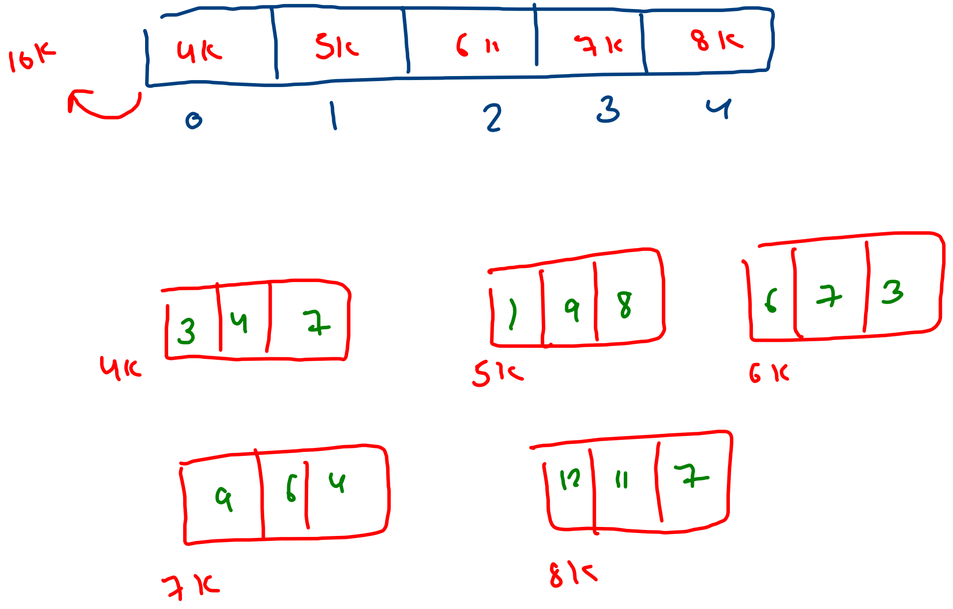
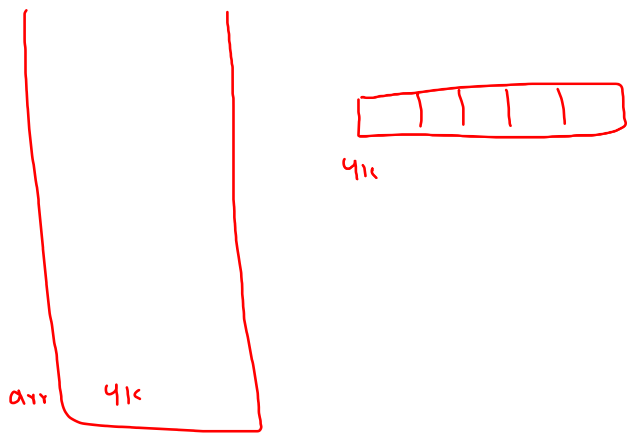
int [ ] [ ] mat = new int [5] [3] ;

int [ ] arr = new int [5];

