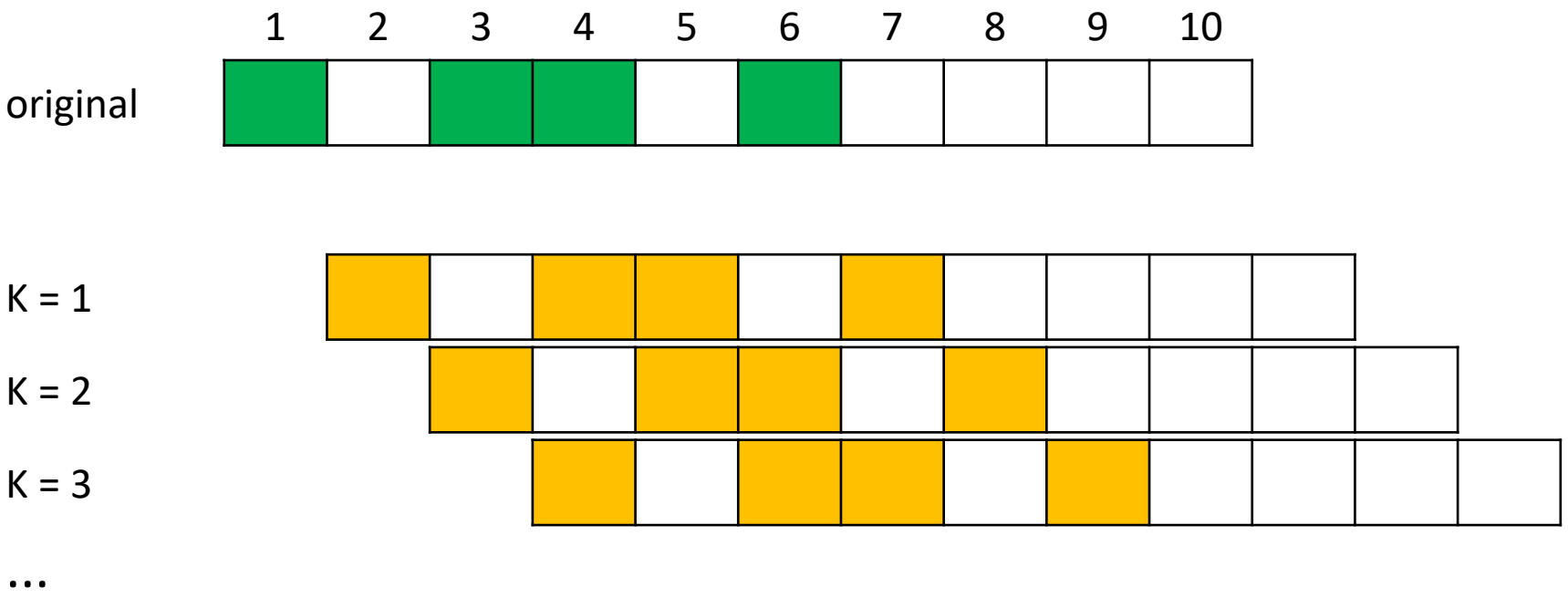
M1914 Nuclear Fusionist

Lau Chi Yung

2 March, 2019

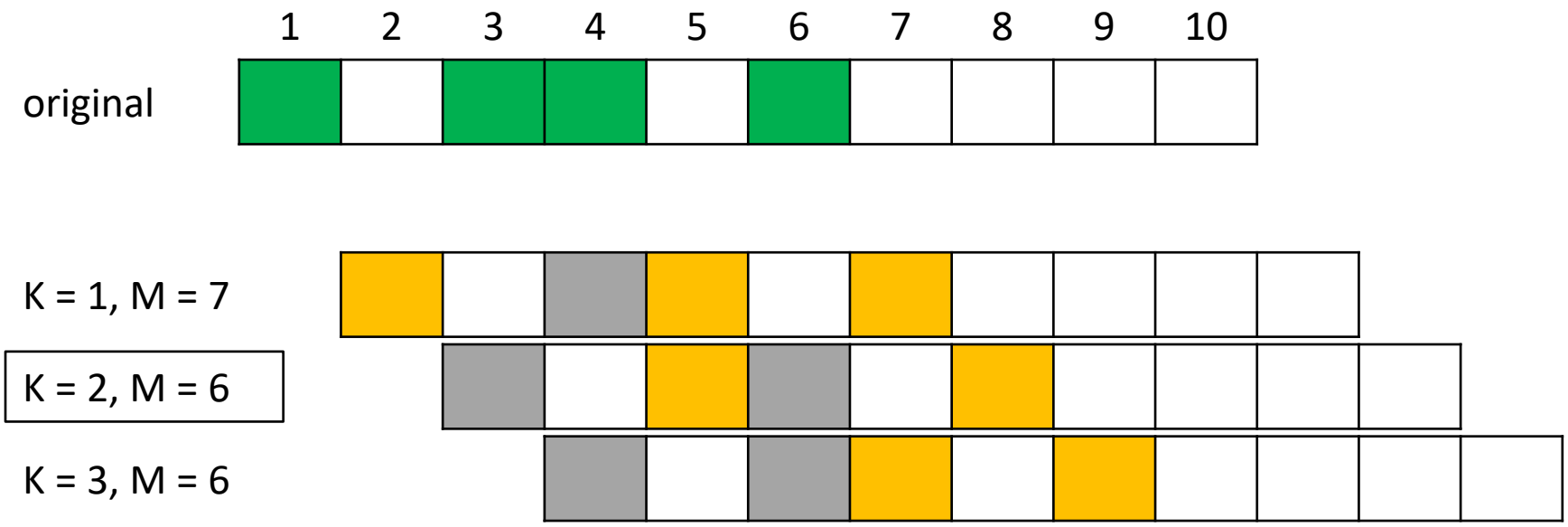
