# LAB-PRAC-11_FUN-PTR

## Name the Clones (p1v1d1)

---

**Name the Clones [20 marks]**

----------------------------------------------------------------------

**Problem Statement**
Mr C has a large number of clones that come to life when you call functions. He wants to give these clones, names for which he wants your help. In the input, you will be given two strictly positive integers n and k, separated by a space. You will have to generate names for the clones of Mr C. These names should all contain exactly k characters and all of these characters must be from the first n lower-case English alphabet characters (no upper-case characters, spaces or punctuation marks allowed). However, the allowed characters can repeat any number of times in a name.

Mr C has requested that the names that you suggest be in *lexicographically* increasing order. We explain lexicographic ordering below. Thus, you have to first output the lexicographically smallest string of length k you can construct out of the first n lower-case letters of the English alphabet. Then find the lexicographically next smallest string of length k and so on till you have generated all such names possible.

**Lexicographic ordering**
Just as given two digit sequences, say 1923 and 3122, we can say which digit sequence is "smaller" and which is "larger" (by interpreting the digit sequences as numbers), two sequences of alphabets, i.e. strings, can also be compared and given two sequences of alphabets, we can say which one is smaller and which one is larger.

The rules of doing so are pretty simple if the two strings are of the same length, which will be the case in this question. We first declare that the character 'a' is smaller than the character 'b', which is in turn smaller than the character 'c' and so on. To compare two strings, simply look at their first characters - the string with the larger character wins. If both strings have the same first character, then the second characters of the two strings are compared and so on.

Thus, we have "cat" = "cat" since the two strings are exactly the same but we have "cap" < "cat" since 'p' is smaller than 't'. Also, we have "mat" > "cat" since 'm' is larger than 'c' and also "aazd" < "abbb" since 'a' is smaller than 'b' (The third characters 'z' and 'b' do not get compared at all since "abbb" wins when the second characters get compared).

**Caution**

1. Using recursion is not compulsory in this question. However, you will have to write much less code if you use recursion.
2. Do not output names in incorrect order. The autograder will heavily penalize you if you do this since it will give you marks only if a correct name is output in its correct location.
3. We will not penalize you for stray newlines at the end of your output. However, do not have stray spaces at the end of each line of your output. You will not pass test cases if you have these.

**HINTS**:
This problem may seem very complicated if you try to write a loop to solve the problem but it can be very elegantly solved using recursion as we explain below. Suppose we want to generate all names of length 3 out of the first two alphabets, in lexicographic order. This can be solved if we first generate all names starting

with a and then generate all names starting with b (since names starting with a come before those starting with b in lexicographic order).

However, notice that generating all names starting with a, and that too in lexicographic order, we simply need to find all names of length 2 in lexicographic order, and simply add an a before them. But this is just a smaller version of the original problem which is why recursion can be used. Indeed, all names of length 2 (formed out of the first two alphabets) in alphabetical order are simply
aa
ab
ba
bb
and if we prepend (add to the beginning) an a to all these names, we get all names starting with a in alphabetical order!
aaa
aab
aba
abb

Write a function that takes partially filled in names and recursively calls itself to complete the names. Write a function generateNames(char* name, int k, int n, int left) which takes in four arguments

1. a character array name of length k+1 (k characters for the names and one for the NULL character)
2. the value of k (will help you know what is the length of the array)
3. the value of n (which all letters of the English alphabet can you use)
4. the value of how many letters are left to be filled in

The base case of the recursion can be left = 0 which means we have a complete name which can simply be printed. To print all names starting with the letter 'a', you could do something like the following:
char name[k+1];
name[k] = '\0'; // Do not forget the NULL character
name[0] = 'a';
generateNames(name, k, n, k - 1); // one character already filled and k-1 left to be filled

Use these hints to completely solve the problem
----------------------------------------------------------------------
**EXAMPLE**:
INPUT
2 3

OUTPUT:
aaa
aab
aba
abb
baa
bab
bba
bbb
----------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 2 3 | aaa<br>aab<br>aba<br>abb<br>baa<br>bab<br>bba<br>bbb |
| 5 2 | aa<br>ab<br>ac<br>ad<br>ae<br>ba<br>bb<br>bc<br>bd<br>be<br>ca<br>cb<br>cc<br>cd<br>ce<br>da<br>db<br>dc<br>dd<br>de<br>ea<br>eb<br>ec<br>ed<br>ee |
| 1 5 | aaaaa |
| 2 5 | aaaaa<br>aaaab<br>aaaba<br>aaabb<br>aabaa<br>aabab<br>aabba<br>aabbb<br>abaaa<br>abaab<br>ababa<br>ababb<br>abbaa<br>abbab |

| | |
|---|---|
| | abbba<br>abbbb<br>baaaa<br>baaab<br>baaba<br>baabb<br>babaa<br>babab<br>babba<br>babbb<br>bbaaa<br>bbaab<br>bbaba<br>bbabb<br>bbbaa<br>bbbab<br>bbbba<br>bbbbb |
| 5 4 | aaaa<br>aaab<br>aaac<br>aaad<br>aaae<br>aaba<br>aabb<br>aabc<br>aabd<br>aabe<br>aaca<br>aacb<br>aacc<br>aacd<br>aace<br>aada<br>aadb<br>aadc<br>aadd<br>aade<br>aaea<br>aaeb<br>aaec<br>aaed<br>aaee<br>abaa<br>abab<br>abac<br>abad<br>abae<br>abba<br>abbb<br>abbc<br>abbd<br>abbe<br>abca<br>abcb<br>abcc<br>abcd |

```
abce
abda
abdb
abdc
abdd
abde
abea
abeb
abec
abed
abee
acaa
acab
acac
acad
acae
acba
acbb
acbc
acbd
acbe
acca
accb
accc
accd
acce
acda
acdb
acdc
acdd
acde
acea
aceb
acec
aced
acee
adaa
adab
adac
adad
adae
adba
adbb
adbc
adbd
adbe
adca
adcb
adcc
adcd
adce
adda
addb
addc
addd
adde
adea
adeb
```

```
adec
aded
adee
aeaa
aeab
aeac
aead
aeae
aeba
aebb
aebc
aebd
aebe
aeca
aecb
aecc
aecd
aece
aeda
aedb
aedc
aedd
aede
aeea
aeeb
aeec
aeed
aeee
baaa
baab
baac
baad
baae
baba
babb
babc
babd
babe
baca
bacb
bacc
bacd
bace
bada
badb
badc
badd
bade
baea
baeb
baec
baed
baee
bbaa
bbab
bbac
bbad
bbae
```

```
bbba
bbbb
bbbc
bbbd
bbbe
bbca
bbcb
bbcc
bbcd
bbce
bbda
bbdb
bbdc
bbdd
bbde
bbea
bbeb
bbec
bbed
bbee
bcaa
bcab
bcac
bcad
bcae
bcba
bcbb
bcbc
bcbd
bcbe
bcca
bccb
bccc
bccd
bcce
bcda
bcdb
bcdc
bcdd
bcde
bcea
bceb
bcec
bced
bcee
bdaa
bdab
bdac
bdad
bdae
bdba
bdbb
bdbc
bdbd
bdbe
bdca
bdcb
bdcc
```

```
bdcd
bdce
bdda
bddb
bddc
bddd
bdde
bdea
bdeb
bdec
bded
bdee
beaa
beab
beac
bead
beae
beba
bebb
bebc
bebd
bebe
beca
becb
becc
becd
bece
beda
bedb
bedc
bedd
bede
beea
beeb
beec
beed
beee
caaa
caab
caac
caad
caae
caba
cabb
cabc
cabd
cabe
caca
cacb
cacc
cacd
cace
cada
cadb
cadc
cadd
cade
caea
```

```
caeb
caec
caed
caee
cbaa
cbab
cbac
cbad
cbae
cbba
cbbb
cbbc
cbbd
cbbe
cbca
cbcb
cbcc
cbcd
cbce
cbda
cbdb
cbdc
cbdd
cbde
cbea
cbeb
cbec
cbed
cbee
ccaa
ccab
ccac
ccad
ccae
ccba
ccbb
ccbc
ccbd
ccbe
ccca
cccb
cccc
cccd
ccce
ccda
ccdb
ccdc
ccdd
ccde
ccea
cceb
ccec
cced
ccee
cdaa
cdab
cdac
cdad
```

```
cdae
cdba
cdbb
cdbc
cdbd
cdbe
cdca
cdcb
cdcc
cdcd
cdce
cdda
cddb
cddc
cddd
cdde
cdea
cdeb
cdec
cded
cdee
ceaa
ceab
ceac
cead
ceae
ceba
cebb
cebc
cebd
cebe
ceca
cecb
cecc
cecd
cece
ceda
cedb
cedc
cedd
cede
ceea
ceeb
ceec
ceed
ceee
daaa
daab
daac
daad
daae
daba
dabb
dabc
dabd
dabe
daca
dacb
```

```
dacc
dacd
dace
dada
dadb
dadc
dadd
dade
daea
daeb
daec
daed
daee
dbaa
dbab
dbac
dbad
dbae
dbba
dbbb
dbbc
dbbd
dbbe
dbca
dbcb
dbcc
dbcd
dbce
dbda
dbdb
dbdc
dbdd
dbde
dbea
dbeb
dbec
dbed
dbee
dcaa
dcab
dcac
dcad
dcae
dcba
dcbb
dcbc
dcbd
dcbe
dcca
dccb
dccc
dccd
dcce
dcda
dcdb
dcdc
dcdd
dcde
```

```
dcea
dceb
dcec
dced
dcee
ddaa
ddab
ddac
ddad
ddae
ddba
ddbb
ddbc
ddbd
ddbe
ddca
ddcb
ddcc
ddcd
ddce
ddda
dddb
dddc
dddd
ddde
ddea
ddeb
ddec
dded
ddee
deaa
deab
deac
dead
deae
deba
debb
debc
debd
debe
deca
decb
decc
decd
dece
deda
dedb
dedc
dedd
dede
deea
deeb
deec
deed
deee
eaaa
eaab
eaac
```

```
eaad
eaae
eaba
eabb
eabc
eabd
eabe
eaca
eacb
eacc
eacd
eace
eada
eadb
eadc
eadd
eade
eaea
eaeb
eaec
eaed
eaee
ebaa
ebab
ebac
ebad
ebae
ebba
ebbb
ebbc
ebbd
ebbe
ebca
ebcb
ebcc
ebcd
ebce
ebda
ebdb
ebdc
ebdd
ebde
ebea
ebeb
ebec
ebed
ebee
ecaa
ecab
ecac
ecad
ecae
ecba
ecbb
ecbc
ecbd
ecbe
ecca
```

```
eccb
eccc
eccd
ecce
ecda
ecdb
ecdc
ecdd
ecde
ecea
eceb
ecec
eced
ecee
edaa
edab
edac
edad
edae
edba
edbb
edbc
edbd
edbe
edca
edcb
edcc
edcd
edce
edda
eddb
eddc
eddd
edde
edea
edeb
edec
eded
edee
eeaa
eeab
eeac
eead
eeae
eeba
eebb
eebc
eebd
eebe
eeca
eecb
eecc
eecd
eece
eeda
eedb
eedc
eedd
```

