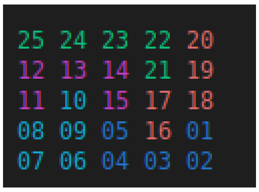
GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 4 Report

Emirkan Burak Yılmaz
1901042659



```
25 24 23 22 20
12 13 14 21 19
11 10 15 17 18
08 09 05 16 01
07 06 04 03 02
```

1. SYSTEM REQUIREMENTS

Q1. A recursive function which searches given pattern string in another given bigger string. The function should return the index of nth occurrence of the query string. Also returns -1, if the pattern doesn't occur in the big string or the number of occurrences is less than desired occurrence.

Q2. A recursive function which takes a sorted integer array and finds recursively the number of items in the array between two given integer values.

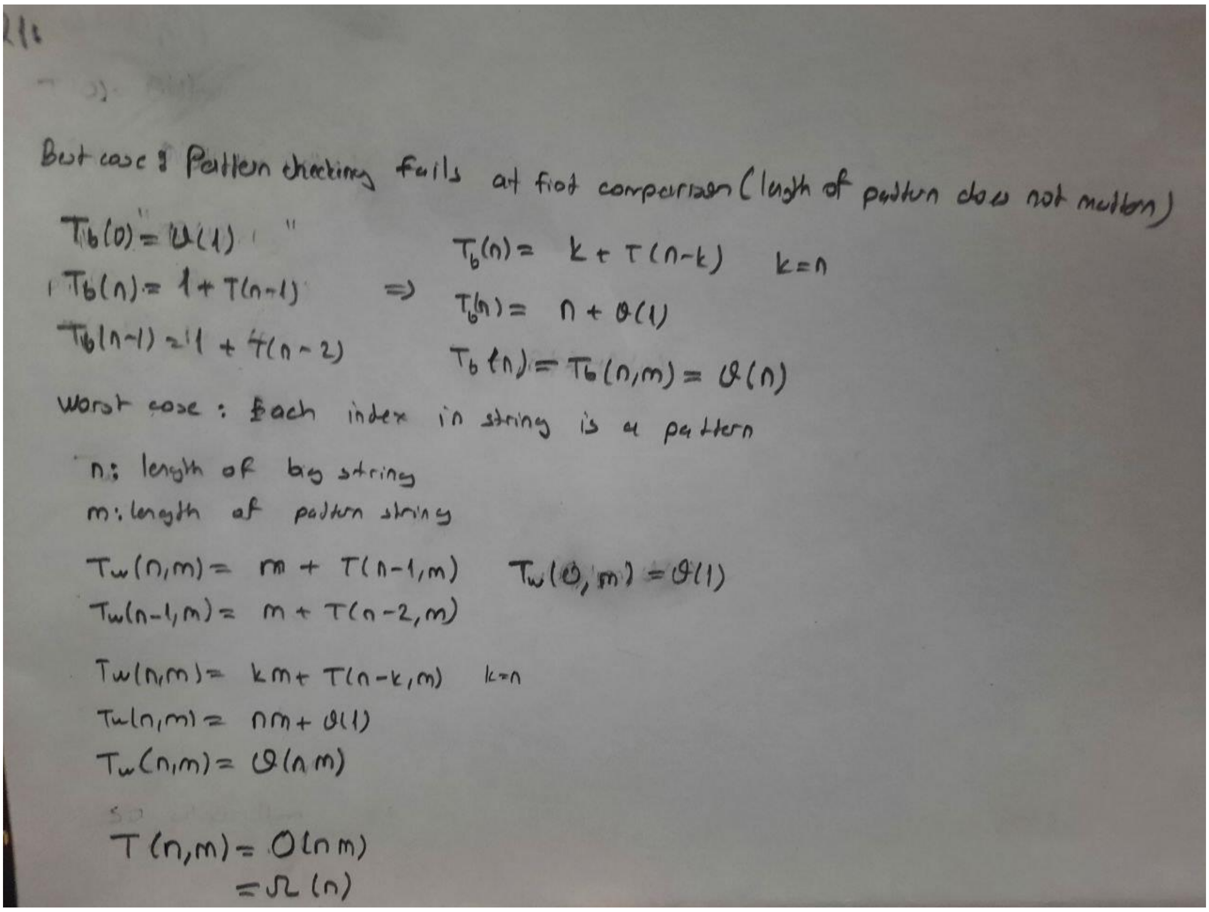
Q3. A recursive function which takes an unsorted integer array and an integer value. By using these two parameters finds contiguous subarray/s that the sum of its/their items is equal to a given integer value.

Q4. Explain the output of the given recursive function and how it works in detail. Analyze the time complexity of this function by analyzing the number of multiplication operations

Q5. A recursive function which takes an integer value *L*. Creates a 1-D array of empty blocks of length *L*. Finds all the possible configurations to fill this array with colored-blocks. Length of colored blocks can be vary from 3 to *L*. There should be at least one empty block between each consecutive colored-block. Print all the steps of the algorithm.