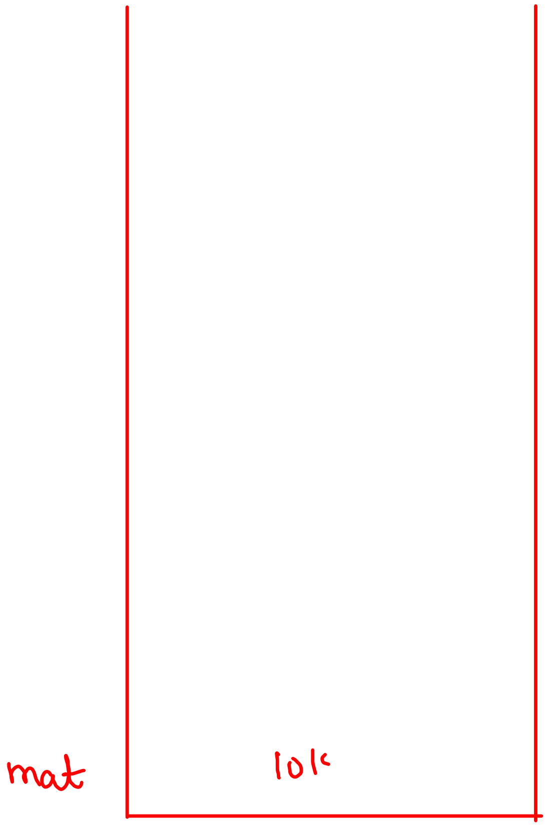


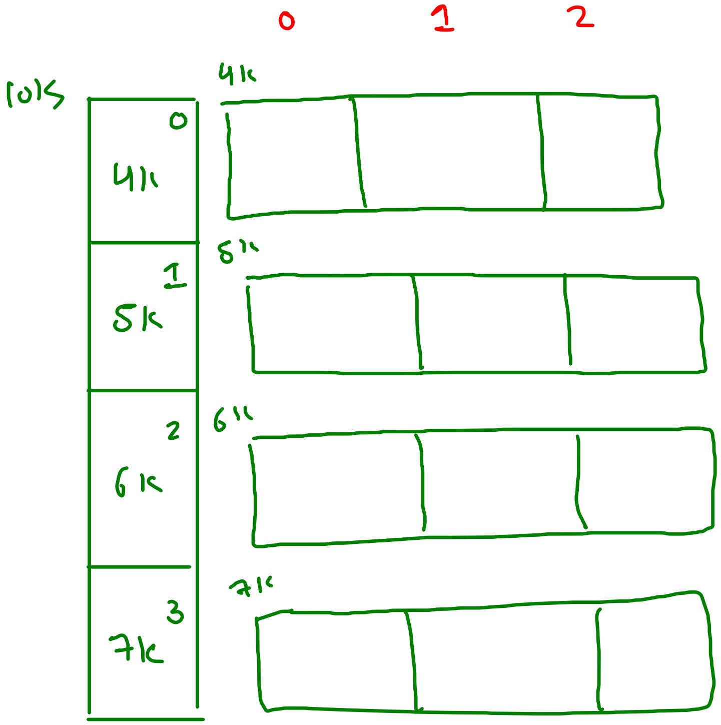


mat [2] [3] → out of bound

6k

int [ ] [ ] mat = new int[4][3];





mat [0] [2]

10k [0] [2]

4k [2]

mat [3] [1]

10k [3] [1]

7k [1]

