

Forum:[Hack Forums](#)

Thread: [Sequence print](#)

ShallowME - February 10, 2020, 5:15 am

I have an array,
`int dwArr[3] = {2,2,2};`

I want the output as:

222
221
220

212
211
210

202
201
200

...

How would I go about this route?

dracx - February 10, 2020, 6:34 am

Code D:\Scripts_Sources>python t.py

2 2 2
2 2 1
2 2 0

2 1 2
2 1 1
2 1 0

2 0 2
2 0 1
2 0 0

=====

1 2 2
1 2 1
1 2 0

1 1 2

```
1 1 1
1 1 0
```

```
1 0 2
1 0 1
1 0 0
```

```
=====
```

```
0 2 2
0 2 1
0 2 0
```

```
0 1 2
0 1 1
0 1 0
```

```
0 0 2
0 0 1
0 0 0
```

```
=====
```

D:\Scripts_Sources>

I dunno if I understood you right, but if the output you want is something like the above one, I have made some python script for it and it shouldn't be hard to convert it to C.

```
Code arr = [2,2,2]
arrlen = len(arr) - 1
```

```
for i in range(0, arr[0]+1):
    first = arr[0]-i
    for j in range(0, arr[1]+1):
        second = arr[1]-j
        for k in range(0, arr[2]+1):
            third = arr[2]-k

            print(first, second, third)
        print("\n")
    print("=====\n") Code #include <stdio.h>
```

```
int main(){
    int arr[3] = {2, 2, 2};

    for (int i = 0; i <= arr[0]; i++){
        int first = arr[i]-i;
        for (int j = 0; j <= arr[1]; j++){
```

```

    int second = arr[1] - j;
    for (int k = 0; k <= arr[2]; k++){
        int third = arr[2] - k;
        // Could proly use sprintf to concat these values if you want em inside a variable
        printf("%d%d%d\n", first, second, third);
    }
}
printf("\n");
}

return 0;
}
Got bored, lol.

```

You possibly could make the code more dynamic by figuring out the lenght of the array and doing a loop accordingly.

Shisho - February 10, 2020, 6:11 pm

(Feb 10, 2020 07:34 AM)dracx Wrote: [\[CITE\]](#) Code D:\Scripts_Sources>python t.py

```

2 2 2
2 2 1
2 2 0

```

```

2 1 2
2 1 1
2 1 0

```

```

2 0 2
2 0 1
2 0 0

```

=====

```

1 2 2
1 2 1
1 2 0

```

```

1 1 2
1 1 1
1 1 0

```

```

1 0 2
1 0 1
1 0 0

```

=====

```
0 2 2
0 2 1
0 2 0
```

```
0 1 2
0 1 1
0 1 0
```

```
0 0 2
0 0 1
0 0 0
```

```
=====
```

D:\Scripts_Sources>

I dunno if I understood you right, but if the output you want is something like the above one, I have made some python script for it and it shouldn't be hard to convert it to C.

```
Code arr = [2,2,2]
arrlen = len(arr) - 1
```

```
for i in range(0, arr[0]+1):
    first = arr[0]-i
    for j in range(0, arr[1]+1):
        second = arr[1]-j
        for k in range(0, arr[2]+1):
            third = arr[2]-k
```

```
print(first, second, third)
print("\n")
print("=====\\n") Code #include <stdio.h>
```

```
int main(){
    int arr[3] = {2, 2, 2};

    for (int i = 0; i <= arr[0]; i++){
        int first = arr[i]-i;
        for (int j = 0; j <= arr[1]; j++){
            int second = arr[1] - j;
            for (int k = 0; k <= arr[2]; k++){
                int third = arr[2] - k;
                // Could proly use sprintf to concat these values if you want em inside a variable
                printf("%d%d%d\\n", first, second, third);
            }
        }
        printf("\\n");
    }

    return 0;
```

```
}
```

Got bored, lol.

You possibly could make the code more dynamic by figuring out the length of the array and doing a loop accordingly.

Nice try but he wants to print the descending sequence of numbers in groups of 3 where the next group has skipped 8 numbers. If you were really bored you would have done things more cleanly and provided a solution that the user could perhaps submit as a homework assignment :)

I've written a solution, which I have commented, that is dynamic to the point where the array (as long as it's not multi-dimensional) can be as long as you want and the numbers inside of it can be as big as you want as long as joined together produce a value that is not larger than an integer.

It properly manages memory by only allocating what is needed to produce the string and reallocates as needed to grow it. It then prints the values in descending order with the requested format!

```
Code #include <stdlib.h>
#include <stdio.h>
```

```
int array[3] = {2,2,2};
```

```
int main() {
```

```
    int arraySize = sizeof(array) / sizeof(array[0]); //get the length of the array by dividing the total
    size by the size of one of the elements
```

```
    char *integerString = NULL; //create a pointer to a string that we will build that will hold each
    element consecutively to build a new number
```

```
    int integerStringSize = 0; //create an int to hold the size of the string as we grow it
```

```
    int offset = 0; //create an int that will hold an offset so that we can add each element to the end
    of the old element
```

```
    for(int i = 0; i < arraySize; i++) { //iterate over each element of the array
```

```
        int integerSize = snprintf(NULL, 0, "%d", array[i]); //get the length of the integer within the
    element
```

```
        integerStringSize += integerSize; //increment the int holding the current size of the string
    we are building
```

```
        char *newPtr = NULL; //allocate a new pointer for the following realloc()
```

```
        newPtr = realloc(integerString, integerStringSize + 1); //reallocate the string we are
    building to hold the element
```

```
        integerString = newPtr; //assign the old pointer to the new pointer
```

```
        snprintf(integerString + offset, integerStringSize + 1, "%d", array[i]); //copy the value of
    the element to the string
```

```
        offset += integerSize; //increase the offset so we can add the following element to the end
    }
```

```
    int newInteger = atoi(integerString); //convert string to new integer
```

```
    int counter = 0; //create a counter so we can print only the numbers we want by incrementing it
```

