Arithmetic Sequence

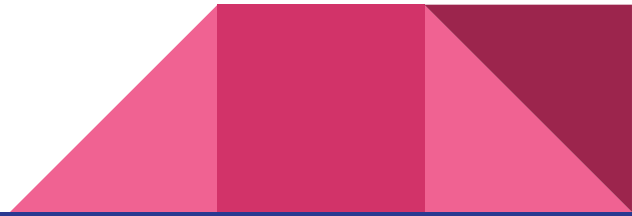
Problem setter: Alex Poon

Problem Statement

Given N and R, output a sequence S[1..N] such that :
1. There is NO arithmetic sub-sequence of length 3
2. S[i] is distinct and 1 <= S[i] <= R

Sample Input:
5 7

Sample Output:
1 7 6 2 4

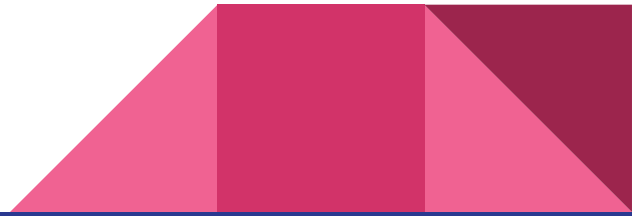


Example of valid / invalid sequence

E.g. N = 6 and R = 10

Valid Sequence :
3 2 4 8 10 9
1 3 6 10 5 2
…

Invalid Sequence:
1 3 8 7 5 4 {1, 3, 5} is an AS
1 2 4 8 16 32 S[5], S[6] > R
...



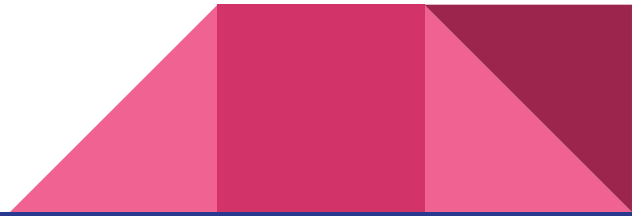
Subtask

	Points	Constraints	
1	13	1 <= N <= 10	R = 2,000,000
2	15	1 <= N <= 10	R = N
3	25	1 <= N <= 250	R = 4,000
4	21	1 <= N <= 10000	R = 2,000,000
5	26	1 <= N <= 10000	R = N

Statistics

Attempt : 57
Attempted Mean : 20.083 (lowest mean among 4 questions)
Attempted std dev: 22.394

100 : 3
53 : 1
28 : 19
13 : 20



Verify a Sequence

How to verify if a sequence is valid ?

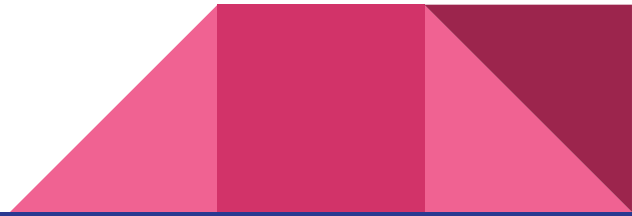
We need to check 3 things
1: Are they distinct
2: Are there any number out of the range [1, R]
3: Are there AS subsequence

Verify a Sequence

Check 1: use an array to record have a number appeared in the sequence

```
for (int i = 1; i <= n; i++)
{
        if (appear[S[i]] == true) not valid;
        appear[S[i]] = true;
}
```

Check 2: easy……



Is there AS subsequence?

To check 3

Trivial method : Exhaust all possible sub-sequence with length = 3 then check it

e.g. Original sequence = 1 8 7 3

1 8 7 3 1 8 7 3
1 8 7 3 1 8 7 3

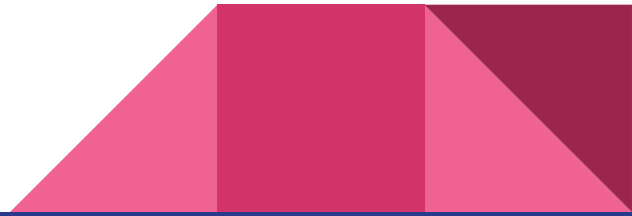
If none of the subsequence is an A.S, then the whole sequence is valid

Verify a Sequence

Code:

```
for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
                for (int k = j + 1; k <= n; k++)
                        if (S[k] - S[j] == S[j] - S[i]) not valid;
```

