Pangram Checker: Complete Guide

Table of Contents

---

1. Problem Definition

Problem Statement: Given a sentence, determine if it is a pangram.
Input: A string (sentence)
Output: Boolean (true if pangram, false otherwise)
Constraints:

- Case-insensitive
- Ignore non-alphabetic characters
- English alphabet (26 letters)

---

2. What is a Pangram?

A **pangram** is a sentence that contains every letter of the alphabet at least once.

Examples:

- *"The quick brown fox jumps over the lazy dog"* ✓
- *"Pack my box with five dozen liquor jugs"* ✓
- *"Hello World"* ✗ (missing many letters)

---

3. Solution Approaches

3.1 Hashmap/HashSet Approach

Idea: Use a set to store unique letters, check if size = 26

Algorithm:

text

1. Convert sentence to lowercase

2. Create empty set

3. For each character in sentence:

   a. If character is a-z, add to set

4. Return true if set size = 26

3.2 Array of Frequencies Approach

Idea: Use boolean array of size 26 to track presence

Algorithm:

text

1. Create boolean array of size 26 (all false)

2. Convert sentence to lowercase

3. For each character in sentence:

   a. If character is a-z:

      index = character - 'a'

      array[index] = true

4. Check if all array elements are true

---

4. Dry Run Examples

Example Sentence: "The quick brown fox jumps over the lazy dog"

4.1 Hashmap Dry Run

text

Step 1: Convert to lowercase

"the quick brown fox jumps over the lazy dog"

Step 2: Process characters:

't' → add to set → set = {t}

'h' → add to set → set = {t, h}

'e' → add to set → set = {t, h, e}

' ' → skip

'q' → add to set → set = {t, h, e, q}

'u' → add to set → set = {t, h, e, q, u}

'i' → add to set → set = {t, h, e, q, u, i}

...

Continue until all letters processed

Step 3: Final set = {a, b, c, d, e, f, g, h, i, j, k, l, m,

n, o, p, q, r, s, t, u, v, w, x, y, z}

Size = 26 → Return true

4.2 Array Dry Run

text

Step 1: Create array[26] = [false, false, ..., false]

Step 2: Convert to lowercase

"the quick brown fox jumps over the lazy dog"

Step 3: Process characters:

't' (ASCII 116) → 't' - 'a' = 19 → array[19] = true

'h' (ASCII 104) → 'h' - 'a' = 7 → array[7] = true

'e' (ASCII 101) → 'e' - 'a' = 4 → array[4] = true

' ' → skip

'q' (ASCII 113) → 'q' - 'a' = 16 → array[16] = true

...

Continue until all letters processed

Step 4: Final array = [true, true, true, ..., true] (all 26 true)

Check all true → Return true

Visual Array State:

text

Index: 0 1 2 3 4 5 6 7 8 9 10 11 12

Letter: a b c d e f g h i j k l m

Value: T T T T T T T T T T T T T

Index: 13 14 15 16 17 18 19 20 21 22 23 24 25

Letter: n o p q r s t u v w x y z

Value: T T T T T T T T T T T T T

---

5. Code Implementations

5.1 Java

java

```
import java.util.HashSet;

public class PangramChecker {

  // Method 1: Using HashSet
  public static boolean isPangramHashSet(String sentence) {
    HashSet<Character> letters = new HashSet<>();
    sentence = sentence.toLowerCase();

    for (char ch : sentence.toCharArray()) {
      if (ch >= 'a' && ch <= 'z') {
        letters.add(ch);
      }
    }
    return letters.size() == 26;
  }

  // Method 2: Using Boolean Array
  public static boolean isPangramArray(String sentence) {
    boolean[] seen = new boolean[26];
    sentence = sentence.toLowerCase();

    for (char ch : sentence.toCharArray()) {
      if (ch >= 'a' && ch <= 'z') {
```

```
                seen[ch - 'a'] = true;
            }
        }

        for (boolean present : seen) {
            if (!present) return false;
        }
        return true;
    }

    // Alternative: Using Streams (Java 8+)
    public static boolean isPangramStream(String sentence) {
        return sentence.toLowerCase().chars()
                .filter(Character::isLetter)
                .distinct()
                .count() == 26;
    }

    public static void main(String[] args) {
        String test = "The quick brown fox jumps over the lazy dog";
        System.out.println("HashSet: " + isPangramHashSet(test));
        System.out.println("Array: " + isPangramArray(test));
        System.out.println("Stream: " + isPangramStream(test));
    }
}
```