each iteration of the following loop and checking the current value

```
for(int i = newInteger; i >= 200; i--) { //loop from the new integer backwards until you reach whatever number you want
```

```
    if(counter < 3) { //print a number if counter isn't 3
        printf("%d\n", i);
    }
```

```
    counter++;
```

```
    if(counter == 10) { //reset the counter and print a new line if we have skipped 8 numbers
        counter = 0;
        printf("\n");
    }
```

```
}
```

Have fun OP and hope you learned a couple of things!

dracx - February 10, 2020, 8:07 pm

(Feb 10, 2020 07:11 PM)Shisho Wrote: [\[CITE\]](#) Nice try but he wants to print the descending sequence of numbers in groups of 3 where the next group has skipped 8 numbers. If you were really bored you would have done things more cleanly and provided a solution that the user could perhaps submit as a homework assignment :)

I've written a solution, which I have commented, that is dynamic to the point where the array (as long as it's not multi-dimensional) can be as long as you want and the numbers inside of it can be as big as you want as long as joined together produce a value that is not larger than an integer.

It properly manages memory by only allocating what is needed to produce the string and reallocates as needed to grow it. It then prints the values in descending order with the requested format!

```
Code #include <stdlib.h>
```

```
#include <stdio.h>
```

```
int array[3] = {2,2,2};
```

```
int main() {
```

```
    int arraySize = sizeof(array) / sizeof(array[0]); //get the length of the array by dividing the total size by the size of one of the elements
```

```
    char *integerString = NULL; //create a pointer to a string that we will build that will hold each element consecutively to build a new number
```

```
    int integerStringSize = 0; //create an int to hold the size of the string as we grow it
```

```
    int offset = 0; //create an int that will hold an offset so that we can add each element to the end of the old element
```

```
    for(int i = 0; i < arraySize; i++) { //iterate over each element of the array
```

```
        int integerSize = snprintf(NULL, 0, "%d", array[i]); //get the length of the integer within the element
```

```
        integerStringSize += integerSize; //increment the int holding the current size of the string
```

we are building

```
char *newPtr = NULL; //allocate a new pointer for the following realloc()
newPtr = realloc(integerString, integerStringSize + 1); //reallocate the string we are
building to hold the element
integerString = newPtr; //assign the old pointer to the new pointer

snprintf(integerString + offset, integerStringSize + 1, "%d", array[i]); //copy the value of
the element to the string

offset += integerSize; //increase the offset so we can add the following element to the end
}

int newInteger = atoi(integerString); //convert string to new integer

int counter = 0; //create a counter so we can print only the numbers we want by incrementing it
each iteration of the following loop and checking the current value

for(int i = newInteger; i >= 200; i--) { //loop from the new integer backwards until you reach
whatever number you want

    if(counter < 3) { //print a number if counter isn't 3
        printf("%d\n", i);
    }

    counter++;

    if(counter == 10) { //reset the counter and print a new line if we have skipped 8 numbers
        counter = 0;
        printf("\n");
    }
}
}
```

Have fun OP and hope you learned a couple of things!! dont get it, lol, haha.

first your code produces this:

Code 222

221

220

212

211

210

202

201

200

Mine produces this:

Code 222

221

220

212

211

210
202
201
200

122
121
120
112
111
110
102
101
100

022
021
020
012
011
010
002
001
000

The OP's sample output was

Code 222

221
220

212
211
210

202
201
200

... // <---- This probably says that there should be more output right?

My code may really not separated the output properly (even tho it was on the python script), and it could be fixed by simply re-positioning the printf function (and as I assumed, nothing the OP could not handle).

But about the "print the descending sequence of numbers in groups of 3 where the next group has skipped 8 numbers.", I thought he was just tryin to recursively descend each number to 0, and not -8 every after 3(you know what am saying, lol).

And I was actually focusing on the algorithm more than the usability/dynamicity of the the code as I assumed, OP was only struggling on the algorithm to produce the desired output, so I just tried to show some.

Shisho - February 10, 2020, 8:30 pm

(Feb 10, 2020 09:07 PM)dracx Wrote: [\[CITE\]](#) I dont get it, lol, haha.

first your code produces this:

Code 222

221

220

212

211

210

202

201

200

Mine produces this:

Code 222

221

220

212

211

210

202

201

200

122

121

120

112

111

110

102

101

100

022

021

020

012

011

010

002

001

000

The OP's sample output was

Code 222

221

220

212

211

210

202

201

200

... // <---- This probably says that there should be more output right?

My code may really not separated the output properly (even tho it was on the python script), and it could be fixed by simply re-positioning the printf function (and as I assumed, nothing the OP could not handle).