Time Complexity: O(N * N * N)



Verify a Sequence

Faster method : Just enumerates 2 integers

e.g. Original sequence = 1 8 7 3 5

1 8 7 3 5 -> find if there 8 + (8 - 1) = 15 in 1 8 { 7 3 5 }
1 8 7 3 5 -> find if there 7 + (7 - 1) =13 in 1 8 7 { 3 5 }
1 8 7 3 5 -> find if there 3 + (3 - 1) = 5 in 1 8 7 3 { 5 }
...
1 8 7 3 5 -> find if there 3 + (3 - 7) = -1 in 1 8 7 3 5

If the answer of all query is no, then it is a valid sequence

Verify a Sequence

1 8 7 3 5 -> find if there 8 + (8 - 1) = 15 in 1 8 { 7 3 5 }
1 8 7 3 5 -> find if there 7 + (7 - 1) =13 in 1 8 7 { 3 5 }
1 8 7 3 5 -> find if there 3 + (3 - 1) = 5 in 1 8 7 3 { 5 }

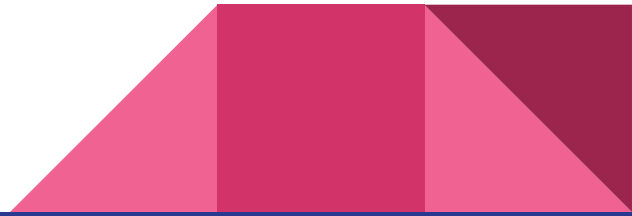
How to find if there x in the subsequence??
We use an array to precompute the position of each value first

E.g. S = {1, 8, 7, 3, 5}, we have an array P = {1, 0, 4, 0, 5, 0, 3, 2}
It denotes 1 appear in ***position 1*** in S
2 does not appear in S
3 appear in ***position 4*** in S...

Verify a Sequence

Code :

```
for (int i = 1; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) {
                x = A[j] + (A[j] - A[i]);
                if (P[x] > j)  not valid;
        }
}
```



Subtask 1

Subtask 1 is an easy subtask. It let candidate get the “feel” of this problem.

Note that R is large and N is small

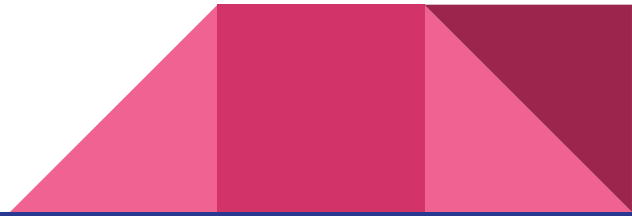
It is easy to find 10 number within 2,000,000 to form a valid sequence

Expected solution : Hardcode / random output / create a general formula

e.g. N random number

e.g. 1 2 4 8 16 32 … (general formula T[i] = 2 ^ i)

e.g. 1 2 10 20 100 200…



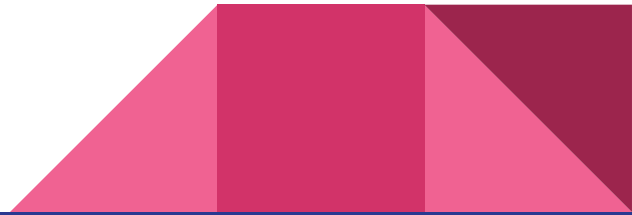
Subtask 2

Note that 1 <= N <= 10 and R = N

As R is small this time, we can’t generate a valid sequence by general formula or random output

However, there are at most P(N, R) = N! = 3628800 possible sequence in total. So we can exhaust all possible sequences and check if it is a valid sequence

Expected Solution: Exhaustion / do it by hand (only 10 cases but not easy actually)
Time complexity : O(N! * checkTime) = O(N*N*N!)