| | |
|---|---|
| | eede<br>eeea<br>eeeb<br>eeec<br>eeed<br>eeee |
| 3 6 | aaaaaa<br>aaaaab<br>aaaaac<br>aaaaba<br>aaaabb<br>aaaabc<br>aaaaca<br>aaaacb<br>aaaacc<br>aaabaa<br>aaabab<br>aaabac<br>aaabba<br>aaabbb<br>aaabbc<br>aaabca<br>aaabcb<br>aaabcc<br>aaacaa<br>aaacab<br>aaacac<br>aaacba<br>aaacbb<br>aaacbc<br>aaacca<br>aaaccb<br>aaaccc<br>aabaaa<br>aabaab<br>aabaac<br>aababa<br>aababb<br>aababc<br>aabaca<br>aabacb<br>aabacc<br>aabbaa<br>aabbab<br>aabbac<br>aabbba<br>aabbbb<br>aabbbc<br>aabbca<br>aabbcb<br>aabbcc<br>aabcaa<br>aabcab<br>aabcac<br>aabcba<br>aabcbb<br>aabcbc |

```
aabcca
aabccb
aabccc
aacaaa
aacaab
aacaac
aacaba
aacabb
aacabc
aacaca
aacacb
aacacc
aacbaa
aacbab
aacbac
aacbba
aacbbb
aacbbc
aacbca
aacbcb
aacbcc
aaccaa
aaccab
aaccac
aaccba
aaccbb
aaccbc
aaccca
aacccb
aacccc
abaaaa
abaaab
abaaac
abaaba
abaabb
abaabc
abaaca
abaacb
abaacc
ababaa
ababab
ababac
ababba
ababbb
ababbc
ababca
ababcb
ababcc
abacaa
abacab
abacac
abacba
abacbb
abacbc
abacca
abaccb
abaccc
abbaaa
```

```
abbaab
abbaac
abbaba
abbabb
abbabc
abbaca
abbacb
abbacc
abbbaa
abbbab
abbbac
abbbba
abbbbb
abbbbc
abbbca
abbbcb
abbbcc
abbcaa
abbcab
abbcac
abbcba
abbcbb
abbcbc
abbcca
abbccb
abbccc
abcaaa
abcaab
abcaac
abcaba
abcabb
abcabc
abcaca
abcacb
abcacc
abcbaa
abcbab
abcbac
abcbba
abcbbb
abcbbc
abcbca
abcbcb
abcbcc
abccaa
abccab
abccac
abccba
abccbb
abccbc
abccca
abcccb
abcccc
acaaaa
acaaab
acaaac
acaaba
acaabb
```

```
acaabc
acaaca
acaacb
acaacc
acabaa
acabab
acabac
acabba
acabbb
acabbc
acabca
acabcb
acabcc
acacaa
acacab
acacac
acacba
acacbb
acacbc
acacca
acaccb
acaccc
acbaaa
acbaab
acbaac
acbaba
acbabb
acbabc
acbaca
acbacb
acbacc
acbbaa
acbbab
acbbac
acbbba
acbbbb
acbbbc
acbbca
acbbcb
acbbcc
acbcaa
acbcab
acbcac
acbcba
acbcbb
acbcbc
acbcca
acbccb
acbccc
accaaa
accaab
accaac
accaba
accabb
accabc
accaca
accacb
accacc
```

```
accbaa
accbab
accbac
accbba
accbbb
accbbc
accbca
accbcb
accbcc
acccaa
acccab
acccac
acccba
acccbb
acccbc
acccca
accccb
accccc
baaaaa
baaaab
baaaac
baaaba
baaabb
baaabc
baaaca
baaacb
baaacc
baabaa
baabab
baabac
baabba
baabbb
baabbc
baabca
baabcb
baabcc
baacaa
baacab
baacac
baacba
baacbb
baacbc
baacca
baaccb
baaccc
babaaa
babaab
babaac
bababa
bababb
bababc
babaca
babacb
babacc
babbaa
babbab
babbac
babbba
```

```
babbbb
babbbc
babbca
babbcb
babbcc
babcaa
babcab
babcac
babcba
babcbb
babcbc
babcca
babccb
babccc
bacaaa
bacaab
bacaac
bacaba
bacabb
bacabc
bacaca
bacacb
bacacc
bacbaa
bacbab
bacbac
bacbba
bacbbb
bacbbc
bacbca
bacbcb
bacbcc
baccaa
baccab
baccac
baccba
baccbb
baccbc
baccca
bacccb
bacccc
bbaaaa
bbaaab
bbaaac
bbaaba
bbaabb
bbaabc
bbaaca
bbaacb
bbaacc
bbabaa
bbabab
bbabac
bbabba
bbabbb
bbabbc
bbabca
bbabcb
```

```
bbabcc
bbacaa
bbacab
bbacac
bbacba
bbacbb
bbacbc
bbacca
bbaccb
bbaccc
bbbaaa
bbbaab
bbbaac
bbbaba
bbbabb
bbbabc
bbbaca
bbbacb
bbbacc
bbbbaa
bbbbab
bbbbac
bbbbba
bbbbbb
bbbbbc
bbbbca
bbbbcb
bbbbcc
bbbcaa
bbbcab
bbbcac
bbbcba
bbbcbb
bbbcbc
bbbcca
bbbccb
bbbccc
bbcaaa
bbcaab
bbcaac
bbcaba
bbcabb
bbcabc
bbcaca
bbcacb
bbcacc
bbcbaa
bbcbab
bbcbac
bbcbba
bbcbbb
bbcbbc
bbcbca
bbcbcb
bbcbcc
bbccaa
bbccab
bbccac
```

```
bbccba
bbccbb
bbccbc
bbccca
bbcccb
bbcccc
bcaaaa
bcaaab
bcaaac
bcaaba
bcaabb
bcaabc
bcaaca
bcaacb
bcaacc
bcabaa
bcabab
bcabac
bcabba
bcabbb
bcabbc
bcabca
bcabcb
bcabcc
bcacaa
bcacab
bcacac
bcacba
bcacbb
bcacbc
bcacca
bcaccb
bcaccc
bcbaaa
bcbaab
bcbaac
bcbaba
bcbabb
bcbabc
bcbaca
bcbacb
bcbacc
bcbbaa
bcbbab
bcbbac
bcbbba
bcbbbb
bcbbbc
bcbbca
bcbbcb
bcbbcc
bcbcaa
bcbcab
bcbcac
bcbcba
bcbcbb
bcbcbc
bcbcca
```

```
bcbccb
bcbccc
bccaaa
bccaab
bccaac
bccaba
bccabb
bccabc
bccaca
bccacb
bccacc
bccbaa
bccbab
bccbac
bccbba
bccbbb
bccbbc
bccbca
bccbcb
bccbcc
bcccaa
bcccab
bcccac
bcccba
bcccbb
bcccbc
bcccca
bccccb
bccccc
caaaaa
caaaab
caaaac
caaaba
caaabb
caaabc
caaaca
caaacb
caaacc
caabaa
caabab
caabac
caabba
caabbb
caabbc
caabca
caabcb
caabcc
caacaa
caacab
caacac
caacba
caacbb
caacbc
caacca
caaccb
caaccc
cabaaa
cabaab
```

```
cabaac
cababa
cababb
cababc
cabaca
cabacb
cabacc
cabbaa
cabbab
cabbac
cabbba
cabbbb
cabbbc
cabbca
cabbcb
cabbcc
cabcaa
cabcab
cabcac
cabcba
cabcbb
cabcbc
cabcca
cabccb
cabccc
cacaaa
cacaab
cacaac
cacaba
cacabb
cacabc
cacaca
cacacb
cacacc
cacbaa
cacbab
cacbac
cacbba
cacbbb
cacbbc
cacbca
cacbcb
cacbcc
caccaa
caccab
caccac
caccba
caccbb
caccbc
caccca
cacccb
cacccc
cbaaaa
cbaaab
cbaaac
cbaaba
cbaabb
cbaabc
```

```
cbaaca
cbaacb
cbaacc
cbabaa
cbabab
cbabac
cbabba
cbabbb
cbabbc
cbabca
cbabcb
cbabcc
cbacaa
cbacab
cbacac
cbacba
cbacbb
cbacbc
cbacca
cbaccb
cbaccc
cbbaaa
cbbaab
cbbaac
cbbaba
cbbabb
cbbabc
cbbaca
cbbacb
cbbacc
cbbbaa
cbbbab
cbbbac
cbbbba
cbbbbb
cbbbbc
cbbbca
cbbbcb
cbbbcc
cbbcaa
cbbcab
cbbcac
cbbcba
cbbcbb
cbbcbc
cbbcca
cbbccb
cbbccc
cbcaaa
cbcaab
cbcaac
cbcaba
cbcabb
cbcabc
cbcaca
cbcacb
cbcacc
cbcbaa
```

```
cbcbab
cbcbac
cbcbba
cbcbbb
cbcbbc
cbcbca
cbcbcb
cbcbcc
cbccaa
cbccab
cbccac
cbccba
cbccbb
cbccbc
cbccca
cbcccb
cbcccc
ccaaaa
ccaaab
ccaaac
ccaaba
ccaabb
ccaabc
ccaaca
ccaacb
ccaacc
ccabaa
ccabab
ccabac
ccabba
ccabbb
ccabbc
ccabca
ccabcb
ccabcc
ccacaa
ccacab
ccacac
ccacba
ccacbb
ccacbc
ccacca
ccaccb
ccaccc
ccbaaa
ccbaab
ccbaac
ccbaba
ccbabb
ccbabc
ccbaca
ccbacb
ccbacc
ccbbaa
ccbbab
ccbbac
ccbbba
ccbbbb
```