But about the "print the descending sequence of numbers in groups of 3 where the next group has skipped 8 numbers.", I thought he was just tryin to recursively descend each number to 0, and not -8 every after 3(you know what am saying, lol).

And I was actually focusing on the algorithm more than the usability/dynamicity of the the code as I assumed, OP was only struggling on the algorithm to produce the desired output, so I just tried to show some.

I stopped it at 200 but it can be easily changed to any number, even a negative value in the reverse for loop. We have no idea where he wants it to stop at do we? The reason I did this was to not flood my terminal but still show the functionality. His example prints 3 numbers, then it skips 8 and prints 3 more. The title is "Sequence print" not recursively subtract 1 :)

dracx - February 10, 2020, 8:44 pm

(Feb 10, 2020 09:30 PM)Shisho Wrote: [\[CITE\]](#) I stopped it at 200 but it can be easily changed to any number, even a negative value in the reverse for loop. We have no idea where he wants it to stop at do we? The reason I did this was to not flood my terminal but still show the functionality. His example prints 3 numbers, then it skips 8 and prints 3 more. The title is "Sequence print" not recursively subtract 1 :)

HAHAHAHA, we looked and understood his sample in different ways lol. I maybe would have thought of something similar like yours if he would have given 222 instead of {2,2,2}.

Shisho - February 10, 2020, 8:46 pm

(Feb 10, 2020 09:44 PM)dracx Wrote: [\[CITE\]](#) HAHAHAHA, we looked and understood his sample in different ways lol. I maybe would have thought of something similar like yours if he would have given 222 instead of {2,2,2}.

You are right, we did understand in different ways :) if he had just given us the number 222 it would have been way too easy. The array makes it more challenging! When he logs back in he can clear up what he wants :P

0xDEAD10CC - February 12, 2020, 3:41 pm

This doesn't explain what should happen if the numbers in the array weren't the same (i.e. { 3, 2, 2 })), and the solutions posted aren't dynamic enough to account for an array that would contain 100 elements.

And if the numbers in the array are always the same, why would you need an array in the first place as input? Just pass params to a function which specify the size and the number to start at.

Btw, this is more of a recursive problem

dracx - February 15, 2020, 4:36 am

(Feb 12, 2020 04:41 PM)0xDEAD10CC Wrote: [\[CITE\]](#) This doesn't explain what should happen if the numbers in the array weren't the same (i.e. { 3, 2, 2 })), and the solutions posted aren't dynamic enough to account for an array that would contain 100 elements.

And if the numbers in the array are always the same, why would you need an array in the first place as input? Just pass params to a function which specify the size and the number to start at.

Btw, this is more of a recursive problem

Only if @OP gave more info tho.

Shisho - February 16, 2020, 9:41 pm

(Feb 12, 2020 04:41 PM)0xDEAD10CC Wrote: [\[CITE\]](#) This doesn't explain what should happen if the numbers in the array weren't the same (i.e. { 3, 2, 2 }), and the solutions posted aren't dynamic enough to account for an array that would contain 100 elements.

And if the numbers in the array are always the same, why would you need an array in the first place as input? Just pass params to a function which specify the size and the number to start at.

Btw, this is more of a recursive problem

My solution is dynamic enough where it does not care how many elements there are in the array or the width of the elements.

0xDEAD10CC - February 17, 2020, 12:23 pm

(Feb 16, 2020 10:41 PM)Shisho Wrote: [\[CITE\]](#) My solution is dynamic enough where it does not care how many elements there are in the array or the width of the elements.

Except it doesn't work:

Shisho - February 17, 2020, 12:29 pm

(Feb 17, 2020 01:23 PM)0xDEAD10CC Wrote: [\[CITE\]](#) Except it doesn't work:

I read the scope of the program as building an integer from concatenation of each element of the array and then printing a decrementing sequence in the specified pattern. The description of what the user wanted is vague so I wrote the case I understood they wanted. From the example you've shown it built the integer "2222" and did exactly what the user showed in his example.

0xDEAD10CC - February 17, 2020, 1:33 pm

(Feb 17, 2020 01:29 PM)Shisho Wrote: [\[CITE\]](#) I read the scope of the program as building an integer from concatenation of each element of the array and then printing a decrementing sequence in the specified pattern. The description of what the user wanted is vague so I wrote the case I understood they wanted. From the example you've shown it built the integer "2222" and did exactly what the user showed in his example.

Only for the first couple groups. At no point in the OP's example output did he show a numerical column increasing. Also, I didn't show what happens at the end values where the rows become 3 digits long instead of 4.

From what I can tell, this is a recursive problem where if the leftmost column digit reaches 0 in the current row, the cycle will repeat with the next leftmost column with the rightmost columns resetting back to their initial values... All you need to do is watch the rightmost column in his example to reason with this; at no point does it reset to 9 (so why would the tens column reset to 9 with your code when dealing with 4 digits?)

Shisho - February 17, 2020, 1:54 pm

(Feb 17, 2020 02:33 PM)0xDEAD10CC Wrote: [\[CITE\]](#) Only for the first couple groups. At no point in the OP's example output did he show a numerical column increasing. Also, I didn't show what happens at the end values where the rows become 3 digits long instead of 4.

From what I can tell, this is a recursive problem where if the leftmost column digit reaches 0 in the current row, the cycle will repeat with the next leftmost column with the rightmost columns resetting back to their initial values... All you need to do is watch the rightmost column in his example to reason with this; at no point does it reset to 9 (so why would the tens column reset to 9 with your code when dealing with 4 digits?)