Sample

```
4
1 3 4 6
```



Sample

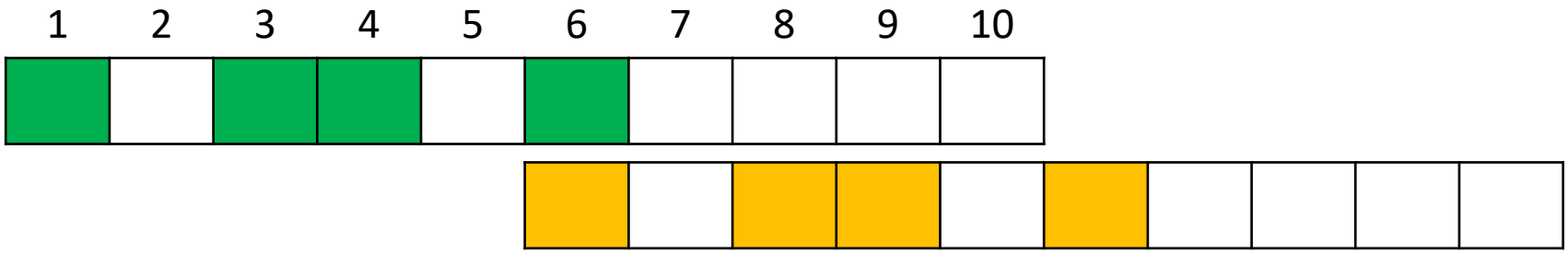
```
4
1 3 4 6
```




```
First minimize M
then minimize K
```