| | |
|---|---|
| | ccbbbc |
| | ccbbca |
| | ccbbcb |
| | ccbbcc |
| | ccbcaa |
| | ccbcab |
| | ccbcac |
| | ccbcba |
| | ccbcbb |
| | ccbcbc |
| | ccbcca |
| | ccbccb |
| | ccbccc |
| | cccaaa |
| | cccaab |
| | cccaac |
| | cccaba |
| | cccabb |
| | cccabc |
| | cccaca |
| | cccacb |
| | cccacc |
| | cccbaa |
| | cccbab |
| | cccbac |
| | cccbba |
| | cccbbb |
| | cccbbc |
| | cccbca |
| | cccbcb |
| | cccbcc |
| | ccccaa |
| | ccccab |
| | ccccac |
| | ccccba |
| | ccccbb |
| | ccccbc |
| | cccccca |
| | cccccb |
| | cccccc |

# The Race of the Clones (p1v2d1)

**The Race of the Clones [20 marks]**

----------------------------------------------------------------------

**Problem Statement**
Mr C and his clones are having a 400 metre race around the track. There are n clones in total and each of them completes the race in a different amount of time (all times are strictly positive integers). In the first line of the input, you will be given n and in the next line, you will be given the n times as strictly positive integers, separated by a space.

At time t = 0, all the clones start the race. However, if a clone finishes a race and finds that not all other clones have finished, that clone feels bad and starts racing again to give others company. Thus, the clones stop running only when all of them reach the finish line simultaneously.

In your output, first give the time it takes for the clones to stop running if only the first two of the n clones were participating in the race. In the next line, give the time it takes for the clones to stop running if all n of them are in the race.

**Caution**

1. The running times of the clones is not guaranteed to be in increasing or decreasing order. Two or more clones may have the same running time.
2. The solution to this question does not necessarily involve recursion. However, you should try to write code in a modular manner, using functions, to make it easier to write and debug.
3. Be careful about extra/missing lines and extra/missing spaces in your output.

----------------------------------------------------------------------

**EXAMPLE**:
INPUT
5
2 3 5 8 4

OUTPUT:
6
120

**Explanation**: If the first two clones were the only ones running, then since the first clone finishes in 2 seconds and the second in 3 seconds, the will reach the finish line together after 6 seconds have passed. However, if all the clones are running, then the first time they will all reach the finish line together is after 120 seconds.
----------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[ Points]**

There will be partial grading in this question. There are two lines in your output. They are worth 25% and 75% of the weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 5<br>2 3 5 8 4 | 6<br>120 |
| 6<br>6 5 4 3 2 1 | 30<br>60 |
| 9<br>6 6 6 6 6 6 6 6 6 | 6<br>6 |
| 10<br>12 6 1 2 3 4 1 2 3 4 | 12<br>12 |
| 10 | 3 |

| | |
|---|---|
| 1 3 5 8 4 300 1231 1200 200 500 | 7386000 |
| 15<br>31 2 1 2 1 100 200 300 11 30 40 33 44 91 13 | 62<br>18618600 |

# Partial Palindrome (p1v3d1)

**Partial Palindrome [20 marks]**

----------------------------------------------------------------------

**Problem Statement**
In the first line of the input, we will give you a string with at most 99 characters. The string will only contain lower-case English alphabet letters. In the first line of the output, you have to print the minimum number of characters that must be appended (i.e. put at the end of the string) to make it a palindrome and in the second line of the output, you have to print the resultant palindrome. If the string given to you is itself a palindrome, print 0 in the first line and the string itself in the second line.

**Caution**

1. There is no specific need to use recursion to solve this problem. However, you may want to write a modular code with functions to make your solution easier to read and easier to debug.
2. Be careful about extra/missing lines and extra/missing spaces in your output.

----------------------------------------------------------------------

**EXAMPLE 1**:
INPUT
abede

OUTPUT:
2
abedeba

**Explanation**: the string is itself not a palindrome. Appending a single character can never make it a palindrome since the string abede* is never a palindrome no matter what character we use in place of *. However, if we append two characters, namely b and a, then the string becomes a palindrome.

**EXAMPLE 2**:
INPUT
mom

OUTPUT:
0
mom

**Explanation**: the string is itself a palindrome.
----------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

### All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| abede | 2<br>abedeba |
| mom | 0<br>mom |
| abcdefghijklmnopqrstuvwxyz | 25<br>abcdefghijklmnopqrstuvwxyzyxwvutsrqponmlkjihgfedcba |
| aaaaabcdefghijjihgfedcbaaaaa | 0<br>aaaaabcdefghijjihgfedcbaaaaa |
| xyzzyxaaaaabcdefghijjihgfedcbaaaaa | 6<br>xyzzyxaaaaabcdefghijjihgfedcbaaaaaxyzzyx |
| iwouldlovetolearnmalayalam | 17<br>iwouldlovetolearnmalayalamnraelotevoldluowi |

## Growth Curve (p2v1d1)

**Growth Curve [20 marks]**

------------------------------------------------------------------

**Problem Statement**
Mr C is fascinated with non-decreasing functions since they represent growth. A function $f: \mathbb{N} \rightarrow \mathbb{N}$ (where $\mathbb{N}$ is the set of natural numbers) is called non-decreasing if whenever $i \geq j$, we have $f(i) \geq f(j)$ where $i,j \in \mathbb{N}$. Thus, as we move from left to right on then number line, the function value may stay the same or go up, but never go down.