Q6. Given an *N*x*N* matrix of empty blocks, propose a recursive solution that calculates all the possible combinations to fill this matrix by using snakes of length exactly *N*. Snakes can bend. Print all the possible configurations for filling the matrix.

2. ALGORITHM ANALYSIS



But case : Pattern checking fails at fiot comparison (lngth of pattern dow not matten)

$T_b(0) = \Theta(1)$

$T_b(n) = 1 + T(n-1)$

$T_b(n-1) = 1 + T(n-2)$

$\Rightarrow$

$T_b(n) = k + T(n-k) \quad k = n$

$T_b(n) = n + \Theta(1)$

$T_b(n) = T_b(n,m) = \Theta(n)$

Worst case : Each index in string is a pattern

n: length of big string

m: length of pattern string

$T_w(n,m) = m + T(n-1,m) \quad T_w(0,m) = \Theta(1)$

$T_w(n-1,m) = m + T(n-2,m)$

$T_w(n,m) = km + T(n-k,m) \quad k=n$

$T_w(n,m) = nm + \Theta(1)$

$T_w(n,m) = \Theta(nm)$

$T(n,m) = O(nm)$

$= \Omega(n)$

Q2:

Best case: Elements are finded in constant time (at most two step)

$T_b(n) = \overbrace{2C(n)}^{\text{time complexity of binary search algorithm}} + \Theta(1)$

$C_b(n) = \Theta(1)$

$T_b(n) = \Theta(1)$

Worst case: Elements are finded in logarithmic time.

$T_w(n) = 2C(n) + \Theta(1)$

$C_w(n) = 1 + C_w(n/2) \qquad C(1) = \Theta(1)$

$C_w(n) = k + C_w(n/2^k) \qquad 2^k = n$

$k = \log_2 n$

$C_w(n) = \log n + \Theta(1)$

$C_w(n) = \Theta(\log n)$

$T_w(n) = 2\,\Theta(\log n) + \Theta(1)$

$T_w(n) = \Theta(\log n)$

$T(n) = O(\log n)$

$= \Omega(1)$

$= \Omega(n)$

Q3:

$T(n) = n + T(n-1) \qquad T(0) = \Theta(1)$

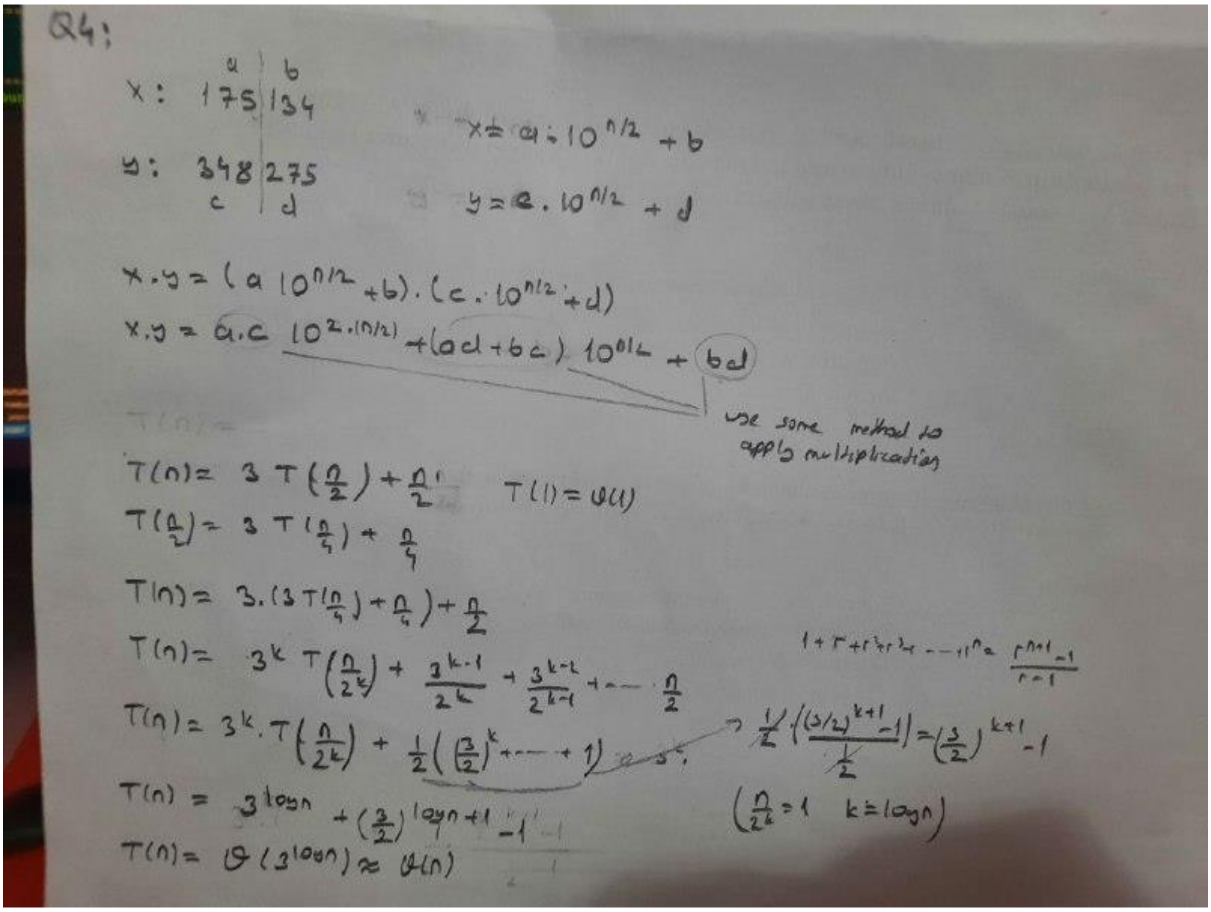
$T(n-1) = n - 1 + T(n-2)$

$T(n) = (n) + (n-1) + T(n-2)$

$T(n) = nk - \frac{k.(k-1)}{2} + T(n-k) \qquad k = n$

$T(n) = n^2 - \frac{n.(n-1)}{2} + \Theta(1)$

$T(n) = \Theta(n^2)$



Q4:

$$\begin{array}{cc|c} & a & b \\ x: & 175 & 134 \\ y: & 348 & 275 \\ & c & d \end{array}$$

$x = a \cdot 10^{n/2} + b$

$y = c \cdot 10^{n/2} + d$

$x \cdot y = (a\, 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d)$

$x \cdot y = a \cdot c\ 10^{2 \cdot (n/2)} + (ad + bc)\ 10^{n/2} + \boxed{bd}$

use some method to apply multiplication

$T(n) = 3\,T\left(\frac{n}{2}\right) + \frac{n}{2} \qquad T(1) = \theta(1)$

$T\left(\frac{n}{2}\right) = 3\,T\left(\frac{n}{4}\right) + \frac{n}{4}$

$T(n) = 3 \cdot \left(3\,T\left(\frac{n}{4}\right) + \frac{n}{4}\right) + \frac{n}{2}$

$T(n) = 3^k\,T\left(\frac{n}{2^k}\right) + \frac{3^{k-1}}{2^k} + \frac{3^{k-2}}{2^{k-1}} + \dots \frac{n}{2}$

$1 + r + r^2 + r^3 + \dots + r^n = \frac{r^{n+1} - 1}{r - 1}$

$T(n) = 3^k \cdot T\left(\frac{n}{2^k}\right) + \underbrace{\frac{1}{2}\left(\left(\frac{3}{2}\right)^k + \dots + 1\right)}$

$\rightarrow \cancel{\frac{1}{2}}\left(\frac{(3/2)^{k+1} - 1}{\cancel{\frac{1}{2}}}\right) = \left(\frac{3}{2}\right)^{k+1} - 1$

$\left(\frac{n}{2^k} = 1 \quad k = \log n\right)$

$T(n) = 3^{\log n} + \left(\frac{3}{2}\right)^{\log n + 1} - 1$

$T(n) = \theta(3^{\log n}) \approx \theta(n)$

3. PROBLEM SOLUTION APPROACH

Q1. Each character in the big string is a possible starting position for pattern string. So, start with start index (initially 0 and increases each recursive call) and compare if the characters are same as pattern. When there is a conflict stop comparing and skip to next big string index as starting index of pattern. If there is no conflict occurs during comparison of, it means an occurrence founded at current start index. Check if the desired occurrence is 1, if so return that starting index, otherwise decrease desired occurrence by one and continue searching recursively from next index to the end of the big string.

Q2. Find the indexes of the given two integer values. Since given array is sorted use binary search algorithm to find the indexes. There should be two binary search method. First one should find the index of the first occurrence of target value and the second one last occurrence. By using these two-method function can determine first occurrence of the given smaller value and the last occurrence of the larger value. The number of items in the two values will be difference of these two indexes minus one. Of course, if any of the given two item cannot find just return -1.

Q3. Each index of the given array is possible subarray starting position and we cannot be sure if the sum of sub array is equal or not the given integer value without checking all the indexes. So, start from first index of the array and check if there is a subarray which starts at current index. Add all the values and check if it's equal to given integer value. If is equal report the first and last index of the subarray and continue to check to end of the array. After comparison is done for current index pass next index by recursively and continue like to end of the array.

Q4. The algorithm calculates the multiplication of given two number in a much more efficient way. Normally multiplying two number takes $\theta($n^2) time complexity which is inefficient to use in daily life. However by dividing given numbers into two half, and multiplying till it's easy to multiply it approach takes around $\theta($n) time complexity. So first of all divide the numbers half and multiply small parts and add them together.

