Hill climbing is a mathematical optimization algorithm. Given a function $f : \mathbb{R}^n \to \mathbb{R}$, the algorithm takes an arbitrary $\overrightarrow{x} \in \mathcal{D}_f$ and compares $f(\overrightarrow{x}), f(\Delta\overrightarrow{x})$. If the sfhit increases the function, it lets $\overrightarrow{x} := \Delta\overrightarrow{x}$; otherwise it lets $\overrightarrow{x} := -\Delta\overrightarrow{x}$. The process is repeated until some criteria is met.

Hill climbing can be used to optimize the greedy coloring of a graph. The use of this algorithm for greedy coloring is justified by a nice theorem.

Given a graph $G = (V, E)$, the parameter of the Greedy coloring algorithm $\mathcal{G}$ is an order $O = v_{i_1}, \ldots, v_{i_n}$ of the vertices, where the i_j define the coloring order of $V = \{v_1, \ldots, v_n\}$. Thus, shiftes in the parameter space correspond to changes in the coloring order.

Such permutations lack a nice property of real functions; namely, that their parameters can only be shifted in two directions: positive or negative. Instead of these two choices, an order O of n vertices has $n!$ permutations. We cannot try them out and chose the one that maximizes our function.

There is a property, however, which reveals permutations that may not necessarily improve the coloring, but are guaranteed not to worsen it. Such property reduces drastically the space of alternative permutations. In fact, given an initial ordering O, this space of alternative permutations is often so small that none of them improve the coloring. Thus, it is commen to generate k random initial orderings $O_1, \ldots, O_k$, so as to explore their (reduced) permutation spaces, in hope that one of their permutations will in fact improve the coloring.

> **Theorem.** Let $G = (V, E)$ a graph with a proper coloring of r colors $C = \{c_1, \ldots, c_r\}$. Let $V_{c_i} := \{x \in V : c(x) = c_i\}$. Let $P : C \to C$ a permutation of $c_1, \ldots, c_r$; i.e. $P(c_i) = c_j$ entails the ith color becomes the jth color.

Then $\mathcal{G}$ with the order $V_{P(c_1)}, \ldots, V_{P(c_r)}$ colors G with at most r colors.

The proof of this theorem is simple to do inductively over r. If $r = 1$ the case is trivial. Assume the theorem holds for $r = k$. Let $x_0 \in \bigcup_{i=0}^{k+1} V_{P(c_i)}$. If x_0 belongs to any of the first k sets, $\mathcal{G}$ colors it with one out of k colors by hypothesis. If $x_0 \in V_{P(c_{i+1})}$, the case where $\mathcal{G}$ colors it with a color less than $k + 2$ makes our statement true. Assume this is not the case. Then there is some $y_0 \in \Gamma(x_0)$ such that $c(y_0) = k + 1$. But then $y_0 \in V_{P(c_{i+1})}$. And since x_0 is also in this set, we should have $c(x_0) = c(y_0)$. Then the coloring is not proper. $(\perp)$

Informally, if $\mathcal{G}$ colored V into groups $V_{c_1}, \ldots, V_{c_r}$, then coloring first the vertices with color $P(c_1), P(c_2), \ldots, P(c_r)$ is at least as good as the original coloring. The permutation P that we must use is arbitrary: the theorem states nothing special about it. In general, one uses permutations that put vertices with highest colors

or highest degrees first, since these are the problematic ones. We will define two such permutations.

Permutation 0. The first permutation will order the vertices from last to first color. Thus, assuming $\mathcal{G}$ used r colors, the order will be $V_{c_r}, \ldots, V_{c_1}$.

To do this, we will create an array $\mathcal{D}$ of r pointers; each $\mathcal{D}[i]$ will point to an array of the vertices $v_1, \ldots, v_k$ whose color is i. Then the array of lenght n spanned by

$$\left\{\mathcal{D}[i]_{j_i} : 1 \leq i \leq r, 1 \leq j_i \leq |\mathcal{D}_i|\right\}$$

has the vertices of G ordered decreasingly by coloring. We do not know the value of each $|\mathcal{D}[i]|$, but we do know $\sum_{i=1}^{r} |\mathcal{D}(i)| = n$.

A pseudo-code implementation:

```
O = int Array[n]
D := Queue Array[r]
for 1 ≤ i ≤ n do
    push!(i, D[c(v_i)])
end
int i := 0
for queue in D do
    while queue ≠ ∅ do
        O[i] = pop(queue)
        i = i + 1
    end
end
return O
```

The first for loop is $O(n)$. Since $\sum_{i}^{r} |\mathcal{D}_i| = n$, the second for loop with its inner while is $O(n)$. $\therefore$ The algorithm is $O(n)$.

Permutation 1. The second permutation we will write is a simple cardinality permutation. The color with the most number of vertices goes first, the color with the least number of vertices goes last. There are at most $O(n)$ sets V_{c_i}. Using QSort we have $O(n \log n)$ complexity.

Permutation 2. We will order the colors using their divisibility. First will come all colors divisible by four, then all colors divisible by 2, and then all other colors. In pseudo-code,

$x = |\{c \in C : c \equiv 0 \mod 4\}|$
$y = |\{c \in C : c \equiv 0 \mod 2 \wedge c \not\equiv 0 \mod 4\}|$
$\mathcal{O} =$ **int** $Array[n]$
int $u = v = w = 0$
int $index$
for $i \in \{1, \ldots, r\}$ **do**
    while $\mathcal{D}[i] \neq \emptyset$ **do**
        $v = pop(\mathcal{D}[i])$
        if $i \equiv 0 \mod 4$ **do**
            $index = u$
            $u = u + 1$
        else if $i \equiv 0 \mod 2$ **do**
            $index = x + v$
            $v = v + 1$
        else do
            $index = x + y + w$
            $w = w + 1$
        fi
        $\mathcal{O}[index] = v$
    od
od

which is $O(n)$ of course.