In the input, you will be given two non-negative integers n and MAX, separated by a space. As your output, in separate lines, you will have to list all non-decreasing functions of the form
\[
f: [0 : n-1] \rightarrow [0 : \text{MAX}]
\]
by printing the values those functions take on the inputs $0, 1, 2, \ldots, n-1$, separated by a space. However, you have to list these functions in a certain *lexicographic* order, as described below.

**Lexicographic ordering**
Just as given two digit sequences, say 1923 and 3122, we can say which digit sequence is "smaller" and which is "larger" (by interpreting the digit sequences as numbers), given two sequences of non-negative numbers, say [0 8 11 4] and [1 0 2 3], we can say which digit sequence is "smaller" and which is "larger" in a very intuitive manner.

The rules of doing so are pretty simple if the two sequences are of the same length, which will be the case in this question (both sequences will have only n elements). To compare two sequences, simply look at the first numbers in the two sequences - the sequence with the larger number wins and is declared larger. If both sequences have the same first number, then the second numbers of the two sequences are compared and so on.

Note that we can use exactly these very rules to pronounce 3122 > 1923 but these rules allow us to declare that [0 8 11 4] < [1 0 2 3] since 0 < 1 (The second numbers 8 and 0 do not get compared at all since [1 0 2 3] wins as the larger number when the first numbers get compared). Similarly we have [3 5 9 3] < [3 8 1 1].

In your output, you will have to first output the "smallest function" i.e. the function that corresponds to the lexicographically smallest sequence, then the next smallest function, then the next smallest function and so on.

**Caution**

1. The integers n and MAX are non-negative. However, they can be zero.
2. Using recursion is not compulsory in this question. However, you will have to write much less code if you use recursion.
3. Do not output functions in incorrect order. The autograder will heavily penalize you if you do this since it will give you marks only if a correct function is output in its correct location.
4. We will not penalize you for stray newlines at the end of your output. However, do not have stray spaces at the end of each line of your output. You will not pass test cases if you have these.

**HINTS**:
This problem may seem complicated if you try to write a loop to solve the problem but it can be very elegantly solved using recursion as we explain below. Suppose we want to generate all functions of the form $f: [0 : 2] \rightarrow [0 : 1]$ (say n = 3 and MAX = 1), in lexicographic order. This can be solved if we first generate all functions that satisfy $f(0) = 0$ and then generate all functions that satisfy $f(0) = 1$ (since functions that have $f(0) = 0$ come before those that have $f(0) = 1$ in lexicographic order).

However, notice that in order to generate all functions with $f(0) = 0$, and that too in lexicographic order, we simply need to find all functions of the form $f: [1 : 2] \rightarrow [0 : 1]$. However, this can simply be thought of as a smaller version of the original problem.

Similarly, in order to generate all functions with $f(0) = 1$, and that too in lexicographic order, we simply need to find all functions of the form $f: [1 : 2] \rightarrow [1]$ (we are only interested in non-decreasing functions and since $f(0) = 1$, we must have $f(1), f(2) \geq 1$ but since $MAX = 1$, we are stuck). There is only one such function [1 1 1].

Write a function that takes partially filled in integer sequences (corresponding to functions) and recursively calls itself to complete the function descriptions. Write a function genFunctions(int *func, int n, int MAX, int i, int y) which takes in five arguments

1. an integer array func with n elements to describe the function values on 0 : n-1
2. the value of n (will help you know what is the length of the array)
3. the value of MAX (which all numbers are valid outputs of the function)
4. the value of i indicating that we have already decided f(0), f(1), ... f(i-1)
5. the value of y = f(i - 1) (you need this since you need to ensure that $f(j) \geq f(i - 1)$ for all $j > i - 1$)

The base case of the recursion can be i = n which indicates that we have filled in the complete sequence corresponding to a function which can simply be printed. To print all functions that satisfy $f(0) = 0$, you could do something like the following:
int func[n];
func[0] = 0;
genFunctions(func, n, MAX, 1, 0); // one function value f(0) = 0 already filled in, rest of function values to

be filled in non-decreasing order

To print all functions that satisfy $f(0) = 0$ and $f(1) = 5$ (assuming $MAX \geq 5$), you could do something like the following:
int func[n];
func[0] = 0;
func[1] = 5;
genFunctions(func, n, MAX, 2, 5); // Two values filled in with current max function value 5 rest to be filled in

Use these hints to completely solve the problem by writing the definition of the function genFunctions. Remember that genFunctions will need to call itself with suitable parameters.

----------------------------------------------------------------------

**EXAMPLE**:
INPUT
3 1

OUTPUT:
0 0 0
0 0 1
0 1 1
1 1 1

**Explanation**: Note that the function [0 1 0] is not on the list since it is not a non-decreasing function. Many other such invalid functions (e.g. [1 0 1] etc) are not on the list.

----------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 3 1 | 0 0 0<br>0 0 1<br>0 1 1<br>1 1 1 |
| 2 4 | 0 0<br>0 1<br>0 2<br>0 3<br>0 4<br>1 1<br>1 2<br>1 3<br>1 4<br>2 2<br>2 3 |

| | |
|---|---|
| | 2 4<br>3 3<br>3 4<br>4 4 |
| 5 0 | 0 0 0 0 0 |
| 5 1 | 0 0 0 0 0<br>0 0 0 0 1<br>0 0 0 1 1<br>0 0 1 1 1<br>0 1 1 1 1<br>1 1 1 1 1 |
| 4 4 | 0 0 0 0<br>0 0 0 1<br>0 0 0 2<br>0 0 0 3<br>0 0 0 4<br>0 0 1 1<br>0 0 1 2<br>0 0 1 3<br>0 0 1 4<br>0 0 2 2<br>0 0 2 3<br>0 0 2 4<br>0 0 3 3<br>0 0 3 4<br>0 0 4 4<br>0 1 1 1<br>0 1 1 2<br>0 1 1 3<br>0 1 1 4<br>0 1 2 2<br>0 1 2 3<br>0 1 2 4<br>0 1 3 3<br>0 1 3 4<br>0 1 4 4<br>0 2 2 2<br>0 2 2 3<br>0 2 2 4<br>0 2 3 3<br>0 2 3 4<br>0 2 4 4<br>0 3 3 3<br>0 3 3 4<br>0 3 4 4<br>0 4 4 4<br>1 1 1 1<br>1 1 1 2<br>1 1 1 3<br>1 1 1 4<br>1 1 2 2<br>1 1 2 3<br>1 1 2 4<br>1 1 3 3<br>1 1 3 4<br>1 1 4 4<br>1 2 2 2 |

| | |
|---|---|
| | 1 2 2 3<br>1 2 2 4<br>1 2 3 3<br>1 2 3 4<br>1 2 4 4<br>1 3 3 3<br>1 3 3 4<br>1 3 4 4<br>1 4 4 4<br>2 2 2 2<br>2 2 2 3<br>2 2 2 4<br>2 2 3 3<br>2 2 3 4<br>2 2 4 4<br>2 3 3 3<br>2 3 3 4<br>2 3 4 4<br>2 4 4 4<br>3 3 3 3<br>3 3 3 4<br>3 3 4 4<br>3 4 4 4<br>4 4 4 4 |
| 6 2 | 0 0 0 0 0 0<br>0 0 0 0 0 1<br>0 0 0 0 0 2<br>0 0 0 0 1 1<br>0 0 0 0 1 2<br>0 0 0 0 2 2<br>0 0 0 1 1 1<br>0 0 0 1 1 2<br>0 0 0 1 2 2<br>0 0 0 2 2 2<br>0 0 1 1 1 1<br>0 0 1 1 1 2<br>0 0 1 1 2 2<br>0 0 1 2 2 2<br>0 0 2 2 2 2<br>0 1 1 1 1 1<br>0 1 1 1 1 2<br>0 1 1 1 2 2<br>0 1 1 2 2 2<br>0 1 2 2 2 2<br>0 2 2 2 2 2<br>1 1 1 1 1 1<br>1 1 1 1 1 2<br>1 1 1 1 2 2<br>1 1 1 2 2 2<br>1 1 2 2 2 2<br>1 2 2 2 2 2<br>2 2 2 2 2 2 |

## The Family Tree of Mr C (p2v2d1)

**The Family Tree of Mr C [20 marks]**

----------------------------------------------------------------------

**Problem Statement**
Believe it or not, Mr C too has parents, grandparents, and higher ancestors, and is quite a darling to them. In this problem, we will get to meet some of them. All of Mr C's ancestors have names that are strictly positive integers. These will be given to you in a certain format. In the first line of the input, you will be given n, a strictly positive integer. In the second line of the input, you will be given n strictly positive integers, separated by a space. These represent the names of n of Mr C's ancestors. Store these names in an array called names. Below we tell you how these names are arranged.