Q5. Start placing the minimum length colored-block to start index (initially is 0 and change in recursive calls). After placing a colored-block print the current combination. Then make a recursive call which the start index is ***startIndex + ColoredBlockLength + 1***. Don't forget to add 1 extra space for an empty block between each colored-blocks. This recursive call places another colored-block and this is continuing to end of the array. After printing all the combination which uses minimum length colored blocks, add one block to increase the length of the colored-block by one and print this combination. Call recursively this method again for each new combination. At that step length of the current colored-blocks are change but the recursive calls started placing colored-blocks from minimum length to length of the array. By doing this tedious task function have printed all the combination which placed a colored-block. Now that set empty first index and find all the combinations which is first index is empty and next index contains a colored-block. To do that call a new recursive call which start index is ***startIndex + 1***. So, it will do all the steps that mentioned above and by continuing like that eventually recursion stop when the end of the array is reached.

Q6. First of all in all combinations there must be at least one snake which start at first row (otherwise it is impossible to place them) that's because all the combinations consist of a snake which begins at first row. For N x N matrix there are N different starting position for snake. So if we can find a to make all the combinations that start with first cell of the matrix then it became the 1/N of all the combinations. Start with the first cell of the matrix and create a snake by trying each 4 cardinal direction. After create current snake pass next snake to place it also. After placing all the snakes display the current matrix which is one of the combinations of bending snakes.

4. TEST CASES

Q1.

N: number of total occurrences

I: desired occurrence

T1. No pattern exists in the big string.

T2. I < 1

T3. I > N

T4. I < N

T5. I = N

T6. Big string size is smaller than pattern size.

Q2.

T1. Given array doesn't contain one of the given integer values.

T2. Wrongly called method (smaller value shouldn't be larger than larger one)

T3. Given two integer value are same.

T4. Array contains no duplication.

T5. Array contains duplicates values for given two integers.

Q3.

T1. No sub array exist that sum of its element is equal to given integer value

T2. There exists more than one sub array which start at same index.

Q4.

T1. One of the given integers has single digit.
T2. Given integers has more than 5 digits.

Q5.

L: length of the blocks array

N: minimum size for colored-block

T1. L > N (L = 12, N = 3)
T2. L < N (L = 3, N = 6)
T3. L = N (L = 11, n = 11)

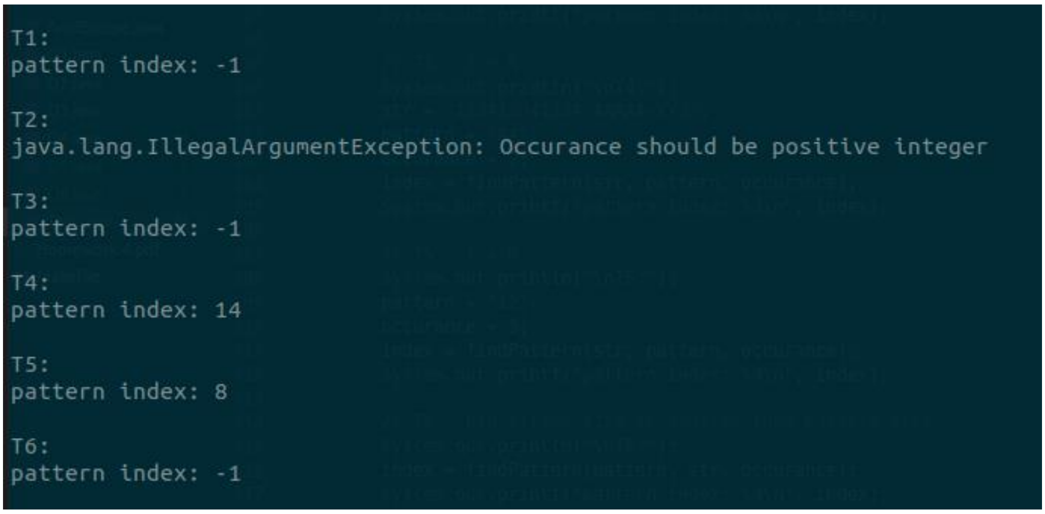
Q6.