Can we assume it's purely coincidence that it looks like he would want a recursive subtraction of 1 from each element? Even then it's not because, like you said, the last element resets itself under

what condition? There is no pattern there to suggest recursion.

It's simple arithmetic. It looks like he's counting down from 222 in groups of 3 while then skipping 8 numbers. That is a sequence, not what you suggest.

0xDEAD10CC - February 17, 2020, 3:31 pm

(Feb 17, 2020 02:54 PM)Shisho Wrote: [\[CITE\]](#) Can we assume it's purely coincidence that it looks like he would want a recursive subtraction of 1 from each element? Even then it's not because, like you said, the last element resets itself under what condition? There is no pattern there to suggest recursion.

It's simple arithmetic. It looks like he's counting down from 222 in groups of 3 while then skipping 8 numbers. That is a sequence, not what you suggest.

I'm saying that this is a typical recursion problem similar to lexical permutations - the pattern is quite clear. The last elements reset once the current leftmost column hits 0 as I said; that's the condition.

If it's a sequence, then why does the rightmost column not go "2, 1, 0, 9, ..."? Why would the tens column be any different?

Shisho - February 17, 2020, 5:22 pm

(Feb 17, 2020 04:31 PM)0xDEAD10CC Wrote: [\[CITE\]](#) I'm saying that this is a typical recursion problem similar to lexical permutations - the pattern is quite clear. The last elements reset once the current leftmost column hits 0 as I said; that's the condition.

If it's a sequence, then why does the rightmost column not go "2, 1, 0, 9, ..."? Why would the tens column be any different?

Because it is subtracting 8 every 3 numbers printed. Why wouldn't the tens and hundreds and thousands places change as a result of that subtraction?

I guess we can agree to two different possible solutions to this problem due to the vagueness of the request which for you is an abstract recursion and for me it is a simple subtraction problem.

0xDEAD10CC - February 17, 2020, 5:26 pm

(Feb 17, 2020 06:22 PM)Shisho Wrote: [\[CITE\]](#) Because it is subtracting 8 every 3 numbers printed. Why wouldn't the tens and hundreds and thousands places change as a result of that subtraction?

I guess we can agree to two different possible solutions to this problem due to the vagueness of the request which for you is an abstract recursion and for me it is a simple subtraction problem.

How do you deduce that 8 is going to be the magic number for every number sequence? What if the values were { 3, 2, 3 }? You said your solution was dynamic, but to me, you've only applied an algorithm to fit this very specific sequence with just an array of { 2, 2, 2 }.

I know we might not get another answer from OP, but I don't see how the sequence I generated using { 2, 2, 2, 2 } with your code exhibits the same pattern as the example output in the first post.

He also put "..." at the end of his post which typically means the sequence should continue. Using the pattern that I implied, I would imagine it would continue down to 000. And going from 2, 1, 0 in the hundreds column is not a subtraction by 8. If anything it would be powers of 10 across the numeric value represented by the array.

Shisho - February 17, 2020, 6:02 pm

(Feb 17, 2020 06:26 PM)0xDEAD10CC Wrote: [\[CITE\]](#) How do you deduce that 8 is going to be the magic number for every number sequence? What if the values were { 3, 2, 3 }? You said your

solution was dynamic, but to me, you've only applied an algorithm to fit this very specific sequence with just an array of { 2, 2, 2 }.

I know we might not get another answer from OP, but I don't see how the sequence I generated using { 2, 2, 2, 2 } with your code exhibits the same pattern as the example output in the first post.

He also put "..." at the end of his post which typically means the sequence should continue. Using the pattern that I implied, I would imagine it would continue down to 000. And going from 2, 1, 0 in the hundreds column is not a subtraction by 8. If anything it would be powers of 10.

See the below explanation.

Code 222 <— print 3 numbers

221

220 <— subtract 8

212 <— print 3 numbers

211

210 <— subtract 8

202 <— print 3 numbers

201

200 <— subtract 8

With my code if the array was "3, 2, 3" it would become:

Code 323

322

321

313

312

311

303

302

301

The code reads every integer out of every element and uses a format specific sprintf to build the starting integer. It does not care if it's "2, 2, 2" or "97, 56, 21". It builds a larger integer by joining left to the right and doing that recursively until it finishes and then begins printing in the pattern. If you cannot see the 8 in the original example you are single minded.

0xDEAD10CC - February 17, 2020, 6:05 pm

(Feb 17, 2020 07:02 PM)Shisho Wrote: [\[CITE\]](#) See the below explanation.

Code 222 <— print 3 numbers

221

220 <— subtract 8

212 <— print 3 numbers

211

210 <— subtract 8

202 <— print 3 numbers

201

200 <— subtract 8

With my code if the array was “3, 2, 3” it would become:

Code 323

322

321

313

312

311

303

302

301

The code reads every integer out of every element and uses a format specific `snprintf` to build the starting integer. It does not care if it's “2, 2, 2” or “97, 56, 21”. It builds a larger integer by joining left to the right and doing that recursively until it finishes and then begins printing in the pattern. If you cannot see the 8 in the original example you are single minded.

If you can't see that OP had a “...” at the end of the example sequence you're "single minded". This means the sequence would continue. You looked at just the direct example and wrote code to replicate that without considering what the pattern really means..