1. names[0] is the name of Mr C's mother
2. names[1] is the name of Mr C's father
3. For any index i, the index 2*(i+1) stores the name of the mother of the person at index i
4. For any index i, the index 2*(i+1) + 1 stores the name of the father of the person at index i

Thus, the index 4 stores the name of Mr C's father's mother, i.e. Mr C's grandmother and index 6 stores the name of Mr C's mother's mother's mother i.e. Mr C's great grand mother.

In the third line of the input, we will give you four strictly positive integers, separated by a space, which are supposed to be names of Mr C's ancestors. For each name, you have to print one of the following messages, on separate lines.

1. If the name does not appear at all in the list, print "NO RELATIONS" (without quotes)
2. If the name appears more than once in the list, print "MULTIPLE RELATIONS" (without quotes)
3. If the name appears just once in the list, print the relation that person has with Mr C. The relation must be printed in ALL CAPITAL letters, as shown in the example below.

**Caution**

1. Be careful to count how "grand" a certain ancestor is e.g. is the person a great grand parent or a great great grand parent?
2. Hint: even indices always store names of "mothers" and odd indices always store names of "fathers".
3. Be careful about extra/missing lines and extra/missing spaces in your output.

----------------------------------------------------------------------

**EXAMPLE**:
INPUT
```
10
1 2 3 4 5 6 7 8 9 10
1 5 7 10
```

OUTPUT:
```
MOTHER
GRAND MOTHER
GREAT GRAND MOTHER
GREAT GRAND FATHER
```

**Explanation**:

1. index 0 is mother
2. index 4 is father's mother
3. index 6 is mother's mother's mother
4. index 9 is mother's father's father

---------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are four lines in your output. Printing each line correctly, in the correct order, carries 25% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 10<br>1 2 3 4 5 6 7 8 9 10<br>1 5 7 10 | MOTHER<br>GRAND MOTHER<br>GREAT GRAND MOTHER<br>GREAT GRAND FATHER |
| 20<br>1 9 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 17 20<br>19 9 20 7 | NO RELATIONS<br>MULTIPLE RELATIONS<br>GREAT GREAT GRAND FATHER<br>GREAT GRAND MOTHER |
| 88<br>5 27 22 51 45 53 88 57 24 31 17 18 61 22 16 38 67 96 68 49 80 36 95 75 35 72 2 32 28 86 13 55 14 99 87 74 30 6 15 42 78 93 54 82 60 46 1 12 10 65 21 41 97 26 11 79 50 91 48 52 29 64 39 19 25 92 90 73 70 94 7 81 37 4 77 71 33 89 40 8 85 34 43 3 69 63 23 59<br>50 80 59 22 | GREAT GREAT GREAT GRAND MOTHER<br>GREAT GREAT GRAND MOTHER<br>GREAT GREAT GREAT GREAT GRAND FATHER<br>MULTIPLE RELATIONS |
| 50<br>52 82 8 83 43 96 46 48 74 15 81 72 30 57 21 5 46 54 13 82 17 24 82 81 14 23 24 29 44 33 83 48 81 80 72 58 25 38 28 84 3 41 39 5 99 7 58 28 94 43<br>52 82 88 83 | MOTHER<br>MULTIPLE RELATIONS<br>NO RELATIONS |

| | |
|---|---|
| | MULTIPLE RELATIONS |
| 50<br>52 82 8 83 43 96 46 48 74 15 81 72 30 57 21 5 46 54 13 82 17 24 82 81 14 23 24 29 44 33 83 48 81 80 72 58 25 38 28 84 3 41 39 5 99 7 58 28 94 43<br>96 74 81 15 | GRAND FATHER<br>GREAT GRAND MOTHER<br>MULTIPLE RELATIONS<br>GREAT GRAND FATHER |
| 50<br>52 82 8 83 43 96 46 48 74 15 81 72 30 57 21 5 46 54 13 82 17 24 82 81 14 23 24 29 44 33 83 48 81 80 72 58 25 38 28 84 3 41 39 5 99 7 58 28 94 43<br>7 99 48 488 | GREAT GREAT GREAT GRAND FATHER<br>GREAT GREAT GREAT GRAND MOTHER<br>MULTIPLE RELATIONS<br>NO RELATIONS |

## Timely Tasks (p2v3d1)

**Timely Tasks [20 marks]**

----------------------------------------------------------------------

**Problem Statement**
Mr C has a lot of tasks to perform (your compilation requests, evaluation requests etc) and each of these tasks take a different amount of time. To avoid taking too much time on a single tasks, and neglecting other tasks, for example if one task runs into an infinite loop, all modern machines (including Prutor) follow a certain set of rules while executing these tasks. He does not execute a task till completion, but only runs it for a certain amount of time, called the *burst time*, say B seconds.

If the task completes within this period, nice. Otherwise, the task is kept aside and the next task waiting in line is picked up and given B seconds of time. Then the third task in line is given B seconds and so on. Tasks that complete before the allotted B second burst time is over, are removed from the waiting line. Once all tasks have been given a chance at B second bursts, the first task that is still in waiting is given a chance at another B seconds and the process continues till all tasks are completed. The time at which a task is completed is called its *completion time*.

The input will first give you the number of tasks n and the burst time value B, separated by a single space. Then it will give you a list of n strictly positive integers which will denote the time it takes for the n tasks to finish also called their *execution time*.

Suppose there are 5 tasks at hand which require various number of seconds to execute, say [5 2 3 6 1] and suppose B = 3 seconds. Then in the first round, each task is given a chance at 3 seconds each

1. Task 1 takes up all the 3 seconds but still needs 2 more seconds so it is kept aside (t = 3 now)
2. Task 2 finishes in 2 seconds itself and is removed from the waiting list (t = 5 now)
3. Task 3 takes up all the 3 seconds and finishes and is removed from the waiting list (t = 8 now)
4. Task 4 takes up all the 3 seconds but still needs 3 more seconds so it is kept aside (t = 11 now)
5. Task 5 finishes in 1 second itself and is removed from the waiting list (t = 12 now)

So the first round finishes after 11 seconds with the following as the state of the tasks (* indicates completed task) [2 * * 3 *]. Now the second round begins.

1. Task 1 finishes in 2 seconds itself and is removed from the waiting list (t = 14 now)
2. Task 4 takes up all the 3 seconds and finishes and is removed from the waiting list (t = 17 now)

Now all tasks have finished so Mr C can rest. Your have to print as your output, on a different line, when did each task finish.

**Caution**

1. Burst time and execution times will always be strictly positive integers.
2. Be careful about extra/missing lines and extra/missing spaces in your output.

----------------------------------------------------------------------

**EXAMPLE**:
INPUT
```
5 3
5 2 3 6 1
```

OUTPUT:
```
14
5
8
17
12
```

**Explanation**: As per the calculations above, task 1 finished at 14 seconds, task 2 finished at 5 seconds etc.
----------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 5 3<br>5 2 3 6 1 | 14<br>5<br>8<br>17<br>12 |
| 2 1<br>1 2 | 1<br>3 |

| | |
|---|---|
| 1 3<br>5 | 5 |
| 5 3<br>12 3 6 18 24 | 39<br>6<br>21<br>54<br>63 |
| 16 7<br>5 22 10 54 21 13 5 1 6 7 8 9 54 22 1 11 | 5<br>182<br>105<br>244<br>167<br>125<br>45<br>46<br>52<br>59<br>126<br>128<br>249<br>197<br>88<br>146 |
| 30 14<br>2 1 3 566 43 78 222 4567 66 1 2 4 5 6 19 444 5 22 10 54 21 13 5 1 6 9 54 22 1 11 | 2<br>3<br>6<br>2256<br>562<br>758<br>1330<br>6263<br>722<br>91<br>93<br>97<br>102<br>108<br>356<br>2026<br>141<br>378<br>165<br>644<br>399<br>206<br>211<br>212<br>218<br>227<br>656<br>421<br>256<br>267 |

## Plenty of Palindromes (p3v1d1)

**Plenty of Palindromes [20 marks]**

----------------------------------------------------------------------

**Problem Statement**
In the input, you will be given two strictly positive integers n and k, separated by a space. You have to generate all possible strings that satisfy the following properties

1. The strings should all be of length k
2. The strings should only contain the first n lower-case English alphabet characters. However, the allowed characters may repeat any number of times (some characters may be absent as well).
3. The strings should all be palindromes

Print each palindrome string on a different line of your output. You have to print the strings in *lexicographically* increasing order, i.e. the order in which these strings would appear in a dictionary. We explain lexicographic ordering below. Thus, you have to first output the lexicographically smallest palindrome string of length k you can construct out of the first n lower-case letters of the English alphabet. Then find the lexicographically next smallest palindrome string of length k and so on till you have generated all such palindromes possible.