N: Matrix size

T1. N = 3
T2. N = 4
T3. N = 5
T4. N = 6

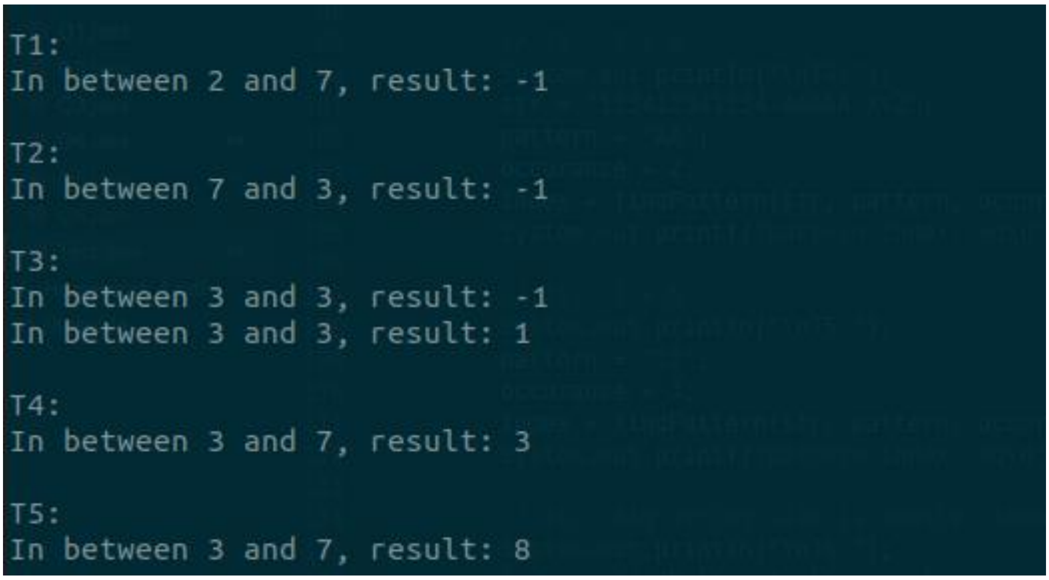
5. RUNNING AND RESULTS

Q1.



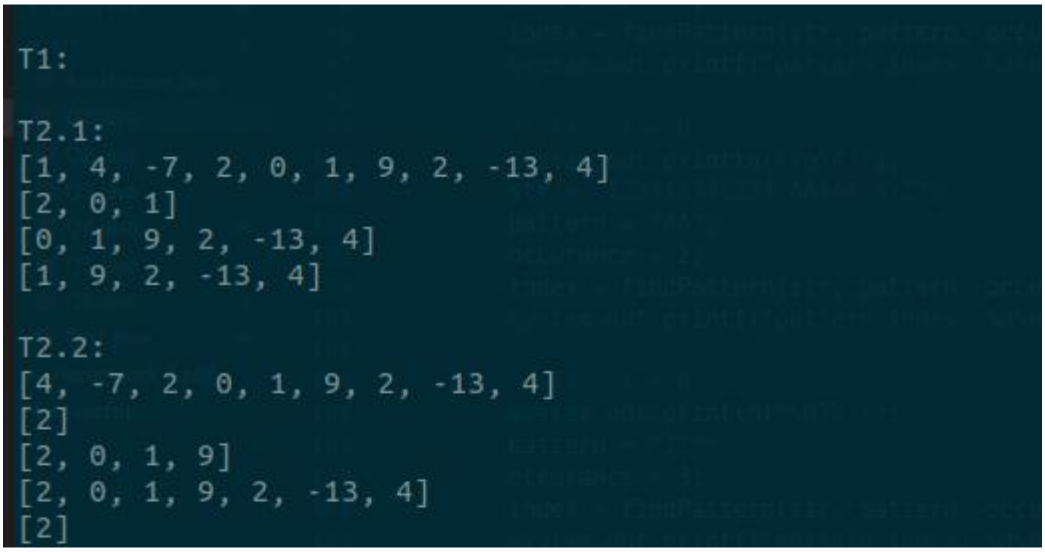
```
T1:
pattern index: -1

T2:
java.lang.IllegalArgumentException: Occurance should be positive integer

T3:
pattern index: -1

T4:
pattern index: 14

T5:
pattern index: 8

T6:
pattern index: -1
```

Q2.



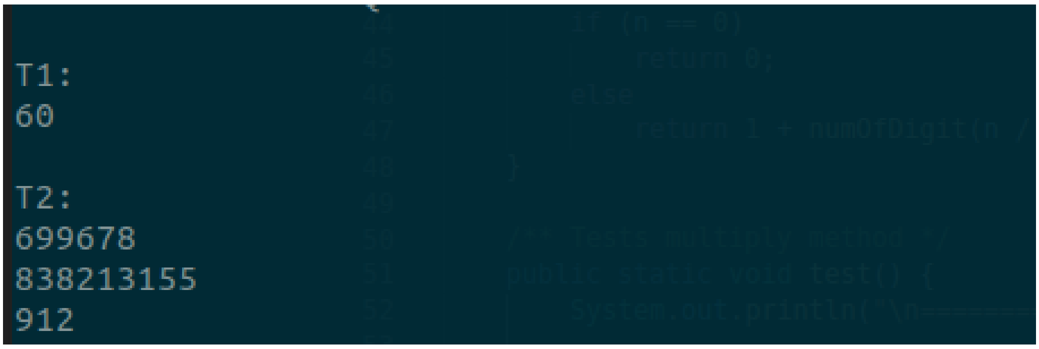
```
T1:
In between 2 and 7, result: -1

T2:
In between 7 and 3, result: -1

T3:
In between 3 and 3, result: -1
In between 3 and 3, result: 1

T4:
In between 3 and 7, result: 3

T5:
In between 3 and 7, result: 8
```

Q3.