And after the first ones and tens place is where your code fails to keep any reasonable number sequence.

Shisho - February 17, 2020, 6:06 pm

(Feb 17, 2020 07:05 PM)0xDEAD10CC Wrote: [\[CITE\]](#) If you can't see that OP had a “...” at the end of the example sequence you're "single minded". This means the sequence would continue.

The code is not designed to stop unless you want it to which is why the for loop can be changed to any minimum. Even a negative value if you want it to be since it uses signed integers...

0xDEAD10CC - February 17, 2020, 6:08 pm

Let me show you an example of what the pattern most likely is meant to be:

Code 2222

2221

2220

2212

2211

2210

2202

2201

2200

2122

2121

2120

2112

2111

2110

2102
2101
2100

2022
2021
2020

continued ...

Your code doesn't come close to this logic since it requires backtracking. Subtraction is only a very small piece of this sequence. If you've ever done basic "coin combination" algorithms using recursion before you would understand what is really going on here.

Here's a code sample (Note: isn't as simplified as it could be but it reduces recursive branching a little bit, it's actually close to an iterative method):

```
Code #include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#define N 4
```

```
void fill(int *arr, int value, size_t n, size_t length)
{
    while (n < length) {
        arr[n++] = value;
    }
}
```

```
void print(const int *arr, size_t length)
{
    size_t i;
    for (i = 0; i < length; i++) {
        printf("%d ", arr[i]);
    }

    putchar('\n');
}
```

```
void process_r(int *arr, int length, int k, int n)
{
    int x = length - 1;

    if (arr[n] == -1) {
        arr[n - 1]--;
        fill(arr, k, n, length);
        n--;
    } else {
        print(arr, length);

        if (arr[length - 1] == 0)
            putchar('\n');

        while (x >= n) {
```

```

        if (arr[x] > (x == n ? -1 : 0)) {
            arr[x]--;
            fill(arr, k, x + 1, length);
            break;
        }
        x--;
    }
}

if (n < 0)
    return;

process_r(arr, length, k, n);
}

void process(int *arr, size_t length, int k)
{
    process_r(arr, length, k, length - 1);
}

int main(void)
{
    int array[N];
    const int k = 2;
    fill(array, k, 0, N);
    process(array, N, k);
    return 0;
}

```

Output:

Code 2 2 2 2

2 2 2 1

2 2 2 0

2 2 1 2

2 2 1 1

2 2 1 0

2 2 0 2

2 2 0 1

2 2 0 0

2 1 2 2

2 1 2 1

2 1 2 0

2 1 1 2

2 1 1 1

2 1 1 0

2 1 0 2

2 1 0 1

2 1 0 0

2022
2021
2020

2012
2011
2010

2002
2001
2000

1222
1221
1220

1212
1211
1210

1202
1201
1200

1122
1121
1120

1112
1111
1110

1102
1101
1100

1022
1021
1020

1012
1011
1010

1002
1001
1000

0222
0221
0220

0212

0 2 1 1
0 2 1 0

0 2 0 2
0 2 0 1
0 2 0 0

0 1 2 2
0 1 2 1
0 1 2 0

0 1 1 2
0 1 1 1
0 1 1 0

0 1 0 2
0 1 0 1
0 1 0 0

0 0 2 2
0 0 2 1
0 0 2 0

0 0 1 2
0 0 1 1
0 0 1 0

0 0 0 2
0 0 0 1
0 0 0 0 Also works for N=4, k=5:
Code 5 5 5 5

5 5 5 4
5 5 5 3
5 5 5 2
5 5 5 1
5 5 5 0

5 5 4 5
5 5 4 4
5 5 4 3
5 5 4 2
5 5 4 1
5 5 4 0

5 5 3 5
5 5 3 4
5 5 3 3
5 5 3 2
5 5 3 1
5 5 3 0

5 5 2 5
5 5 2 4

5523
5522
5521
5520

5515
5514
5513
5512
5511
5510

5505
5504
5503
5502
5501
5500

5455
5454
5453
5452
5451
5450

5445
5444
5443
5442
5441
5440

5435
5434
5433
5432
5431
5430

5425
5424
5423
5422
5421
5420

5415
5414
5413
5412
5411
5410

5405
5404
5403
5402
5401
5400

5355
5354
5353
5352
5351
5350

5345
5344
5343
5342
5341
5340

5335
5334
5333
5332
5331
5330

5325
5324
5323
5322
5321
5320

5315
5314
5313
5312
5311
5310

5305
5304
5303
5302
5301
5300

5255
5254
5253

5 2 5 2
5 2 5 1
5 2 5 0

5 2 4 5
5 2 4 4
5 2 4 3
5 2 4 2
5 2 4 1
5 2 4 0

5 2 3 5
5 2 3 4
5 2 3 3
5 2 3 2
5 2 3 1
5 2 3 0

5 2 2 5
5 2 2 4
5 2 2 3
5 2 2 2
5 2 2 1
5 2 2 0

5 2 1 5
5 2 1 4
5 2 1 3
5 2 1 2
5 2 1 1
5 2 1 0

5 2 0 5
5 2 0 4
5 2 0 3
5 2 0 2
5 2 0 1
5 2 0 0

5 1 5 5
5 1 5 4
5 1 5 3
5 1 5 2
5 1 5 1
5 1 5 0

5 1 4 5
5 1 4 4
5 1 4 3
5 1 4 2
5 1 4 1
5 1 4 0

5135
5134
5133
5132
5131
5130

5125
5124
5123
5122
5121
5120

5115
5114
5113
5112
5111
5110

5105
5104
5103
5102
5101
5100

5055
5054
5053
5052
5051
5050

5045
5044
5043
5042
5041
5040

5035
5034
5033
5032
5031
5030

5025
5024
5023
5022

5021
5020

5015
5014
5013
5012
5011
5010

5005
5004
5003
5002
5001
5000

4555
4554
4553
4552
4551
4550

4545
4544
4543
4542
4541
4540

4535
4534
4533
4532
4531
4530

4525
4524
4523
4522
4521
4520

4515
4514
4513
4512
4511
4510

4505

4 5 0 4
4 5 0 3
4 5 0 2
4 5 0 1
4 5 0 0

4 4 5 5
4 4 5 4
4 4 5 3
4 4 5 2
4 4 5 1
4 4 5 0

4 4 4 5
4 4 4 4
4 4 4 3
4 4 4 2
4 4 4 1
4 4 4 0

4 4 3 5
4 4 3 4
4 4 3 3
4 4 3 2
4 4 3 1
4 4 3 0

4 4 2 5
4 4 2 4
4 4 2 3
4 4 2 2
4 4 2 1
4 4 2 0

4 4 1 5
4 4 1 4
4 4 1 3
4 4 1 2
4 4 1 1
4 4 1 0

4 4 0 5
4 4 0 4
4 4 0 3
4 4 0 2
4 4 0 1
4 4 0 0

4 3 5 5
4 3 5 4
4 3 5 3
4 3 5 2
4 3 5 1

4 3 5 0

4 3 4 5

4 3 4 4

4 3 4 3

4 3 4 2

4 3 4 1

4 3 4 0

4 3 3 5

4 3 3 4

4 3 3 3

4 3 3 2

4 3 3 1

4 3 3 0

4 3 2 5

4 3 2 4

4 3 2 3

4 3 2 2

4 3 2 1

4 3 2 0

4 3 1 5

4 3 1 4

4 3 1 3

4 3 1 2

4 3 1 1

4 3 1 0

4 3 0 5

4 3 0 4

4 3 0 3

4 3 0 2

4 3 0 1

4 3 0 0

4 2 5 5

4 2 5 4

4 2 5 3

4 2 5 2

4 2 5 1

4 2 5 0

4 2 4 5

4 2 4 4

4 2 4 3

4 2 4 2

4 2 4 1

4 2 4 0

4 2 3 5

4 2 3 4

4 2 3 3
4 2 3 2
4 2 3 1
4 2 3 0

4 2 2 5
4 2 2 4
4 2 2 3
4 2 2 2
4 2 2 1
4 2 2 0

4 2 1 5
4 2 1 4
4 2 1 3
4 2 1 2
4 2 1 1
4 2 1 0

4 2 0 5
4 2 0 4
4 2 0 3
4 2 0 2
4 2 0 1
4 2 0 0

4 1 5 5
4 1 5 4
4 1 5 3
4 1 5 2
4 1 5 1
4 1 5 0

4 1 4 5
4 1 4 4
4 1 4 3
4 1 4 2
4 1 4 1
4 1 4 0

4 1 3 5