**Lexicographic ordering**
Just as given two digit sequences, say 1923 and 3122, we can say which digit sequence is "smaller" and which is "larger" (by interpreting the digit sequences as numbers), two sequences of alphabets, i.e. strings, can also be compared and given two sequences of alphabets, we can say which one is smaller and which one is larger.

The rules of doing so are pretty simple if the two strings are of the same length, which will be the case in this question. We first declare that the character 'a' is smaller than the character 'b', which is in turn smaller than the character 'c' and so on. To compare two strings, simply look at their first characters - the string with the larger character wins. If both strings have the same first character, then the second characters of the two strings are compared and so on.

Thus, we have "cat" = "cat" since the two strings are exactly the same but we have "cap" < "cat" since 'p' is smaller than 't'. Also, we have "mat" > "cat" since 'm' is larger than 'c' and also "aazd" < "abbb" since 'a' is smaller than 'b' (The third characters 'z' and 'b' do not get compared at all since "abbb" wins when the second characters get compared).

**Caution**

1. Using recursion is not compulsory in this question. However, you will have to write much less code if you use recursion.
2. Do not output the strings in incorrect order. The autograder will heavily penalize you if you do this since it will give you marks only if a correct string is output in its correct location.
3. We will not penalize you for stray newlines at the end of your output. However, do not have stray spaces at the end of each line of your output. You will not pass test cases if you have these.

**HINTS**:
This problem may seem very complicated if you try to write a loop to solve the problem but it can be very elegantly solved using recursion as we explain below. Suppose we want to generate all palindromes of length 4 out of the first two alphabet letters, in lexicographic order. This can be solved if we first generate all palindromes starting with a and then generate all palindromes starting with b (since strings starting with a come before those starting with b in lexicographic order).

However, if a palindrome starts with an a, it must also end with an a. This means that all such palindromes must be of the form a**a where ** is a palindrome of length 2. Thus, we can solve this problem using a smaller version of the original problem which is why recursion can be used (to generate all palindromes of length 4, we need all palindromes of length 2).

Write a function that takes partially filled in palindromes and recursively calls itself to complete the palindromes. Write a function generatePalindromes(char* str, int k, int n, int next) which takes in four arguments

1. a character array name of length k+1 (k characters for the names and one for the NULL character)
2. the value of k (will help you know what is the length of the array)
3. the value of n (which all letters of the English alphabet can you use)
4. the first unfilled position in the array (denoted by * in the above example)

The base case of the recursion can be when only one or two positions are left unfilled in the string (depending on whether k is odd or even). In this base case, we simply need to loop through all valid characters and use them to fill in the empty positions. To start off, you can invoke this recursion using something like the following

```
char str[k+1];
str[k] = '\0';
generatePalindromes(str, k, n, 0);
```

where next = 0 indicates that the array is completely empty at this point.

Use these hints to completely solve the problem

----------------------------------------------------------------------

**EXAMPLE 1**:
INPUT
```
2 4
```

OUTPUT:
```
aaaa
abba
baab
bbbb
```

**EXAMPLE 2**:
INPUT
```
2 5
```

OUTPUT:
```
aaaaa
aabaa
ababa
abbba
baaab
babab
bbabb
bbbbb
```
----------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 2 4 | aaaa<br>abba<br>baab<br>bbbb |
| 5 2 | aa<br>bb<br>cc<br>dd<br>ee |
| 3 8 | aaaaaaaa<br>aaabbaaa<br>aaaccaaa<br>aabaabaa<br>aabbbbaa<br>aabccbaa<br>aacaacaa<br>aacbbcaa<br>aaccccaa<br>abaaaaba<br>ababbaba<br>abaccaba<br>abbaabba<br>abbbbbba<br>abbccbba<br>abcaacba<br>abcbbcba<br>abccccba<br>acaaaaca<br>acabbaca<br>acaccaca<br>acbaabca<br>acbbbbca<br>acbccbca<br>accaacca<br>accbbcca<br>acccccca<br>baaaaaab<br>baabbaab<br>baaccaab<br>babaabab<br>babbbbab<br>babccbab<br>bacaacab<br>bacbbcab<br>baccccab<br>bbaaaabb |

| | |
|---|---|
| | bbabbabb |
| | bbaccabb |
| | bbbaabbb |
| | bbbbbbbb |
| | bbbccbbb |
| | bbcaacbb |
| | bbcbbcbb |
| | bbccccbb |
| | bcaaaacb |
| | bcabbacb |
| | bcaccacb |
| | bcbaabcb |
| | bcbbbbcb |
| | bcbccbcb |
| | bccaaccb |
| | bccbbccb |
| | bccccccb |
| | caaaaaac |
| | caabbaac |
| | caaccaac |
| | cabaabac |
| | cabbbbac |
| | cabccbac |
| | cacaacac |
| | cacbbcac |
| | caccccac |
| | cbaaaabc |
| | cbabbabc |
| | cbaccabc |
| | cbbaabbc |
| | cbbbbbbc |
| | cbbccbbc |
| | cbcaacbc |
| | cbcbbcbc |
| | cbccccbc |
| | ccaaaacc |
| | ccabbacc |
| | ccaccacc |
| | ccbaabcc |
| | ccbbbbcc |
| | ccbccbcc |
| | cccaaccc |
| | cccbbccc |
| | cccccccc |
| 7 1 | a |
| | b |
| | c |
| | d |
| | e |
| | f |
| | g |
| 1 7 | aaaaaaa |
| 2 9 | aaaaaaaaa |
| | aaaabaaaa |
| | aaababaaa |
| | aaabbbaaa |
| | aabaaabaa |

| | |
|---|---|
| | aabababaa<br>aabbabbaa<br>aabbbbbaa<br>abaaaaaba<br>abaabaaba<br>ababababa<br>ababbbaba<br>abbaaabba<br>abbababba<br>abbbabbba<br>abbbbbbba<br>baaaaaaab<br>baaabaaab<br>baababaab<br>baabbbaab<br>babaaabab<br>babababab<br>babbabbab<br>babbbbbab<br>bbaaaaabb<br>bbaabaabb<br>bbabababb<br>bbabbbabb<br>bbbaaabbb<br>bbbababbb<br>bbbbabbbb<br>bbbbbbbbb |

## Count and Say Sequence (p3v2d1)

### Count and Say Sequence [20 marks]

---------------------------------------------------------------------

**Problem Statement**
Sometimes Mr C likes to play games to relieve himself from all the hard work he has to do compiling and running your programs. One of his favorite games is *Speak and Say* and is described below.

1. The game starts with a single digit number, for example 4
2. The first line of the game is simply the starting digit itself i.e. the string "4" (without quotes)
3. The second line of the game is obtained by speaking out the first line and then writing it in numbers. Since there is just "one four" in the first line, the second line is "14" (without quotes).
4. The third line is obtained by speaking and writing the second line. Since there is "one one followed by one four" in the second line, the third line is "1114" (without quotes).
5. The fourth line is "3114" (without quotes) since there are "three ones followed by one four" in the third line
6. The game continues for several rounds like this.

The input will give you two strictly positive numbers, n and k, separated by a space. The number n will be a single digit. You have to print k lines of the game starting with the digit n.