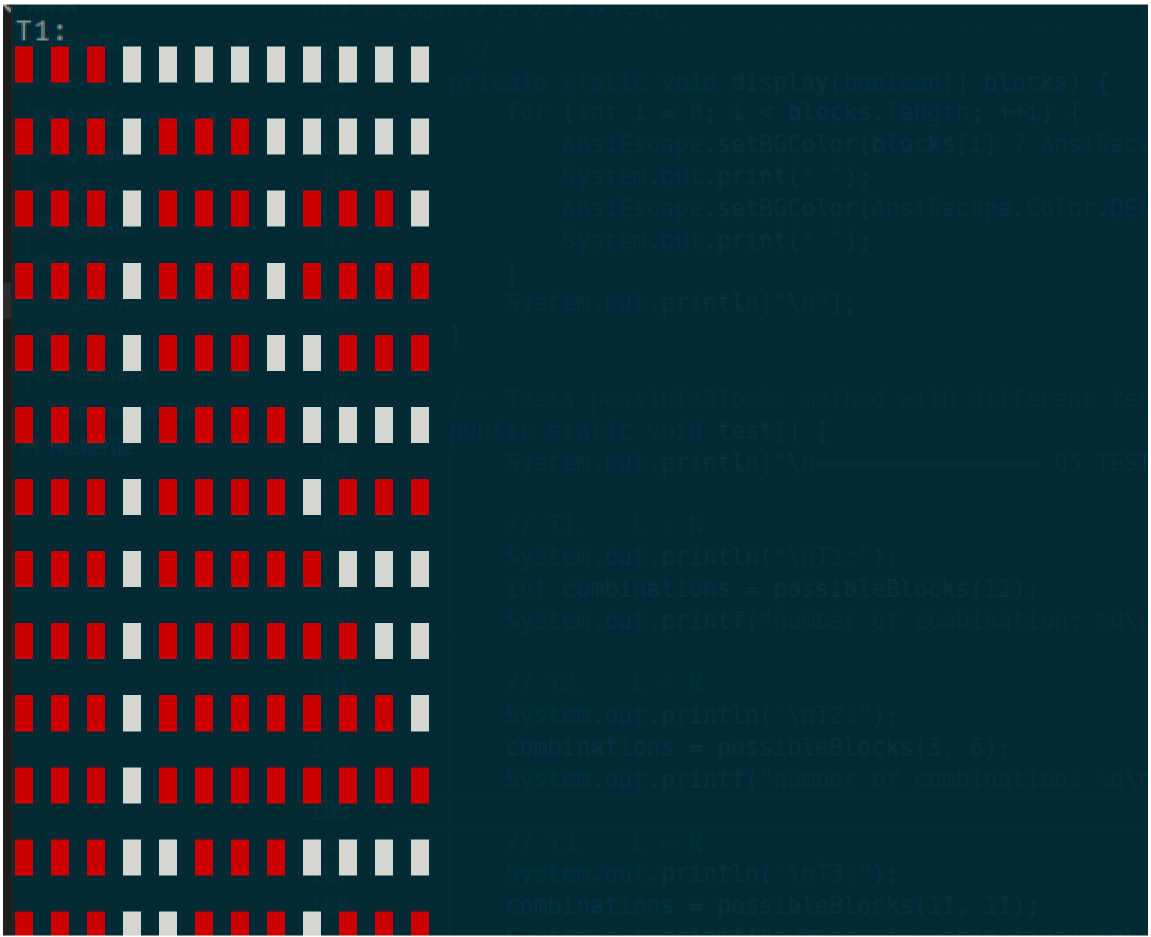
```
T1:

T2.1:
[1, 4, -7, 2, 0, 1, 9, 2, -13, 4]
[2, 0, 1]
[0, 1, 9, 2, -13, 4]
[1, 9, 2, -13, 4]

T2.2:
[4, -7, 2, 0, 1, 9, 2, -13, 4]
[2]
[2, 0, 1, 9]
[2, 0, 1, 9, 2, -13, 4]
[2]
```

Q4.