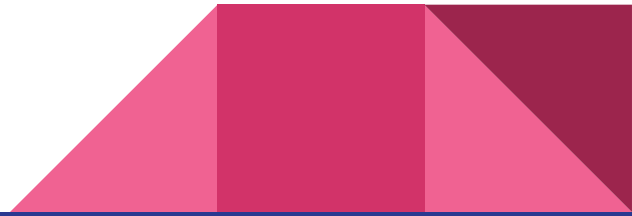


Subtask 3

In subtask 3, 1 <= N <= 250 and R = 4000

N is large so we can’t use exhaustion this time. We need to focus on how to construct a valid sequence

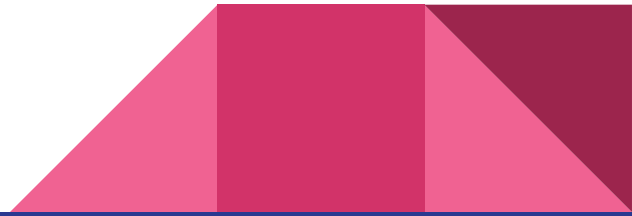
In subtask 1, we can generate a valid sequence by some mathematical method
But for subtask 3 onward, we need some computational method to generate it



Subtask 3

If we have a valid sequence with length N - 1, we can construct a valid sequence with length N by inserting a number that the end of it.

E.g.
We know that {1, 5, 3, 2, 6, 4} is an valid sequence
We can try to insert a number at the end of it
{1, 5, 3, 2, 6, 4, 1}, {1, 5, 3, 2, 6, 4, 2}, {1, 5, 3, 2, 6, 4, 3} …
Until you find a number such that it is valid to insert it
{1, 5, 3, 2, 6, 4, 8} is valid

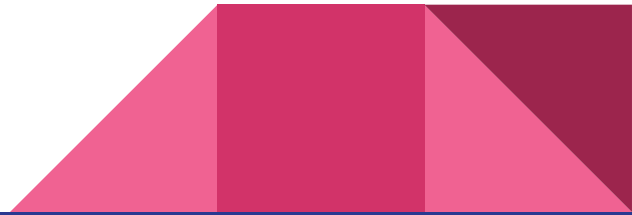


Subtask 3

As we know {1} is a valid sequence

So, we may expand the sequence {1} to length N by the same method

{1} -> {1, 1} fail, {1, 2} valid
{1, 2} -> {1, 2, 1} fail {1, 2, 2} fail {1, 2, 3} fail {1, 2, 4} valid
{1, 2, 4} -> {1, 2, 4, 1} fail…… {1, 2, 4, 5} valid
{1, 2, 4, 5} -> {1, 2, 4, 5, 1} fail …… {1, 2, 4, 5, 10} valid



Subtask 3

Actually, we don’t need to check from 1 again each time

e.g.
We have {1, 2, 4, 5} now
We can check start from {1, 2, 4, 5, 6} instead of {1, 2, 4, 5, 1}
Why?
We know {a, b} is invalid then {a, ?, ?, ? ,? .... b} is invalid
We know {a, b, c} is invalid then {a, ?, ?, ? b, ?, ?, ?, ?, c} is invalid

Finally, we will get a sequence starting with 1, 2, 4, 5, 10, 11, 13, 14……

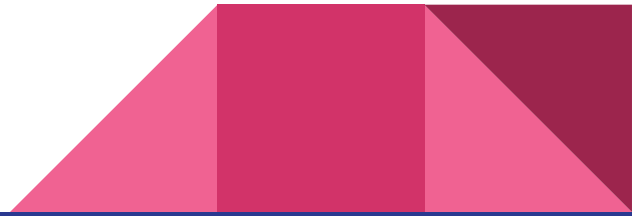
Subtask 3

Using this method, the 250th number of the sequence generated = 3269
the 10000th number of the sequence = 1679657

Therefore, the sequence is sufficient to pass subtask 3 and 4

However, What is the time complexity to generate the sequence?
O((N + R) * CheckTime) = O(R * N * N)
Which is too slow for subtask 4

So this algorithm can only pass subtask 3



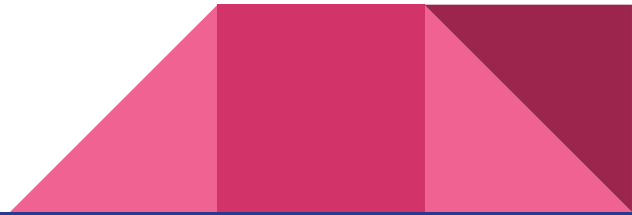
Subtask 4 (briefly)

Maybe the observation is too trickly, so let discuss it briefly

Consider the same sequence in subtask 3
1, 2, 4, 5, 10, 11, 13, 14……

Note that {10, 11, 13, 14} = {9 + 1, 9 + 2, 9 + 4, 9 + 5}
i.e. S[5] = S[1] + 9; S[6] = S[2] + 9; S[7] = S[3] + 9; S[8] = S[4] + 9;

Note that {4, 5} = {3 + 1, 3 + 2}
i.e. S[3] = S[1] + 3; S[4] = S[2] + 3;



Subtask 4 (briefly)

Seems that we can generate S[1..2 * N] by S[1..N], let verify it.