**Caution**

1. We assure you that all lines of the game will require 99 or less characters to print.
2. We assure you that in none of the lines, will any digit occur more than 9 times consecutively
3. This question will not necessarily benefit from recursive use of functions. However, you should use functions to write neat and easy-to--debug code.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

------------------------------------------------------------------
**EXAMPLE**:
INPUT
1 6

OUTPUT:
1
11
21
1211
111221
312211
------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 1 6 | 1<br>11<br>21<br>1211<br>111221<br>312211 |
| 4 8 | 4<br>14<br>1114<br>3114<br>132114<br>1113122114<br>311311222114<br>13211321322114 |
| 5 10 | 5<br>15<br>1115<br>3115<br>132115<br>1113122115 |

| | |
|---|---|
| | 311311222115 |
| | 13211321322115 |
| | 11131221131211132221155 |
| | 3113112221131112311332215 |
| 1 12 | 1 |
| | 11 |
| | 21 |
| | 1211 |
| | 111221 |
| | 312211 |
| | 13112221 |
| | 1113213211 |
| | 31131211131221 |
| | 13211311123113112211 |
| | 11131221133112132113212221 |
| | 3113112221232112111312211312113211 |
| 2 14 | 2 |
| | 12 |
| | 1112 |
| | 3112 |
| | 132112 |
| | 1113122112 |
| | 311311222112 |
| | 13211321322112 |
| | 1113122113121113222112 |
| | 31131122211311123113322112 |
| | 132113213221133112132123222112 |
| | 11131221131211132221232112111312111213322112 |
| | 31131122211311123113321112131221123113111231121123222112 |
| | 132113213221133112132123123112111311222112132113311213211221121332211 2 |
| 9 11 | 9 |
| | 19 |
| | 1119 |
| | 3119 |
| | 132119 |
| | 1113122119 |
| | 311311222119 |
| | 13211321322119 |
| | 1113122113121113222119 |
| | 31131122211311123113322119 |
| | 132113213221133112132123222119 |

# Orbiting Indices (p3v3d1)

**Orbiting Indices [20 marks]**

----------------------------------------------------------------------

**Problem Statement**
The first line of the input will give you a strictly positive number n. The next line of the input will give you a list of n non-negative integers, separated by a space. Store these in an array, say arr. We will play a game with this array. Say our starting index is j. Then in the next step, we will walk arr[j] + 1 steps to the right of the array. If we reach the end of the array while walking to the right, we simply wrap around and start walking from index 0. This is the first round of the game.

Say the index we land after following the above procedure is k. Then we repeat the above process again for another round, this time moving arr[k] + 1 steps to the right and wrapping around if we ever reach the end of the array. If following this process again and again, if we ever reach the original, starting index j, then index j is called an *orbiting index* since we keep coming back to that index again and again when playing this game.

In your output, print, for each of the n indices of the array, if that index is an orbiting index or not. If the index is not an orbiting index, print -1 else print the number of minimum number of rounds required to reach that index from the index itself.

**Caution**

1. This problem will not necessarily benefit from recursive formulations but you should write your program using functions to make it neat and easy-to-debug.
2. The integers in the list will be non-negative but can be zero.
3. Be careful about spelling and capitalization errors.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

---------------------------------------------------------------------

**EXAMPLE 1**:
INPUT
4
1 1 1 1

OUTPUT:
2
2
2
2

**Explanation**: All the indices are orbiting indices. The orbits for the different indices are given below. All orbits are of length 2.

1. 0 => 2 => 0
2. 1 => 3 => 1
3. 2 => 0 => 2
4. 3 => 1 => 3

**EXAMPLE 2**:
INPUT
4
0 0 0 2

OUTPUT:
-1
-1
2
2

**Explanation**: The first two indices are not orbiting indices - starting from them causes us to get stuck inside a (3 => 2 => 3) loop and we are never able to reach the original indices themselves. The orbits for last two indices are given below. Both orbits are of length 2.

1. 2 => 3 => 2
2. 3 => 2 => 3

---------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 4<br>1 1 1 1 | 2<br>2<br>2<br>2 |
| 4<br>0 0 0 2 | -1<br>-1<br>2<br>2 |
| 6<br>1 2 3 4 2 3 | 2<br>2<br>2<br>-1<br>2<br>-1 |
| 10<br>1 0 1 0 1 0 1 0 1 0 | 5<br>-1<br>5<br>-1<br>5<br>-1<br>5<br>-1<br>5<br>-1 |
| 10<br>1 0 1 0 1 0 2 3 4 1 | -1<br>5<br>5<br>-1<br>5<br>-1<br>5<br>-1<br>-1<br>5 |

| | |
|---|---|
| 11<br>0 0 0 8 0 0 0 0 6 0 10 | -1<br>3<br>3<br>3<br>5<br>5<br>5<br>5<br>5<br>-1<br>1 |

# Zig-zag Numbers (p4v1d1)

**Zig-zag Numbers [20 marks]**

----------------------------------------------------------------------

**Problem Statement**
A number is called a zig-zag number if the second digit of the number is strictly greater than the first, the third digit of the number is strictly smaller than the second, the fourth digit is strictly smaller than the third, and so on. Some examples include 121212, 1425 etc.

In the input, you will be given two strictly positive integers n and k, separated by a space. You will have to generate all possible zig-zag numbers that have exactly k digits in them, using only the digits 1,2,..,n. You must print all the possible zig-zag numbers in increasing order. A digit may repeat any number of times in a zig-zag number but the zig-zag rule must be followed.

We assure you that n will be strictly greater than 1 but strictly less than 10 (i.e. it will be a single digit number but not 1 or 0). To make this question a bit easier for you, we also assure you that k will be always an even number.

**Caution**

1. Using recursion is not compulsory in this question. However, you will have to write much less code if you use recursion.
2. Do not output the numbers in incorrect order. The autograder will heavily penalize you if you do this since it will give you marks only if a correct number is output at its correct location.
3. We will not penalize you for stray newlines at the end of your output. However, do not have stray spaces at the end of each line of your output. You will not pass test cases if you have these.

**HINTS**:
This problem may seem very complicated if you try to write a loop to solve the problem but it can be very elegantly solved using recursion as we explain below. It is easy to see that there can be no zig-zag numbers using a single digit. Let us try to construct all zig-zag numbers with two digits. It is again easy to see, that if n = 2, then there is only one zig-zag number with 2 digits i.e. 12. If n = 3, then we have more two-digit zig-zag numbers, namely 12, 13 and 23. If n = 4, then we have the possible two-digit zig-zag numbers as 12, 13, 14, 23, 24.

This forms the simplest case of a recursive definition of zig-zag numbers. Suppose we want to generate all zig-zag numbers with 8 digits starting with 15, then notice that this problem can be solved simply by appending to 15, all possible zig-zag numbers with 6 digits. However, there is a tiny restriction here that these 6 digit zig-zag numbers must start with a digit that is strictly smaller than 5.

Write a function that takes partially filled in zig-zag numbers and recursively calls itself to complete the numbers. Write a function generateZigZag(char* num, int k, int n, int left, int max) which takes in five arguments

1. a character array name of length k+1 (k characters for the number and one for the NULL character)
2. the value of k (will help you know what is the length of the array)
3. the value of n (which all digits can be used in the number)
4. left: how many digits are left to be filled in?
5. max: what is the maximum value that the first digit in the remaining number can take

The base case of the recursion can be left = 0 which means we have a complete number which can simply be printed. To print all zig-zag numbers using digits 1,2, ..., n starting with the "15", you could do something like the following:

```
char num[k+1];
num[k] = '\0'; // Do not forget the NULL character
num[0] = '1';
num[1] = '5';
generateZigZag(num, k, n, k - 2, 4) // two digits already filled, k-2 left to be filled, next digit must be strictly
smaller than 5
```

Use these hints to completely solve the problem.
-------------------------------------------------------------------
**EXAMPLE 1**:
INPUT
2 4

OUTPUT:
1212

**Explanation**: 1212 is the only four-digit zig-zag number possible using the digits 1,2.

**EXAMPLE 2**:
INPUT
3 4

OUTPUT:
```
1212
1213
1312
1313
1323
2312
2313
2323
```

**Explanation**: Notice that all the numbers are placed in increasing order.
-------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 2 4 | 1212 |
| 3 4 | 1212<br>1213<br>1312<br>1313<br>1323<br>2312<br>2313<br>2323 |
| 5 2 | 12<br>13<br>14<br>15<br>23<br>24<br>25<br>34<br>35<br>45 |
| 3 6 | 121212<br>121213<br>121312<br>121313<br>121323<br>131212<br>131213<br>131312<br>131313<br>131323<br>132312<br>132313<br>132323<br>231212<br>231213<br>231312<br>231313<br>231323<br>232312<br>232313<br>232323 |
| 4 4 | 1212<br>1213<br>1214<br>1312<br>1313<br>1314 |

| | |
|---|---|
| | 1323<br>1324<br>1412<br>1413<br>1414<br>1423<br>1424<br>1434<br>2312<br>2313<br>2314<br>2323<br>2324<br>2412<br>2413<br>2414<br>2423<br>2424<br>2434<br>3412<br>3413<br>3414<br>3423<br>3424<br>3434 |
| 4 6 | 121212<br>121213<br>121214<br>121312<br>121313<br>121314<br>121323<br>121324<br>121412<br>121413<br>121414<br>121423<br>121424<br>121434<br>131212<br>131213<br>131214<br>131312<br>131313<br>131314<br>131323<br>131324<br>131412<br>131413<br>131414<br>131423<br>131424<br>131434<br>132312<br>132313<br>132314<br>132323 |

```
132324
132412
132413
132414
132423
132424
132434
141212
141213
141214
141312
141313
141314
141323
141324
141412
141413
141414
141423
141424
141434
142312
142313
142314
142323
142324
142412
142413
142414
142423
142424
142434
143412
143413
143414
143423
143424
143434
231212
231213
231214
231312
231313
231314
231323
231324
231412
231413
231414
231423
231424
231434
232312
232313
232314
232323
232324
232412
```

```
232413
232414
232423
232424
232434
241212
241213
241214
241312
241313
241314
241323
241324
241412
241413
241414
241423
241424
241434
242312
242313
242314
242323
242324
242412
242413
242414
242423
242424
242434
243412
243413
243414
243423
243424
243434
341212
341213
341214
341312
341313
341314
341323
341324
341412
341413
341414
341423
341424
341434
342312
342313
342314
342323
342324
342412
342413
342414
```

| | |
|---|---|
| | 342423<br>342424<br>342434<br>343412<br>343413<br>343414<br>343423<br>343424<br>343434 |