rows = mat.length

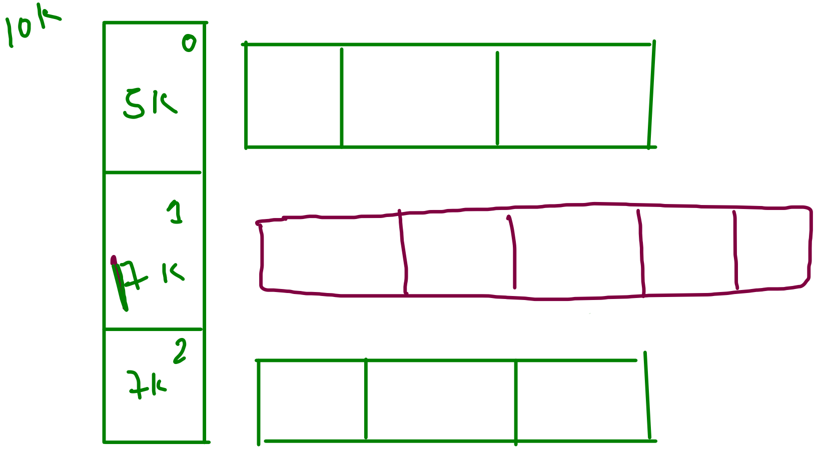
cols = mat[0].length

2d Array

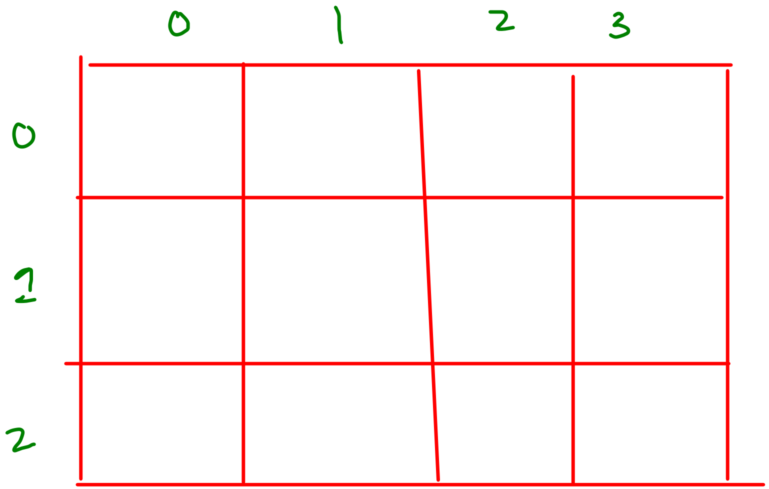
vs

matrix

```
int [ ] [ ] mat = new int [3] [4];

✓mat [1] = new int [5];
```



matrix



matrix mult

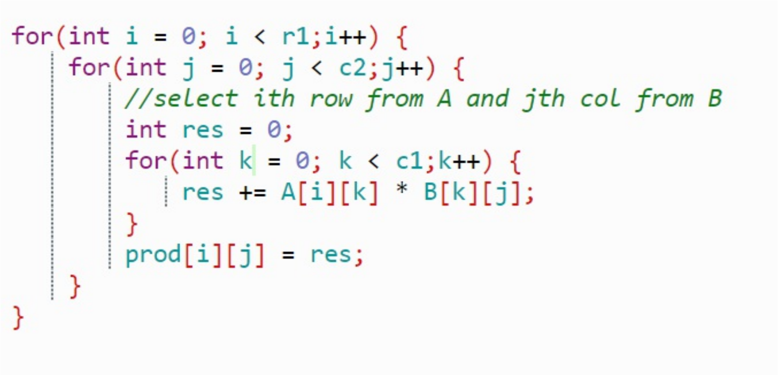
✓ matrix mult → $\boxed{c1 == r2}$

product mat → $r1 \times c2$

$$
\begin{array}{c|ccc}
 & 0 & 1 & 2 \\
\hline
0 & 1 & 2 & 3 \\
1 & 0 & 4 & 0 \\
2 & 2 & 0 & 6
\end{array}_{3 \times 3}
\times
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 2 & 0 & 1 \\
2 & 4 & 0 & -1 & 3
\end{array}_{3 \times 4}
= \rightarrow
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\
\hline
0 & & & & \\
1 & & & \bullet & \\
2 & & & &
\end{array}_{3 \times 4}
$$

A $\qquad$ r1 c1 $\qquad$ B $\qquad$ r2 c2

$$prod[1][2] = A[1][0] * B[0][2] + A[1][1] * B[1][2] + A[1][2] * B[2][2]$$



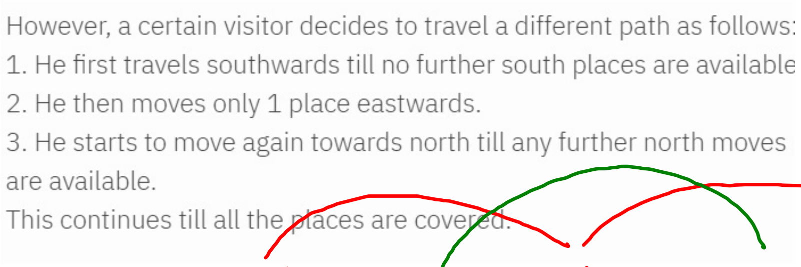
```
for(int i = 0; i < r1;i++) {
    for(int j = 0; j < c2;j++) {
        //select ith row from A and jth col from B
        int res = 0;
        for(int k = 0; k < c1;k++) {
            res += A[i][k] * B[k][j];
        }
        prod[i][j] = res;
    }
}
```

r1 = 2   r2 = 3

c1 = 3   c2 = 2

$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1, & 0, & 10 \\ 1 & 2, & 20, & 0 \end{array}_{2 \times 3} \times \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 0 \\ 1 & 2 & 0 \\ 2 & 0 & 3 \end{array}_{3 \times 2} = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 30 \\ 1 & 42 & 0 \end{array}$$