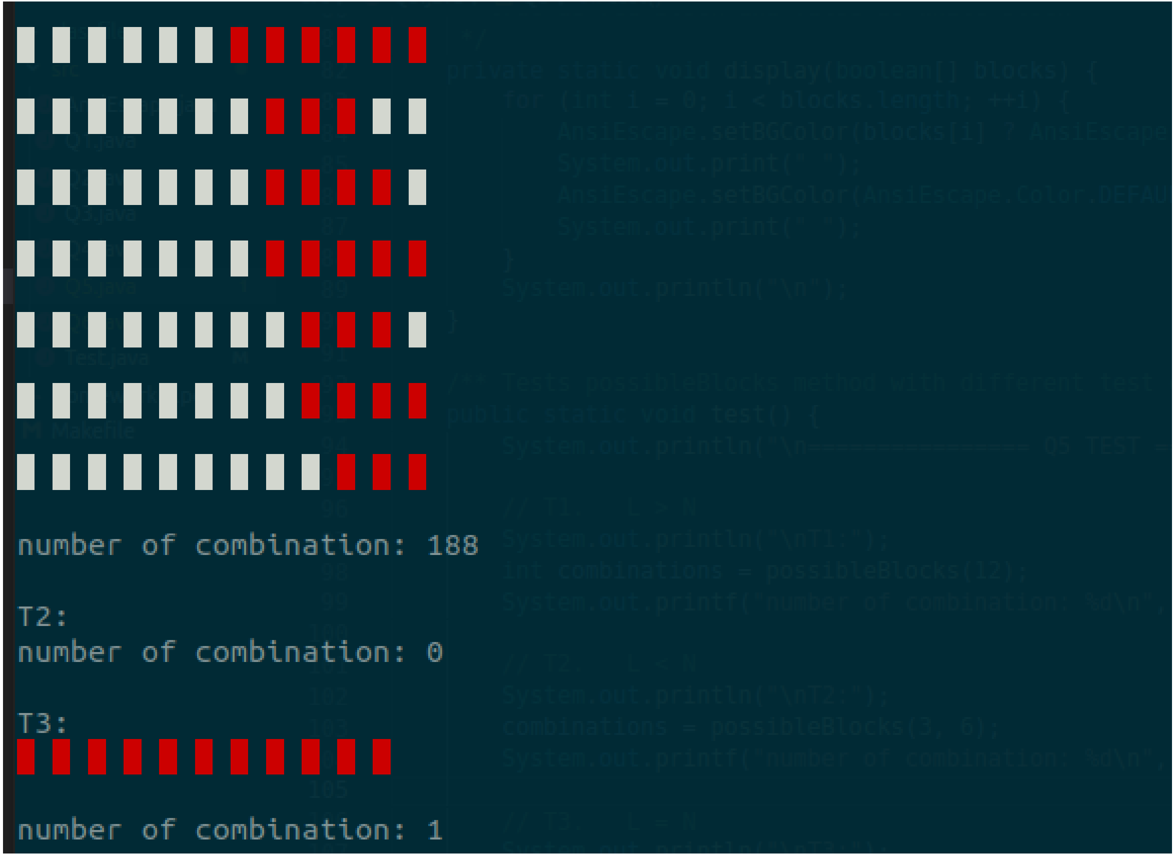


```
T1:
60

T2:
699678
838213155
912
```

Q5.

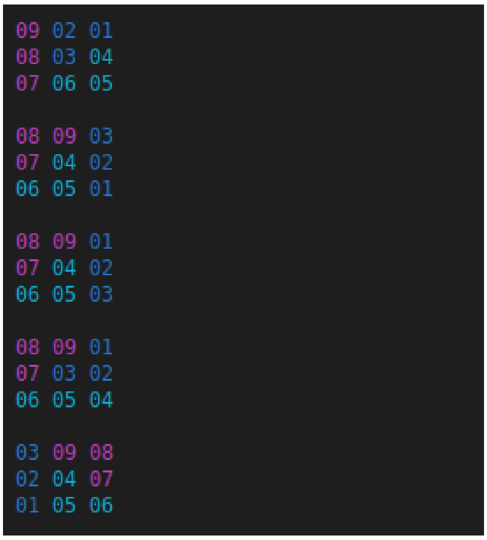


```
T1:
🟥🟥🟥⬜⬜⬜⬜⬜⬜⬜⬜⬜

🟥🟥🟥⬜🟥🟥🟥⬜⬜⬜⬜⬜

🟥🟥🟥⬜🟥🟥🟥⬜🟥🟥🟥⬜

🟥🟥🟥⬜🟥🟥🟥⬜🟥🟥🟥🟥

🟥🟥🟥⬜🟥🟥🟥⬜⬜🟥🟥🟥

🟥🟥🟥⬜🟥🟥🟥🟥⬜⬜⬜⬜

🟥🟥🟥⬜🟥🟥🟥🟥⬜🟥🟥🟥

🟥🟥🟥⬜🟥🟥🟥🟥🟥⬜⬜⬜

🟥🟥🟥⬜🟥🟥🟥🟥🟥🟥⬜⬜

🟥🟥🟥⬜🟥🟥🟥🟥🟥🟥🟥⬜

🟥🟥🟥⬜🟥🟥🟥🟥🟥🟥🟥🟥

🟥🟥🟥⬜⬜🟥🟥🟥⬜⬜⬜⬜

🟥🟥🟥⬜⬜🟥🟥🟥⬜🟥🟥🟥
```

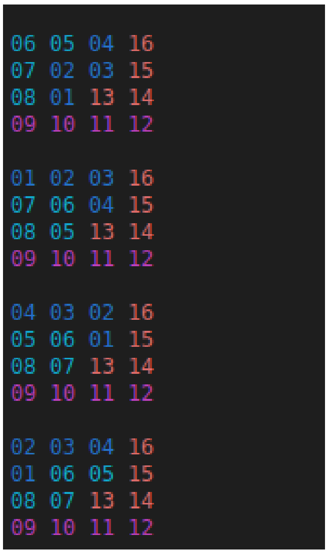


```
⬜⬜⬜⬜⬜⬜🟥🟥🟥🟥🟥🟥

⬜⬜⬜⬜⬜⬜⬜🟥🟥🟥⬜⬜

⬜⬜⬜⬜⬜⬜⬜🟥🟥🟥🟥⬜

⬜⬜⬜⬜⬜⬜⬜🟥🟥🟥🟥🟥

⬜⬜⬜⬜⬜⬜⬜⬜🟥🟥🟥⬜

⬜⬜⬜⬜⬜⬜⬜⬜🟥🟥🟥🟥

⬜⬜⬜⬜⬜⬜⬜⬜⬜🟥🟥🟥

number of combination: 188

T2:
number of combination: 0

T3:
🟥🟥🟥🟥🟥🟥🟥🟥🟥🟥🟥

number of combination: 1
```