5.2 C++

cpp

```
#include <iostream>
#include <unordered_set>
#include <string>
#include <cctype>
```

```cpp
#include <algorithm>
using namespace std;

// Method 1: Using unordered_set
bool isPangramHashSet(const string& sentence) {
    unordered_set<char> letters;

    for (char ch : sentence) {
        if (isalpha(ch)) {
            letters.insert(tolower(ch));
        }
    }
    return letters.size() == 26;
}

// Method 2: Using Boolean Array
bool isPangramArray(const string& sentence) {
    bool seen[26] = {false};

    for (char ch : sentence) {
        if (isalpha(ch)) {
            seen[tolower(ch) - 'a'] = true;
        }
    }

    // Check all letters present
    for (bool present : seen) {
        if (!present) return false;
    }
    return true;
}
```

```
// Method 3: Using bitset (memory efficient)
bool isPangramBitset(const string& sentence) {
    bitset<26> letters;

    for (char ch : sentence) {
        if (isalpha(ch)) {
            letters.set(tolower(ch) - 'a');
        }
    }
    return letters.all(); // Check if all bits are set
}

int main() {
    string test = "The quick brown fox jumps over the lazy dog";
    cout << "HashSet: " << (isPangramHashSet(test) ? "True" : "False") << endl;
    cout << "Array: " << (isPangramArray(test) ? "True" : "False") << endl;
    cout << "Bitset: " << (isPangramBitset(test) ? "True" : "False") << endl;
    return 0;
}
```

5.3 Python

```
python
# Method 1: Using Set
def is_pangram_set(sentence):
    letters = set()
    for ch in sentence.lower():
        if 'a' <= ch <= 'z':
            letters.add(ch)
    return len(letters) == 26

# Method 2: Using Boolean List/Array
```

```
def is_pangram_array(sentence):
    seen = [False] * 26
    for ch in sentence.lower():
        if 'a' <= ch <= 'z':
            seen[ord(ch) - ord('a')] = True
    return all(seen)

# Method 3: Using Pythonic Set Comparison
def is_pangram_pythonic(sentence):
    alphabet = set('abcdefghijklmnopqrstuvwxyz')
    return alphabet <= set(sentence.lower())

# Method 4: Using all() with ascii_lowercase
import string
def is_pangram_all(sentence):
    sentence_lower = sentence.lower()
    return all(ch in sentence_lower for ch in string.ascii_lowercase)

# Test
test_sentence = "The quick brown fox jumps over the lazy dog"
print(f"Set method: {is_pangram_set(test_sentence)}")
print(f"Array method: {is_pangram_array(test_sentence)}")
print(f"Pythonic method: {is_pangram_pythonic(test_sentence)}")
print(f"All method: {is_pangram_all(test_sentence)}")
```

---

6. Complexity Analysis

Time Complexity

Method	Time Complexity	Description
Hashmap/Set	O(n)	Iterate through n characters, O(1) insert

Method	Time Complexity	Description
Array	O(n)	Iterate through n characters, O(1) update
Bitset (C++)	O(n)	Same as array but with bit operations
Pythonic (set comparison)	O(n + 26)	O(n) to build set, O(26) for comparison

Note: n = length of sentence

Space Complexity

Method	Space Complexity	Description
Hashmap/Set	O(26) = O(1)	Store at most 26 letters
Array	O(26) = O(1)	Fixed size array
Bitset (C++)	O(1)	26 bits only

All are O(1) space

---

7. Test Cases

python

```
# Test Cases Table
test_cases = [
    ("The quick brown fox jumps over the lazy dog", True),  # Classic
    ("Pack my box with five dozen liquor jugs", True),      # Another classic
    ("abcdefghijklmnopqrstuvwxyz", True),                   # Alphabet
    ("ABCDEFGHIJKLMNOPQRSTUVWXYZ", True),                   # Uppercase
    ("Hello World", False),                                 # Missing letters
    ("", False),                                            # Empty
    ("1234567890!@#$%^&*()", False),                        # No letters
    ("The five boxing wizards jump quickly", True),         # Another pangram
    ("This sentence is definitely not a pangram", False),   # Missing letters
```

```
    ("Mr. Jock, TV quiz PhD, bags few lynx", True),     # Short pangram
]
```

Edge Cases to Consider:

1. Empty string
2. Very long string
3. String with only special characters
4. Mixed case letters
5. Unicode characters (should be ignored)
6. String with spaces, punctuation

---

8. Comparison of Approaches

Aspect	Hashmap/Set	Array	Bitset
Memory Usage	Higher (objects overhead)	Medium (26 booleans)	Lowest (26 bits)
Speed	Fast O(1) inserts	Very fast direct access	Fastest bit operations
Code Clarity	Very clear	Clear	Less clear
Flexibility	Can count frequencies	Only presence/absence	Only presence/absence
Best For	When also need frequencies	Simple presence check	Memory-constrained