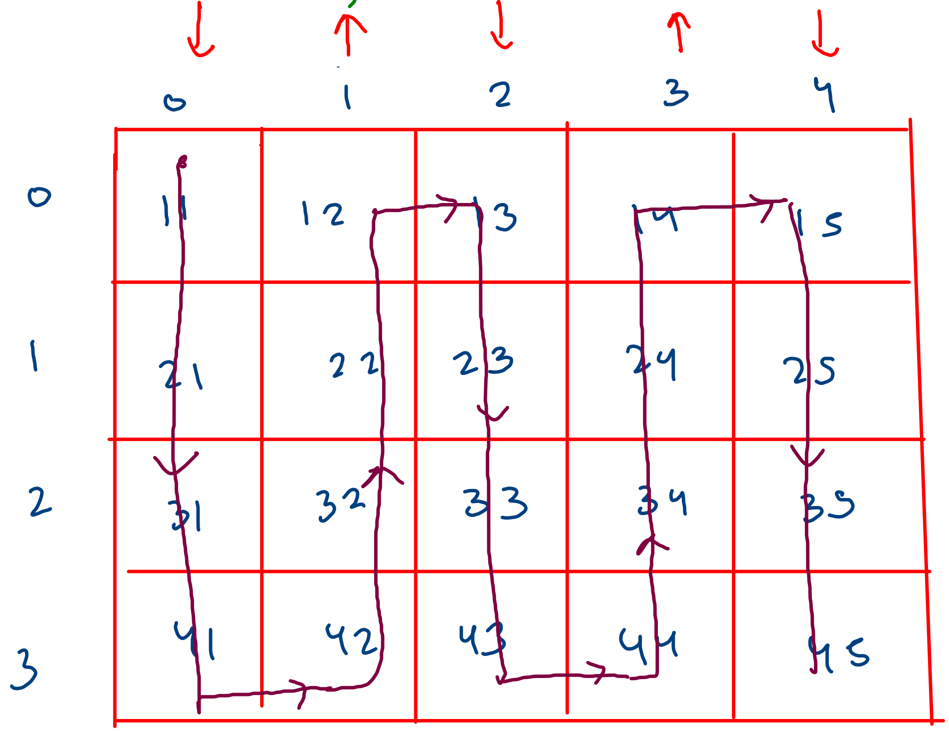
A   B   prod



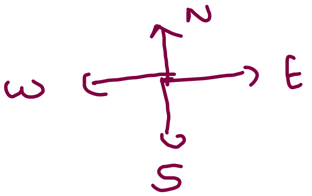
However, a certain visitor decides to travel a different path as follows:

1. He first travels southwards till no further south places are available
2. He then moves only 1 place eastwards.
3. He starts to move again towards north till any further north moves are available.

This continues till all the places are covered.