4 1 3 4
4 1 3 3
4 1 3 2
4 1 3 1
4 1 3 0

4 1 2 5
4 1 2 4
4 1 2 3
4 1 2 2
4 1 2 1
4 1 2 0

4 1 1 5
4 1 1 4
4 1 1 3
4 1 1 2
4 1 1 1
4 1 1 0

4 1 0 5
4 1 0 4
4 1 0 3
4 1 0 2
4 1 0 1
4 1 0 0

4 0 5 5
4 0 5 4
4 0 5 3
4 0 5 2
4 0 5 1
4 0 5 0

4 0 4 5
4 0 4 4
4 0 4 3
4 0 4 2
4 0 4 1
4 0 4 0

4 0 3 5
4 0 3 4
4 0 3 3
4 0 3 2
4 0 3 1
4 0 3 0

4 0 2 5
4 0 2 4
4 0 2 3
4 0 2 2
4 0 2 1
4 0 2 0

4 0 1 5
4 0 1 4
4 0 1 3
4 0 1 2
4 0 1 1
4 0 1 0

4 0 0 5
4 0 0 4
4 0 0 3

4002
4001
4000

3555
3554
3553
3552
3551
3550

3545
3544
3543
3542
3541
3540

3535
3534
3533
3532
3531
3530

3525
3524
3523
3522
3521
3520

3515
3514
3513
3512
3511
3510

3505
3504
3503
3502
3501
3500

3455
3454
3453
3452
3451
3450

3 4 4 5
3 4 4 4
3 4 4 3
3 4 4 2
3 4 4 1
3 4 4 0

3 4 3 5
3 4 3 4
3 4 3 3
3 4 3 2
3 4 3 1
3 4 3 0

3 4 2 5
3 4 2 4
3 4 2 3
3 4 2 2
3 4 2 1
3 4 2 0

3 4 1 5
3 4 1 4
3 4 1 3
3 4 1 2
3 4 1 1
3 4 1 0

3 4 0 5
3 4 0 4
3 4 0 3
3 4 0 2
3 4 0 1
3 4 0 0

3 3 5 5
3 3 5 4
3 3 5 3
3 3 5 2
3 3 5 1
3 3 5 0

3 3 4 5
3 3 4 4
3 3 4 3
3 3 4 2
3 3 4 1
3 3 4 0

3 3 3 5
3 3 3 4
3 3 3 3
3 3 3 2

3331
3330

3325
3324
3323
3322
3321
3320

3315
3314
3313
3312
3311
3310

3305
3304
3303
3302
3301
3300

3255
3254
3253
3252
3251
3250

3245
3244
3243
3242
3241
3240

3235
3234
3233
3232
3231
3230

3225
3224
3223
3222
3221
3220

3215

3 2 1 4
3 2 1 3
3 2 1 2
3 2 1 1
3 2 1 0

3 2 0 5
3 2 0 4
3 2 0 3
3 2 0 2
3 2 0 1
3 2 0 0

3 1 5 5
3 1 5 4
3 1 5 3
3 1 5 2
3 1 5 1
3 1 5 0

3 1 4 5
3 1 4 4
3 1 4 3
3 1 4 2
3 1 4 1
3 1 4 0

3 1 3 5
3 1 3 4
3 1 3 3
3 1 3 2
3 1 3 1
3 1 3 0

3 1 2 5
3 1 2 4
3 1 2 3
3 1 2 2
3 1 2 1
3 1 2 0

3 1 1 5
3 1 1 4
3 1 1 3
3 1 1 2
3 1 1 1
3 1 1 0

3 1 0 5
3 1 0 4
3 1 0 3
3 1 0 2
3 1 0 1

3100

3055

3054

3053

3052

3051

3050

3045

3044

3043

3042

3041

3040

3035

3034

3033

3032

3031

3030

3025

3024

3023

3022

3021

3020

3015

3014

3013

3012

3011

3010

3005

3004

3003

3002

3001

3000

2555

2554

2553

2552

2551

2550

2545

2544

2 5 4 3
2 5 4 2
2 5 4 1
2 5 4 0

2 5 3 5
2 5 3 4
2 5 3 3
2 5 3 2
2 5 3 1
2 5 3 0

2 5 2 5
2 5 2 4
2 5 2 3
2 5 2 2
2 5 2 1
2 5 2 0

2 5 1 5
2 5 1 4
2 5 1 3
2 5 1 2
2 5 1 1
2 5 1 0

2 5 0 5
2 5 0 4
2 5 0 3
2 5 0 2
2 5 0 1
2 5 0 0

2 4 5 5
2 4 5 4
2 4 5 3
2 4 5 2
2 4 5 1
2 4 5 0

2 4 4 5
2 4 4 4
2 4 4 3
2 4 4 2
2 4 4 1
2 4 4 0

2 4 3 5
2 4 3 4
2 4 3 3
2 4 3 2
2 4 3 1
2 4 3 0

2 4 2 5
2 4 2 4
2 4 2 3
2 4 2 2
2 4 2 1
2 4 2 0

2 4 1 5
2 4 1 4
2 4 1 3
2 4 1 2
2 4 1 1
2 4 1 0

2 4 0 5
2 4 0 4
2 4 0 3
2 4 0 2
2 4 0 1
2 4 0 0

2 3 5 5
2 3 5 4
2 3 5 3
2 3 5 2
2 3 5 1
2 3 5 0

2 3 4 5
2 3 4 4
2 3 4 3
2 3 4 2
2 3 4 1
2 3 4 0

2 3 3 5
2 3 3 4
2 3 3 3
2 3 3 2
2 3 3 1
2 3 3 0

2 3 2 5
2 3 2 4
2 3 2 3
2 3 2 2
2 3 2 1
2 3 2 0

2 3 1 5
2 3 1 4
2 3 1 3

2 3 1 2
2 3 1 1
2 3 1 0

2 3 0 5
2 3 0 4
2 3 0 3
2 3 0 2
2 3 0 1
2 3 0 0

2 2 5 5
2 2 5 4
2 2 5 3
2 2 5 2
2 2 5 1
2 2 5 0

2 2 4 5
2 2 4 4
2 2 4 3
2 2 4 2
2 2 4 1
2 2 4 0

2 2 3 5
2 2 3 4
2 2 3 3
2 2 3 2
2 2 3 1
2 2 3 0

2 2 2 5
2 2 2 4
2 2 2 3
2 2 2 2
2 2 2 1
2 2 2 0

2 2 1 5
2 2 1 4
2 2 1 3
2 2 1 2
2 2 1 1
2 2 1 0

2 2 0 5
2 2 0 4
2 2 0 3
2 2 0 2
2 2 0 1
2 2 0 0

2155
2154
2153
2152
2151
2150

2145
2144
2143
2142
2141
2140

2135
2134
2133
2132
2131