# Parent Palindrome (p4v2d1)

**Parent Palindrome [20 marks]**

------------------------------------------------------------------

**Problem Statement**
In the first line of the input, we will give you a string, lets call it str, with at most 99 characters. str will only contain lower-case English alphabet letters. You have to find the shortest palindrome string, lets call it ptr, such that str is a substring of ptr. In the first line of the output, you have to print the length of ptr and in the second line of the output, you have to print the string ptr itself.

If str is itself a palindrome, print the length of str in the first line and str itself in the second line. If there are multiple strings of the same (shortest) length that are palindromes which contain str as a substring, print the string that is *lexicographically smallest*. We explain lexicographic ordering below.

**Lexicographic ordering**
Just as given two digit sequences, say 1923 and 3122, we can say which digit sequence is "smaller" and which is "larger" (by interpreting the digit sequences as numbers), two sequences of alphabets, i.e. strings, can also be compared and given two sequences of alphabets, we can say which one is smaller and which one is larger.

The rules of doing so are pretty simple if the two strings are of the same length, which will be the case in this question. We first declare that the character 'a' is smaller than the character 'b', which is in turn smaller than the character 'c' and so on. To compare two strings, simply look at their first characters - the string with the larger character wins. If both strings have the same first character, then the second characters of the two strings are compared and so on.

Thus, we have "cat" = "cat" since the two strings are exactly the same but we have "cap" < "cat" since 'p' is smaller than 't'. Also, we have "mat" > "cat" since 'm' is larger than 'c' and also "aazd" < "abbb" since 'a' is smaller than 'b' (The third characters 'z' and 'b' do not get compared at all since "abbb" wins when the second characters get compared).

**Caution**

1. There is no specific need to use recursion to solve this problem. However, you may want to write a modular code with functions to make your solution easier to read and easier to debug.
2. Be careful about extra/missing lines and extra/missing spaces in your output.

------------------------------------------------------------------

**EXAMPLE 1**:
INPUT

edens

OUTPUT:
7
snedens

**Explanation**: the string is itself not a palindrome. However, adding two appropriate letters to the left of the string makes it a palindrome.

**EXAMPLE 2**:
INPUT
malayalam

OUTPUT:
9
malayalam

**Explanation**: the string is itself a palindrome.
----------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| edens | 7<br>snedens |
| malayalam | 9<br>malayalam |
| aoob | 7<br>aoobooa |
| abcde | 9<br>abcdedcba |
| iwouldlovetolearnmalayalam | 43<br>iwouldlovetolearnmalayalamnraelotevoldluowi |
| momars | 9<br>sramomars |

# Leaderboard (p4v3d1)

**Leaderboard [20 marks]**

----------------------------------------------------------------------

**Problem Statement**
Mr C is participating in a contest where one can play any number of times. Every time Mr C plays, he gets a non-negative score. Based on his score and scores that other players have received, he is assigned a rank. However, the rank is calculated in a very careful manner. The highest score is awarded rank 1, the next highest score is awarded rank 2, and so on. However, if two scores are the same, then they are awarded the same rank.

Consider the following example. Suppose the scores of the players are [100 100 50 40 40 20 10]. Then the first two 100 scores both get rank 1. However, the score 50 gets rank 2. The next two 40 scores get rank 3. The score 20 gets rank 4 and the score 10 gets rank 5.

In the first line of the input you will be given a strictly positive integer n, separated by a space. n tells you how many player scores are already there. In the next line, there will be n non-negative player scores give to you in non-increasing order (i.e. from left to right, the scores will stay the same or go down, never go up), two scores separated by a space. Store these scores in an array.

In the next line you will be given another strictly positive integer m, which tells you the number of attempts Mr C makes in the game. In the next line, you will be given m non-negative scores that Mr C got in each of his attempts, two scores separated by a space. Read each score, store it in the correct position in the array and find the rank of Mr C's score and print it on a separate line of the output.

**Caution**

1. You may want to create an array of size m+n since you will have to store those many elements eventually (n original scores of other players and m scores of Mr C).
2. Mr C's rank for his second score should be calculated after his first score has been inserted into the array at the correct location. Mr C's rank for his third score should be calculated after his first two scores have been inserted into the array at the correct location.
3. Be careful about extra/missing lines and extra/missing spaces in your output.

----------------------------------------------------------------------

**EXAMPLE**:
INPUT
```
7
100 100 50 40 40 20 10
4
5 25 50 120
```

OUTPUT:
```
6
4
2
1
```

**Explanation**

1. Initially the score list is [100 100 50 40 40 20 10].
2. Inserting 5 into the list gives us [100 100 50 40 40 40 20 10 5] and its rank is 6
3. Inserting 25 into the list gives us [100 100 50 40 40 40 25 20 10 5] and its rank is 4
4. Inserting 50 into the list gives us [100 100 50 50 40 40 40 25 20 10 5] and its rank is 2
5. Inserting 120 into the list gives us [120 100 100 50 50 40 40 40 25 20 10 5] and its rank is 1

----------------------------------------------------------------------

**Grading Scheme**:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

## All Test Cases (Visible + Hidden)

| Input | Output |
|---|---|
| 7 100 100 50 40 40 20 10 4 5 25 50 120 | 6 4 2 1 |
| 10 200 180 150 130 120 100 70 60 50 40 6 40 70 100 130 170 190 | 10 7 6 4 3 2 |
| 15 5000 1300 1299 1100 1000 1000 1000 1000 900 900 800 754 700 400 300 8 500 755 800 900 1000 1000 4000 5000 | 10 8 7 6 5 5 2 1 |
| 200 997 988 981 966 957 937 933 930 929 928 927 926 922 920 916 915 903 896 887 874 872 866 863 863 860 859 858 857 857 847 847 842 830 819 815 809 803 797 796 794 794 789 785 783 778 772 765 765 764 757 755 751 744 740 737 733 730 730 724 716 710 709 691 690 684 677 676 653 652 650 625 620 619 602 587 587 585 583 571 568 568 556 552 546 541 540 538 531 530 529 527 506 504 501 498 493 493 492 489 482 475 468 457 452 445 442 441 438 435 435 433 430 429 427 422 422 414 408 404 400 396 394 387 384 380 379 374 371 369 369 369 368 366 365 363 354 351 341 337 336 328 325 318 316 314 307 306 302 287 282 281 277 276 271 246 238 236 230 229 229 228 227 220 212 199 194 179 173 171 168 150 144 136 125 125 124 122 118 98 98 95 92 88 85 70 68 61 60 59 44 43 35 32 30 28 23 20 13 12 12 50 83 129 140 184 198 300 312 325 341 344 349 356 370 405 423 444 465 471 491 500 506 508 539 543 569 591 607 612 614 623 645 670 689 726 744 747 764 773 777 787 805 811 819 829 841 905 918 918 955 997 | 169 160 159 153 152 136 133 129 125 125 125 123 118 108 105 97 94 93 90 87 84 |

| | |
|---|---|
| | 84<br>79<br>77<br>73<br>69<br>68<br>68<br>68<br>66<br>65<br>62<br>59<br>53<br>48<br>48<br>44<br>42<br>42<br>39<br>34<br>33<br>31<br>31<br>30<br>17<br>15<br>15<br>6<br>1 |
| 7<br>100 100 50 40 40 20 10<br>5<br>5 25 50 100 120 | 6<br>4<br>2<br>1<br>1 |
| 10<br>200 180 150 130 120 100 70 60 50 40<br>9<br>40 70 100 130 170 190 200 200 200 | 10<br>7<br>6<br>4<br>3<br>2<br>1<br>1<br>1 |