	0	1	2	3	4
0	11	12	13	14	15
1	21	22	23	24	25
2	31	32	33	34	35
3	41	42	43	44	45



11 21 31 41 42 32

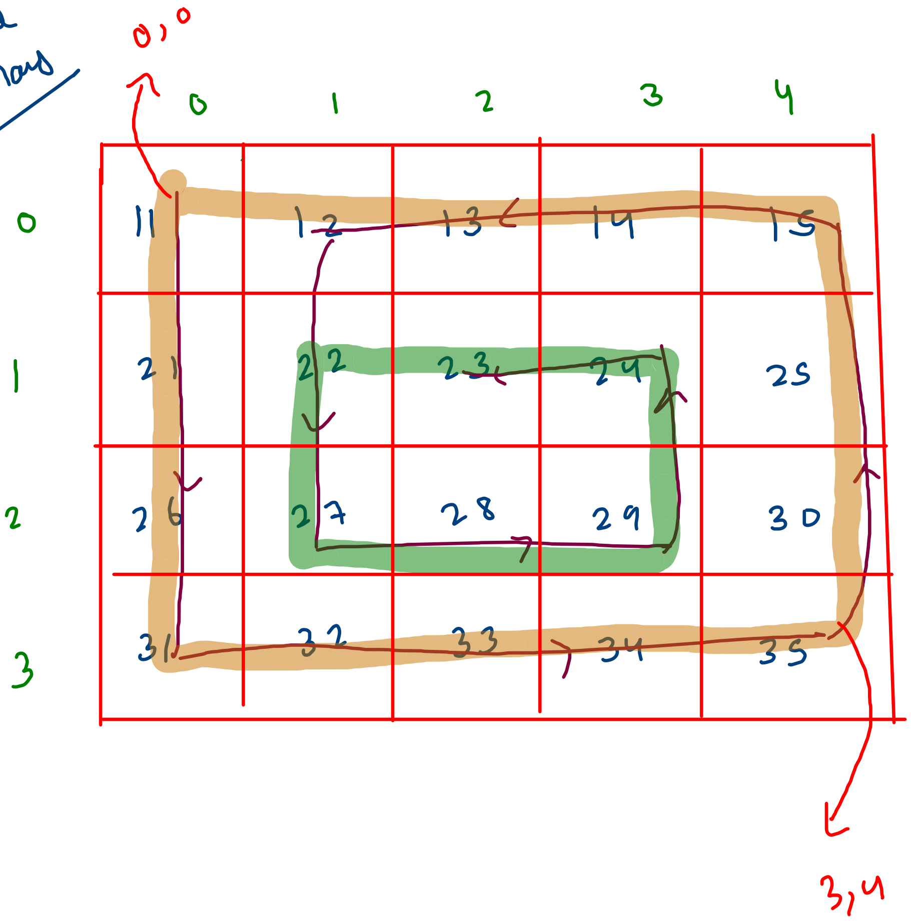
22 12 13 23 33 43

44 34 24 14 15 25

35 45.

Column traversal

Spiral display



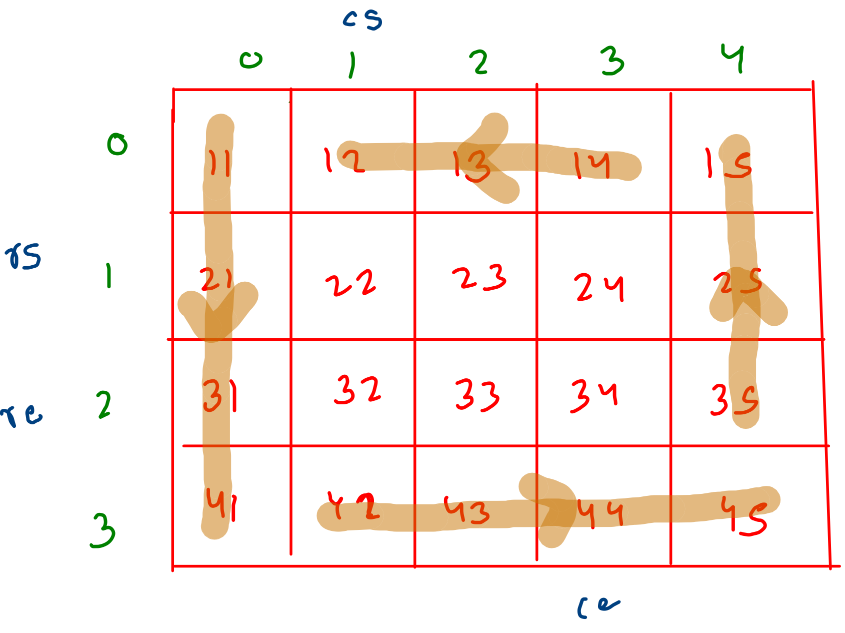
	0	1	2	3	4
0	11	12	13	14	15
1	21	22	23	24	25
2	26	27	28	29	30
3	31	32	33	34	35