2130

2125
2124
2123
2122
2121
2120

2115
2114
2113
2112
2111
2110

2105
2104
2103
2102
2101
2100

2055
2054
2053
2052
2051
2050

2045
2044
2043
2042

2041
2040

2035
2034
2033
2032
2031
2030

2025
2024
2023
2022
2021
2020

2015
2014
2013
2012
2011
2010

2005
2004
2003
2002
2001
2000

1555
1554
1553
1552
1551
1550

1545
1544
1543
1542
1541
1540

1535
1534
1533
1532
1531
1530

1525

1524
1523
1522
1521
1520

1515
1514
1513
1512
1511
1510

1505
1504
1503
1502
1501
1500

1455
1454
1453
1452
1451
1450

1445
1444
1443
1442
1441
1440

1435
1434
1433
1432
1431
1430

1425
1424
1423
1422
1421
1420

1415
1414
1413
1412
1411

1 4 1 0

1 4 0 5

1 4 0 4

1 4 0 3

1 4 0 2

1 4 0 1

1 4 0 0

1 3 5 5

1 3 5 4

1 3 5 3

1 3 5 2

1 3 5 1

1 3 5 0

1 3 4 5

1 3 4 4

1 3 4 3

1 3 4 2

1 3 4 1

1 3 4 0

1 3 3 5

1 3 3 4

1 3 3 3

1 3 3 2

1 3 3 1

1 3 3 0

1 3 2 5

1 3 2 4

1 3 2 3

1 3 2 2

1 3 2 1

1 3 2 0

1 3 1 5

1 3 1 4

1 3 1 3

1 3 1 2

1 3 1 1

1 3 1 0

1 3 0 5

1 3 0 4

1 3 0 3

1 3 0 2

1 3 0 1

1 3 0 0

1 2 5 5

1 2 5 4

1 2 5 3
1 2 5 2
1 2 5 1
1 2 5 0

1 2 4 5
1 2 4 4
1 2 4 3
1 2 4 2
1 2 4 1
1 2 4 0

1 2 3 5
1 2 3 4
1 2 3 3
1 2 3 2
1 2 3 1
1 2 3 0

1 2 2 5
1 2 2 4
1 2 2 3
1 2 2 2
1 2 2 1
1 2 2 0

1 2 1 5
1 2 1 4
1 2 1 3
1 2 1 2
1 2 1 1
1 2 1 0

1 2 0 5
1 2 0 4
1 2 0 3
1 2 0 2
1 2 0 1
1 2 0 0

1 1 5 5
1 1 5 4
1 1 5 3
1 1 5 2
1 1 5 1
1 1 5 0

1 1 4 5
1 1 4 4
1 1 4 3
1 1 4 2
1 1 4 1
1 1 4 0

1 1 3 5
1 1 3 4
1 1 3 3
1 1 3 2
1 1 3 1
1 1 3 0

1 1 2 5
1 1 2 4
1 1 2 3
1 1 2 2
1 1 2 1
1 1 2 0

1 1 1 5
1 1 1 4
1 1 1 3
1 1 1 2
1 1 1 1
1 1 1 0

1 1 0 5
1 1 0 4
1 1 0 3
1 1 0 2
1 1 0 1
1 1 0 0

1 0 5 5
1 0 5 4
1 0 5 3
1 0 5 2
1 0 5 1
1 0 5 0

1 0 4 5
1 0 4 4
1 0 4 3
1 0 4 2
1 0 4 1
1 0 4 0

1 0 3 5
1 0 3 4
1 0 3 3
1 0 3 2
1 0 3 1
1 0 3 0

1 0 2 5
1 0 2 4
1 0 2 3

1022
1021
1020

1015
1014
1013
1012
1011
1010

1005
1004
1003
1002
1001
1000

0555
0554
0553
0552
0551
0550

0545
0544
0543
0542
0541
0540

0535
0534
0533
0532
0531
0530

0525
0524
0523
0522
0521
0520

0515
0514
0513
0512
0511
0510

0505
0504
0503
0502
0501
0500

0455
0454
0453
0452
0451
0450

0445
0444
0443
0442
0441
0440

0435
0434
0433
0432
0431
0430

0425
0424
0423
0422
0421
0420

0415
0414
0413
0412
0411
0410

0405
0404
0403
0402
0401
0400

0355
0354
0353
0352

0351
0350

0345
0344
0343
0342
0341
0340

0335
0334
0333
0332
0331
0330

0325
0324
0323
0322
0321
0320

0315
0314
0313
0312
0311
0310

0305
0304
0303
0302
0301
0300

0255
0254
0253
0252
0251
0250

0245
0244
0243
0242
0241
0240

0235

0 2 3 4
0 2 3 3
0 2 3 2
0 2 3 1
0 2 3 0

0 2 2 5
0 2 2 4
0 2 2 3
0 2 2 2
0 2 2 1
0 2 2 0

0 2 1 5
0 2 1 4
0 2 1 3
0 2 1 2
0 2 1 1
0 2 1 0

0 2 0 5
0 2 0 4
0 2 0 3
0 2 0 2
0 2 0 1
0 2 0 0

0 1 5 5
0 1 5 4
0 1 5 3
0 1 5 2
0 1 5 1
0 1 5 0

0 1 4 5
0 1 4 4
0 1 4 3
0 1 4 2
0 1 4 1
0 1 4 0

0 1 3 5
0 1 3 4
0 1 3 3
0 1 3 2
0 1 3 1
0 1 3 0

0 1 2 5
0 1 2 4
0 1 2 3
0 1 2 2
0 1 2 1

0120

0115

0114

0113

0112

0111

0110

0105

0104

0103

0102

0101

0100

0055

0054

0053

0052

0051

0050

0045

0044

0043

0042

0041

0040

0035

0034

0033

0032

0031

0030

0025

0024

0023

0022

0021

0020

0015

0014

0013

0012

0011

0010

0005

0004

0003
0002
0001
0000