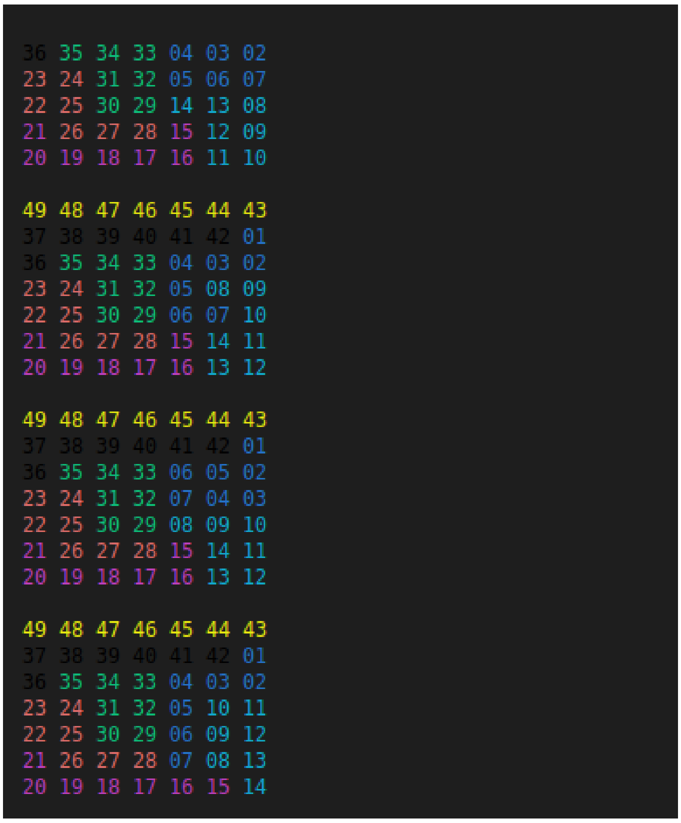
Q6.



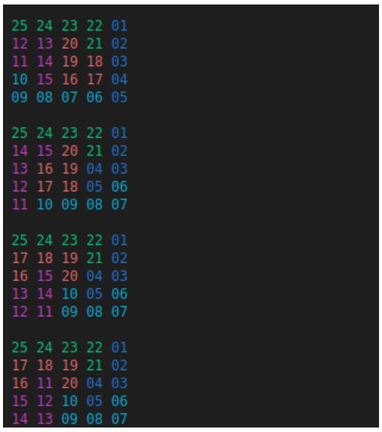
```
09 02 01
08 03 04
07 06 05

08 09 03
07 04 02
06 05 01

08 09 01
07 04 02
06 05 03

08 09 01
07 03 02
06 05 04

03 09 08
02 04 07
01 05 06
```



```
06 05 04 16
07 02 03 15
08 01 13 14
09 10 11 12

01 02 03 16
07 06 04 15
08 05 13 14
09 10 11 12

04 03 02 16
05 06 01 15
08 07 13 14
09 10 11 12

02 03 04 16
01 06 05 15
08 07 13 14
09 10 11 12
```



```
36 35 34 33 04 03 02
23 24 31 32 05 06 07
22 25 30 29 14 13 08
21 26 27 28 15 12 09
20 19 18 17 16 11 10

49 48 47 46 45 44 43
37 38 39 40 41 42 01
36 35 34 33 04 03 02
23 24 31 32 05 08 09
22 25 30 29 06 07 10
21 26 27 28 15 14 11
20 19 18 17 16 13 12

49 48 47 46 45 44 43
37 38 39 40 41 42 01
36 35 34 33 06 05 02
23 24 31 32 07 04 03
22 25 30 29 08 09 10
21 26 27 28 15 14 11
20 19 18 17 16 13 12

49 48 47 46 45 44 43
37 38 39 40 41 42 01
36 35 34 33 04 03 02
23 24 31 32 05 10 11
22 25 30 29 06 09 12
21 26 27 28 07 08 13
20 19 18 17 16 15 14
```



```
25 24 23 22 01
12 13 20 21 02
11 14 19 18 03
10 15 16 17 04
09 08 07 06 05

25 24 23 22 01
14 15 20 21 02
13 16 19 04 03
12 17 18 05 06
11 10 09 08 07

25 24 23 22 01
17 18 19 21 02
16 15 20 04 03
13 14 10 05 06
12 11 09 08 07

25 24 23 22 01
17 18 19 21 02
16 11 20 04 03
15 12 10 05 06
14 13 09 08 07
```