Let we have {1, 2} initially
We can generate {{1, 2} concatinate {1 + 3, 2 + 3}} -> {1, 2, 4, 5}
{1, 2, 4, 5} concatinate {1 + 9, 2 + 9, 4 + 9, 5 + 9} -> {1, 2, 4, 5, 10, 11, 13, 14}
{1, 2, 4, 5, 10, 11, 13, 14} concatinate {1 + 27, 2 + 27 … 14 + 27} ->
{1, 2, 4, 5, 10, 11, 13, 14, 28, 29, 31, 32, 37, 38, 40, 41}

i.e. If we have S[1..2 ^ i], we can generate S[1..2 ^ (i + 1)]
by concatinate T[1..2^i] where T[j] = S[j] + 3^i

Time Complexity : O(N)

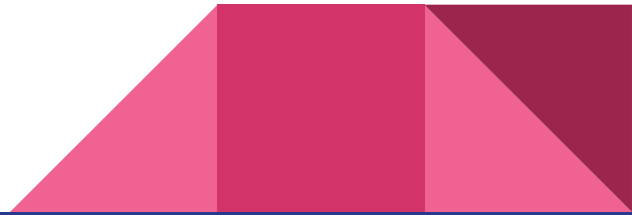
Full solution

Constraint : N = R and 1 <= N <= 10000

This time we can’t use the greedy method to generate the sequence as R = N

In order to solve this subtask, we should analyze the property of an A.S

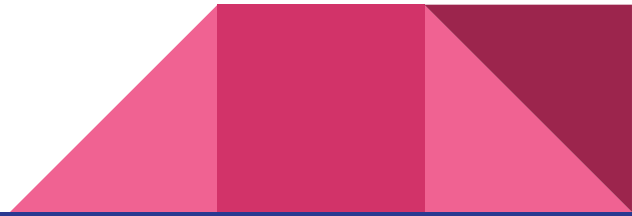
A.S. can be represent in form of {x, x + d, x + 2d, x + 3d…} where d is called the common different



Full solution

Note that if d is even, the sequence will have same parity
e.g. {1, 3, 5, 7, 9, 11} all of them are odd number
e.g. {2, 6, 10, 14} all of them are even number

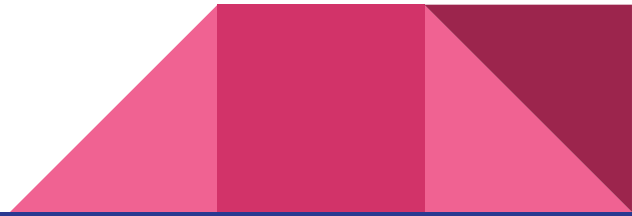
If d is odd, the sequence will have different parity to its adjacent terms
e.g. {1, 4, 7, 10, 13} odd, even, odd, even, odd
e.g. {2, 11, 20, 29} even, odd, even, odd



Full Solution

Therefore, an AS with length 3 should be in form of
{odd, odd, odd}, {odd, even odd}, {even, odd, even} or {even, even, even}
But not {odd, odd, even}, {odd, even, even}, {even, odd, odd} or {even, even, odd}

Main Observation:
When we construct a valid sequence
If we put all odd numbers in the first half, all even numbers in the second half
There will be NO A.S crossing the first half and second half

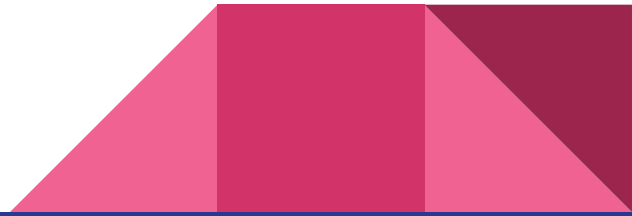


Full Solution

e.g. When need to construct an sequence with length = 10

We can put 1, 3, 5, 7, 9 to the first half, 2, 4, 6, 8, 10 to the second half
we have {1, 3, 5, 7, 9, 2, 4, 6, 8, 10}