11 21 26 31 32 33

34 35 30 25 15 14

13 12 22 27 28 29

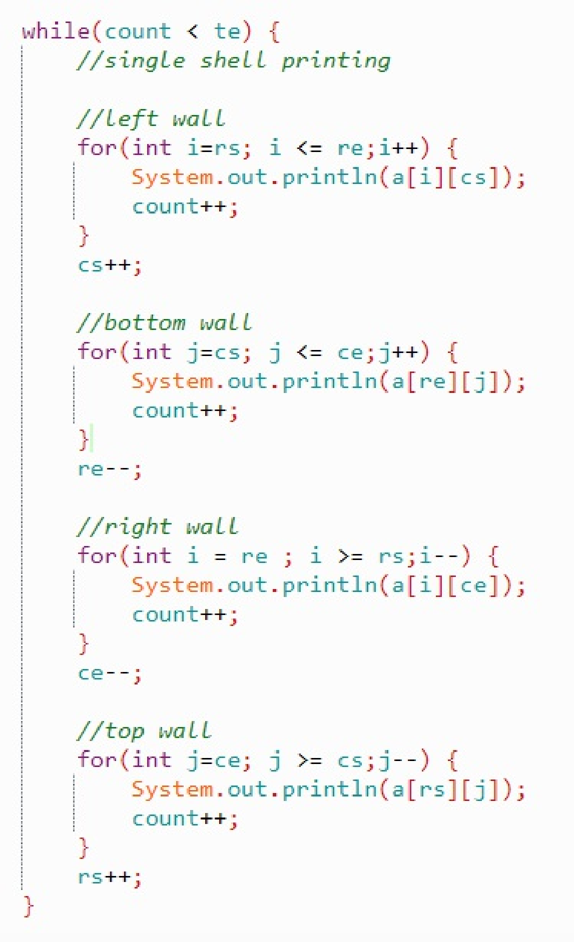
24 23



	0	1	2	3	4
0	11	12	13	14	15
1	21	22	23	24	25
2	31	32	33	34	35
3	41	42	43	44	45

```
// left wall
for(int i = rs ; i <= re ; i++) {
        syso ( a[i] [cs]);

}
cs++;
// bottom wall
for( int j = cs; j <= ce ; j++) {
        syso (a [re] [j]);

}
re--;
// right wall
for(int i = re ;  i >= rs ; i--) {
        syso ( a[i] [ce])
}
ce--
```

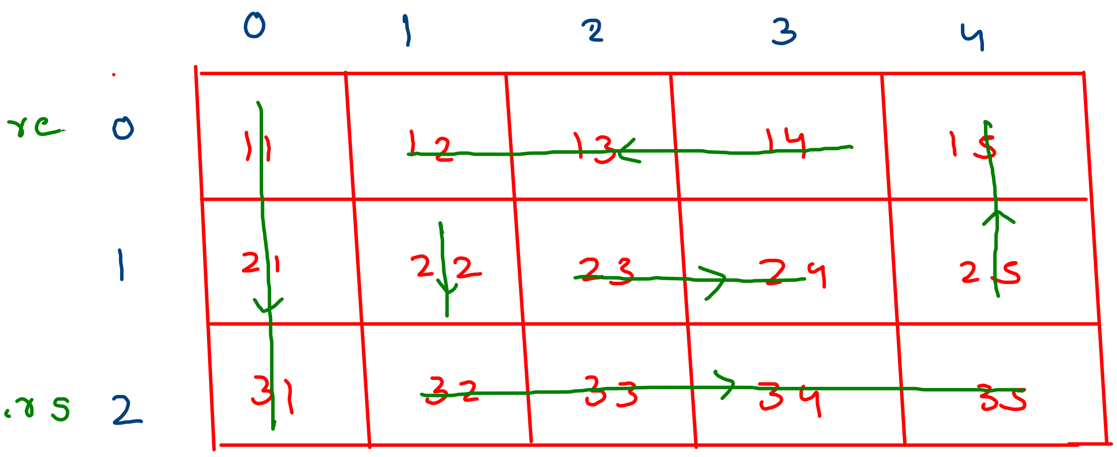
```
// top wall

for(int j = ce ; j >= cs ; j--) {
        syso (a [rs] [j]);

}
rs++;
```



```
while(count < te) {
    //single shell printing

    //left wall
    for(int i=rs; i <= re;i++) {
        System.out.println(a[i][cs]);
        count++;
    }
    cs++;

    //bottom wall
    for(int j=cs; j <= ce;j++) {
        System.out.println(a[re][j]);
        count++;
    }
    re--;

    //right wall
    for(int i = re ; i >= rs;i--) {
        System.out.println(a[i][ce]);
        count++;
    }
    ce--;

    //top wall
    for(int j=ce; j >= cs;j--) {
        System.out.println(a[rs][j]);
        count++;
    }
    rs++;
}
```

ce

cs



		0	1	2	3	4
re	0	11	12	13	14	15
	1	21	22	23	24	25
rs	2	31	32	33	34	35

te = 15

count = ~~0~~ ~~3~~ ~~7~~ ~~9~~ ~~12~~ ~~13~~ ~~15~~ 16

11 21 31 32 33 34 35 25 15

14 13 12 22 23 24 23