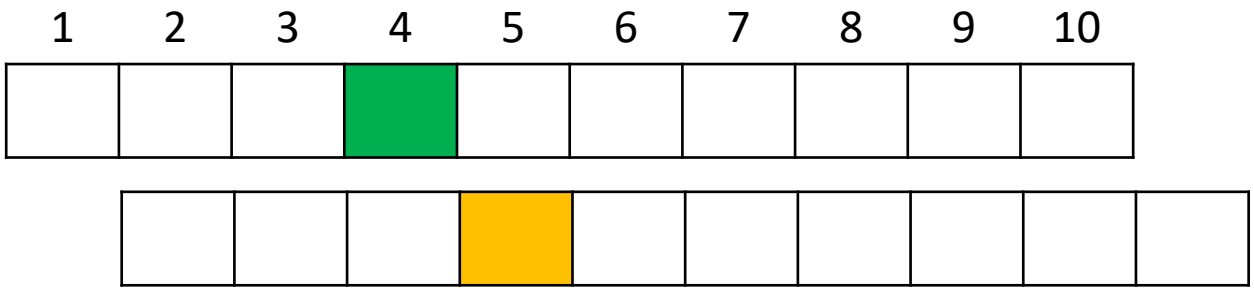
Observation

- Optimal M is at most 2N – 1 when N > 1



N = 1

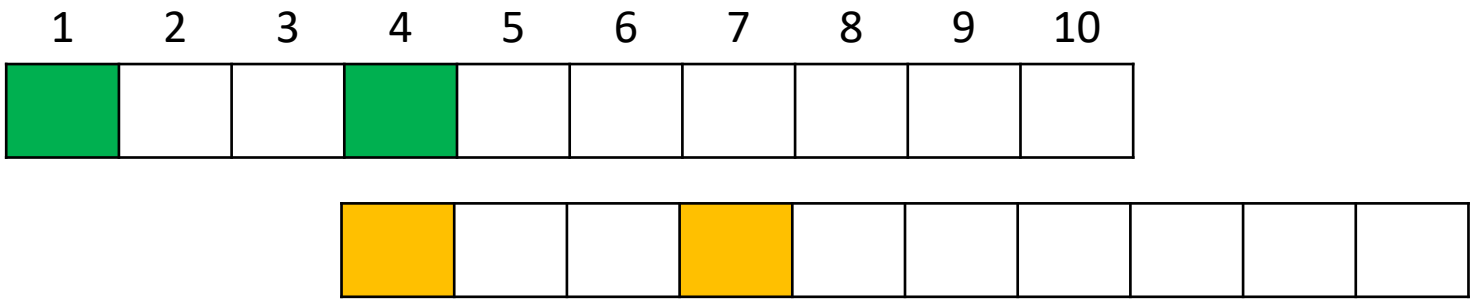
```
1
4
```




```
Optimal solution:
K = 1
M = 2
```

N = 2

```
2
1 4
```



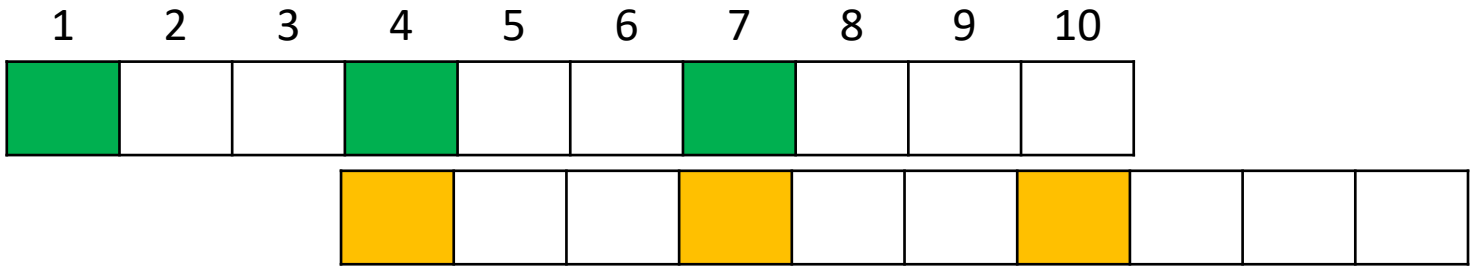
Optimal solution:

$K = P_2 - P_1 + 1$

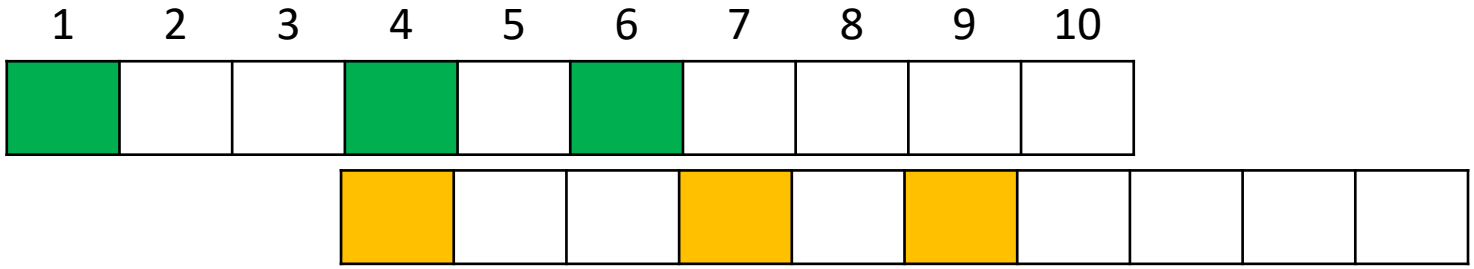
$M = 3$

N = 3

$P_3 - P_2 = P_2 - P_1$

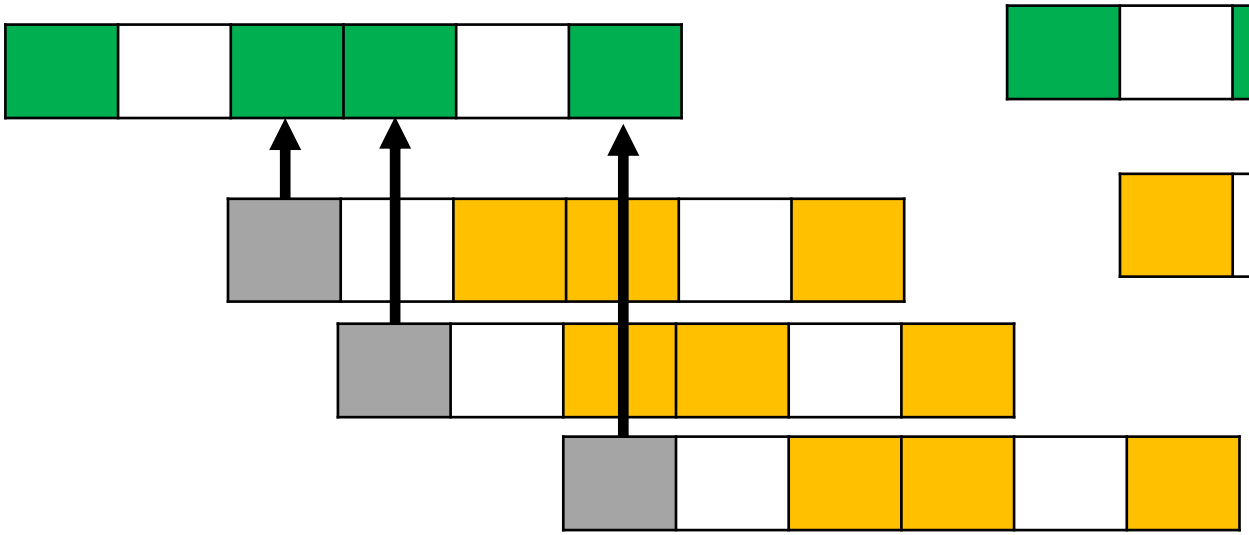


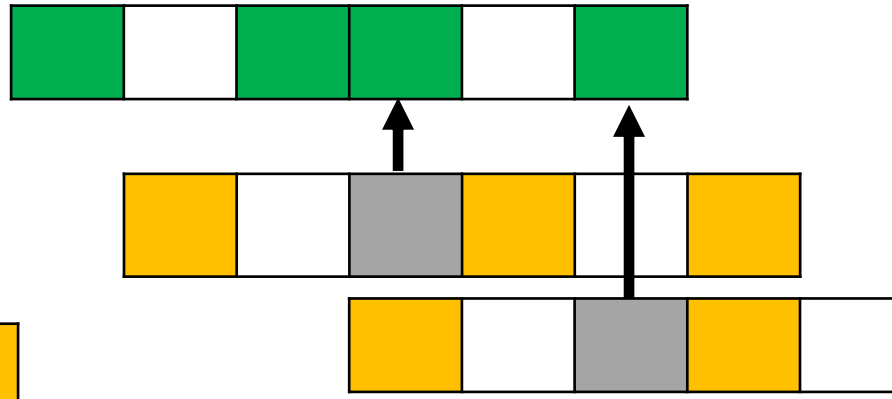
Otherwise



N <= 30, N <= 200

- Exhaust all possible K? $1 <= K <= 10^9$
- In fact, we only need to exhaust N(N-1)/2 of them:





...

N <= 30, N <= 200

- For each of those $O(N^2)$ possible K, compute M with
  - two simple nested for-loops:
    $O(N^2) * O(N^2) = O(N^4)$
  - for-loop + binary search:
    $O(N^2) * O(N\log N) = O(N^3 \log N)$
  - two pointers:
    $O(N^2) * O(N) = O(N^3)$
- Output:
  bubble sort $P_1, P_2, \ldots, P_N, P_1+K, P_2+K, \ldots, P_N+K$
  output only the unique ones

P_i <= 1000

- For each 1 <= K <= 1000, compute M
- Since 1 <= P_i + K <= 2000, we can also use counting array when computing M

N <= 1000

- Given a value of K, we need to compute M quickly
- Let S be a data structure containing
  $P_j - P_i$ for $1 <= i < j <= N$
- $|S| = N(N-1)/2$
- Given a value of K,
  M = 2*N – number of values in S that is equal to K
- S can be hash table, binary search tree, sorted array

N <= 1000

- sorted array

3	4	6	7	7	7	12	12	13	14
1	1	1	1	2	3	1	2	1	1

- Overall time complexity: $O(N^2 \log N)$