Then we only need to concern about the sequence {1, 3, 5, 7, 9} and {2, 4, 6, 8, 10} seperately as we know there must be NO A.S. formed acrossing these two sets



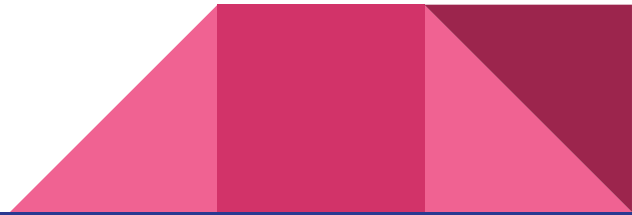
Full Solution

The problem reduce to build a valid sequence with {1, 3, 5, 7, 9} and {2, 4, 6, 8, 10}

Literally, build a valid sequence with {1, 3, 5, 7, 9} or {2, 4, 6, 8, 10} is same as build a sequence with {1, 2, 3, 4, 5}

e.g. {1, 4, 5, 2, 3} is valid then {2, 8, 10, 4, 6} or {1, 7, 9, 3, 5} must be valid as well

Therefore, we can assume they are {1, 2, 3, 4, 5} and use the same approach to break down the sequence until it is small enough



Full solution

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
{1, 3, 5, 7, 9} + {2, 4, 6, 8, 10} // “+” mean concatinate
{1, 5, 9} + {3, 7} + {2, 6, 10} + {4, 8} //assume they are 1-5, break down it again
{1, 9} + {5} + {3, 7} + {2, 10} + {6} + {4, 8} // break down until the length of it <= 2
{1, 9, 5, 3, 7, 2, 10, 6, 4, 8} -> valid sequence

We divide the orginal problem: Constructing S[1 .. N]
Into sub-problem : Constructing S[1..N / 2]
This technique call “Divided and Conquer”

Time Complexity : O(N)

Full Solution

In Short, the algorithm is:

1. Assign 1..N to S[1..N] repectively

2.1. If the length of the sequence we are building <= 2, there must be no A.S. in this sequence -> complete

2.2. Else, we put the “odd” number to the 1st half, “even” number to the 2nd half and then we deal with the 1st half and 2nd half seperately.

Full Solution

How to code? Recursion

```
void solve(int l, int r) // solve(l, r) mean we are trying to make S[l] ... S[r] are valid sequence
{
    // Base Case: if the length of the sequence <= 2, stop dividing it
    int len = r - l + 1;
    if (len <= 2) return;
    int B[10005], w = 0;

    // Conquer
    for (int i = l; i <= r; i += 2) B[w++] = A[i]; // put the odd to the first half
    for (int i = l + 1; i <= r; i += 2) B[w++] = A[i]; // put the even to the second half
    for (int i = l; i <= r; i++) A[l] = B[i - l];

    // Divide
    int mid = (l + r) / 2;
    if (len % 2 == 0) { solve(l, mid); solve(mid + 1, r); }
    else if (len % 2 == 1) { solve(l, mid + 1); solve(mid + 2, r); }
}
```

Conclusion

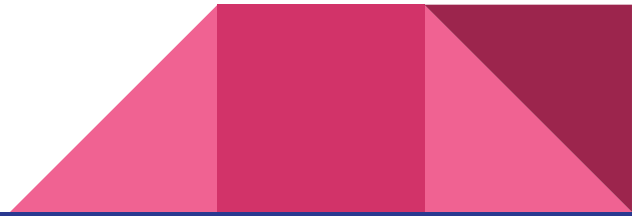
Subtask 1 tests whether you read the constraint carefully

Subtask 2 tests if you can write an exhaustion about permutation generating

Subtask 3, 4 tests if you familiar with greedy algorithm and strategy to construct a sequence

Subtask 5 tests can you make some important observation about the problem and tests whether you familiar with “Divide and Conquer” method.

(p.s. D&C is taught by me on team training last year xd)



Challenge

If the constraint is 1 <= N <= 100,000 and R = N for all subtasks

How can we write a program to verify if a sequence is valid in O(N lg N)?

tag: Data structure (III), (String algorithm/data structure (II))