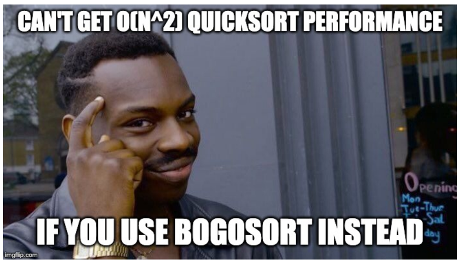
Quicksort

Quicksort is a simple sorting algorithm that works well in practice, but is not quite as quick from a theoretical point of view. In this task we will have a further look at this property.



Quicksort is a recursive algorithm:

- choose a pivot element p
- move the smaller elements of the to the left of the pivot p
- move the bigger elements to the right of the pivot p
- sort the sections right and left to the pivot p with recursive calls

The maximum recursion depth can be used as a runtime approximation.

To implement Quicksort in C++, there is a standard template library function called `std::stable_partition`, which partitions an array in a left and right half, without rearranging the elements within each half relative to the other („stable"). What the algorithm is doing behind the scenes isn't relevant for this task.

```
auto mid = std::stable_partition(bgn, end, [p](int x) {
    return x < p;
});
```

Here the area of from `bgn` to `end` exclusive is rearranged, in such a way that all elements `< p` are to the left and all other elements are to the right of p. The return value of the call in `mid` points to the first element satisfying `>= p`.

For the task we will be considering the following quicksort implementation:

```
#include <algorithm>

template<class Iterator>
int quicksort(Iterator bgn, Iterator end) {
    if (bgn == end)
        return 0;
    int p = *bgn;
    auto mid = std::stable_partition(bgn, end, [p](int x) {
        return x < p;
    });
    return 1 + std::max(quicksort(bgn, mid),
                        quicksort(mid+1, end));
}
```

The function is called with two iterators [1] on a range of integers sorting these. The return value represents the maximum recursion depth. Two points that might be useful to point out:

- the first element is choosen as the pivot.
- the pivot element p is placed at `mid`, as it is the first element and `stable_partition` is a stable algorithm.

Can you use these properties to design input data, such that the algorithm *exactly* reaches the recursion depth k. You are given n and k. Find an arbitrary permutation of $1, 2, \ldots n$, for which `quicksort` reaches the recursion depth k. If there is none print -1 instead.

Input

One line with n and k.

Output

an permutatoin from $1, 2, \ldots n$ or -1, if there is no solution. If there are multiple solutions, output any one of them.

Examples

Input	Output	Comments
`5 5`	`1 2 3 4 5`	Here all elements are shifted to the right

Input	Output	Comments
`4 3`	`2 1 3 4`	alternative solutions are: e.g.: 2 1 4 3 or 3 1 2 4.

Input	Output	Comments
`2 1`	`-1`	Both 1 2 and 2 1 have a recursion depth of 2.

Scoring

In general it holds that: $1 \leq k \leq n \leq 10^5$.

Subtask 1 (15 Points): $n \leq 10$

Subtask 2 (10 Points): $n \leq 20$

[1] i.e. `a.begin()` and `a.end()` of a `std::vector<int> a`.

Subtask 3 (10 Points): $n \leq 100$

Subtask 4 (15 Points): $n \leq 500$

Subtask 5 (10 Points): $n \leq 3000$

Subtask 6 (10 Points): $k = n$ (worst case)

Subtask 7 (10 Points): $k = \lceil \log_2 n \rceil$ (best case)

Subtask 8 (20 Points): No additional constraints

Constraints

Time limit: 1 s **Memory limit:** 256 MB