Ex7.s6: Repeated subsequences

🔍 **Find all repeated substrings of length ≥ *m*** in a string *s* by identifying positions *i* and *j* where `s[i..i+m] == s[j..j+m]`

Ex7.s6: Repeated subsequences

🔍 **Find all repeated substrings of length ≥ *m*** in a string *s* by identifying positions *i* and *j* where `s[i..i+m] == s[j..j+m]`

💡For simplicity, let's say `s= "banana banana”; m=3`

1. **Generate all suffixes of string s:**

i	suffix
0	banana banana
1	anana banana
2	nana banana
3	ana banana
4	na banana
5	a banana
6	banana
7	banana
8	anana
9	nana
10	ana
11	na
12	a

Ex7.s6: Repeated subsequences

🔍 **Find all repeated substrings of length ≥ *m*** in a string *s* by identifying positions *i* and *j* where `s[i..i+m] == s[j..j+m]`

💡For simplicity, let's say `s= "banana banana”; m=3`

2. Sort all suffixes alphabetically:

i	suffix
6	banana
12	a
5	a banana
10	ana
3	ana banana
1	anana banana
8	anana
0	banana banana
7	banana
2	nana banana
9	nana
11	na
4	na banana

Ex7.s6: Repeated subsequences

🔍 **Find all repeated substrings of length ≥ *m*** in a string *s* by identifying positions *i* and *j* where `s[i..i+m] == s[j..j+m]`

💡For simplicity, let’s say `s= "banana banana”; m=3`

💡 now, **any repeated substring of length ≥ 3** must appear as a common prefix between **neighbouring suffixes** in this sorted list

2. **Sort all suffixes alphabetically:**

i	suffix
6	banana
12	a
5	a banana
10	ana
3	ana banana
1	anana banana
8	anana
0	banana banana
7	banana
2	nana banana
9	nana
11	na
4	na banana

Ex7.s6: Repeated subsequences

🔍 **Find all repeated substrings of length ≥ *m*** in a string *s* by identifying positions *i* and *j* where `s[i..i+m] == s[j..j+m]`

💡For simplicity, let's say `s= "banana banana”; m=3`

💡 now, **any repeated substring of length ≥ 3** must appear as a common prefix between **neighbouring suffixes** in this sorted list

➞ For each pair of neighbours, check whether they share more than 3 characters at the beginning

2. **Sort all suffixes alphabetically:**

i	suffix
6	banana
12	a
5	a banana
10	ana
3	ana banana
1	anana banana
8	anana
0	banana banana
7	banana
2	nana banana
9	nana
11	na
4	na banana

Ex7.s6: Repeated subsequences

🔍 **Find all repeated substrings of length ≥ *m*** in a string *s* by identifying positions *i* and *j* where `s[i..i+m] == s[j..j+m]`

💡For simplicity, let's say `s= "banana banana”; m=3`

💡 now, **any repeated substring of length ≥ 3** must appear as a common prefix between **neighbouring suffixes** in this sorted list

➞ For each pair of neighbours, check whether they share more than 3 characters at the beginning

2. Sort all suffixes alphabetically:

i	suffix	
6	banana	0
12	a	1
5	a banana	1
10	ana	3
3	ana banana	3
1	anana banana	5
8	anana	0
0	banana banana	6
7	banana	0
2	nana banana	4
9	nana	2
11	na	2
4	na banana	

Ex7.s6: Repeated subsequences

🔍 **Find all repeated substrings of length ≥ *m*** in a string *s* by identifying positions *i* and *j* where `s[i..i+m] == s[j..j+m]`

💡For simplicity, let's say `s= "banana banana"; m=3`

💡 now, **any repeated substring of length ≥ 3** must appear as a common prefix between **neighbouring suffixes** in this sorted list

➞ For each pair of neighbours, check whether they share more than 3 characters at the beginning

2. Sort all suffixes alphabetically:

i	suffix	
6	banana	0
12	a	1
5	a banana	1
10	ana	3 ✓
3	ana banana	✓ 3
1	anana banana	5 ✓
8	anana	0
0	banana banana	6 ✓
7	banana	0
2	nana banana	4 ✓
9	nana	2
11	na	2
4	na banana	

Ex7.s6: Repeated subsequences

🔍 **Find all repeated substrings of length ≥ *m*** in a string *s* by identifying positions *i* and *j* where `s[i..i+m] == s[j..j+m]`

Running time

- Sorting n suffixes: $O(n \log n)$ comparisons (Quicksort), where each comparison takes $O(m)$ (because we sort by the first m characters)

⇒ total: $O(n \log n) \cdot O(m)$