In this article, let’s explore how to generate Lyndon words of length n. An integer is given n, with some characters in an array arr. The aim is to create **Lyndon words** of length n with characters provided through a vector. A Lyndon word is a non-empty string that is strictly smaller than all of its rotations in terms of lexicographic order. It is a nonempty word that is strictly smaller in lexicographic order than any of its nontrivial rotations.

**Let’s understand it with a few examples:**

**Example 1:**

Suppose the given characters are ‘0’,’1’,’2’, and we have to generate Lyndon words of length 2. So, Lyndon's words are ‘01’,’02’,’12’. These all are Lyndon words as they are less than its rotation in lexicographic order ‘10’,’20’,’21’.

But if we take ‘10’, it is greater than its rotation in lexicographic order ‘01’. So, ‘10’ is not a Lyndon word.

**Example 2:**

Suppose the given characters are ‘0’,’1’,’2’, and we have to generate Lyndon words of length 3. So, Lyndon's words are ‘001’,’002’,’011’,’012’,’021’,’022’,’112’,’122’. These all are Lyndon words as they are less than its rotations lexicographic order. To verify, let’s take an example of ‘012’ then. It is less than all its rotations in the lexicographic order ‘120’,’201’.

But if we take ‘102’, it is greater than its rotation in lexicographic order ‘021’. So, ‘102’ is not a Lyndon word.

Suppose the given characters are ‘a’,’b’,’c’, and we have to generate Lyndon words of length 3. So, Lyndon's words are ‘aab’,’aac’,’abb’,’abc’,’acb’,’acc’,’bbc’,’bcc’. These all are Lyndon words as they comes first in its rotations lexicographic order. To verify, let’s take an example of ‘abc’ then. It comes first from all its rotations in the lexicographic order ‘bca’,’cab’.

But if we take ‘bac’, then its lexicographical order ‘acb’ comes before ‘bac’. So, ‘bac’ is not a lyndon word.

**As "000" equals the string created by spinning it, it is not considered a Lyndon word.**

So, all the strings like aaa, 111, 222 are not considered as the lyndon words.

## **Approach 1: Without Using Recursion**

In this approach, we start with the shortest Lyndon word of the length ‘n’ with given characters supplied in the form of a vector. Then, generate the lyndon words and print to the console.

### **Algorithm**

**Step 1:** Create a lyndon\_words function, which accepts characters which are used in generating lyndon word and an integer n which defines the size of lyndon word.

**Step 2:** Sort the characters which are used in generating lyndon word in increasing order.

**Step 3:** Create a vector called result and set its initial value to -1.

**Step 4:** Start a loop until the result vector is empty.

**Step 5:** Inside the loop, check the words generated one by one whether it is lyndon word or not.

**Step 6:** Print the lyndon words on the console.

### **Program**

|  |
| --- |
| #include <iostream>  #include <algorithm>  #include <vector>  #include <string>  using namespace std;  void lyndon\_words(vector<char>& s, int n) {  sort(s.begin(), s.end());  vector<int> result = {-1};  int k = s.size();  while (!result.empty()) {  result.back()++;  int m = result.size();  if (m == n) {  string output;  for (int i : result) {  output += s[i];  }  cout << output << endl;  }  while (result.size() < n) {  result.push\_back(result[result.size() - m]);  }  while (!result.empty() && result.back() == k - 1) {  result.pop\_back();  }  }  }  int main() {  int n;  cout << "Enter the length of the word: ";  cin >> n;  vector<char> s = {'2', '1', '0'};  lyndon\_words(s, n);  return 0;  } |

**Output:**

|  |
| --- |
| Enter the length of the word: 2  01  02  12 |

## **Approach 2: Using Recursion**

In this approach we build a function which get recursively called to print the lyndon words on to the console with the given length using the characters provided.

### **Algorithm**

**Step 1:** Create a recursive helper function.

**Step 2:** This function check if the output is a lyndon word or not.

**Step 3:** It will print the word if it is a lyndon word.

**Step 4:** This recursive helper function get recursively called to print all the lyndon words of given length.

### **Program**

|  |
| --- |
| #include <iostream>  #include <algorithm>  #include <vector>  #include <string>  using namespace std;  void lyndon\_words\_helper(const vector<char>& s, const int n, string& current, const int i) {  if (current.length() == n) {  // check if the output is a sequence of the same number  bool isSequence = true;  for (int j = 0; j < current.length() - 1; j++) {  if (current[j] != current[j + 1]) {  isSequence = false;  break;  }  }  // only output if it's not a sequence  if (!isSequence) {  cout << current << endl;  }  return;  }  for (int j = i; j < s.size(); j++) {  current += s[j];  lyndon\_words\_helper(s, n, current, j);  current.pop\_back();  }  }  void lyndon\_words(vector<char>& s, int n) {  sort(s.begin(), s.end());  string current = "";  lyndon\_words\_helper(s, n, current, 0);  }  int main() {  int n;  cout << "Enter the length of the word: ";  cin >> n;  vector<char> s = {'2', '1', '0'};  lyndon\_words(s, n);  return 0;  } |

**Output:**

|  |
| --- |
| Enter the length of the word: 3  001  002  011  012  022  112  122 |

## **Conclusion**

In this article, we understand about Lyndon words. We know how to generate Lyndon words of length N. Then we learn to find the Lyndon words with algorithm and code implementation in c++.