**Part 1: String Basics (Definition & Moderate)**

**Q1. [Definition] What is the time complexity to access a character at a specific index in a Java String?**  
A. O(n)  
B. O(log n)  
C. O(1)  
D. O(n log n)  
**Answer: C**  
**Explanation:** Java String is backed by a character array, allowing O(1) access.

**Q2. [Definition] Which of the following is immutable in Java?**  
A. StringBuilder  
B. StringBuffer  
C. String  
D. char[]  
**Answer: C**  
**Explanation:** Java String objects are immutable, i.e., cannot be changed once created.

**Q3. [Definition] Which method is used to compare two strings ignoring case in Java?**  
A. equals()  
B. compareToIgnoreCase()  
C. equalsIgnoreCase()  
D. contains()  
**Answer: C**  
**Explanation:** equalsIgnoreCase() compares strings without considering case.

**Q4. [Moderate] What is the output of the following code?**

String s = "hello";

System.out.println(s.substring(1, 3));

A. "he"  
B. "ell"  
C. "el"  
D. "lo"  
**Answer: C**  
**Explanation:** substring(1, 3) returns characters from index 1 to 2 → "el".

**Q5. [Moderate] Which method should be used to reverse a string in Java efficiently?**  
A. String.reverse()  
B. StringBuffer.reverse()  
C. Collections.reverse()  
D. Arrays.reverse()  
**Answer: B**  
**Explanation:** StringBuffer and StringBuilder offer efficient reverse operations.

**Q6. [Moderate] What will be the result of "hello" + 5 + 2?**  
A. "hello7"  
B. "hello52"  
C. Compilation Error  
D. "hello"  
**Answer: B**  
**Explanation:** String concatenation happens left to right → "hello" + 5 → "hello5" → "hello5" + 2 → "hello52".

**Q7. [Moderate] Which of the following is best to use for frequent string modifications?**  
A. String  
B. StringBuilder  
C. StringBuffer  
D. char[]  
**Answer: B**  
**Explanation:** StringBuilder is non-thread-safe but faster for frequent string operations.

**🔹 Part 2: Hashing in Strings**

**Q8. [Definition] What is the use of a hash map in string problems?**  
A. Sorting  
B. Counting characters or substrings  
C. Reversing a string  
D. String formatting  
**Answer: B**  
**Explanation:** Hash maps efficiently store frequencies and mappings for string manipulation.

**Q9. [Moderate] What is the output of the following code?**

String s = "abcabc";

Map<Character, Integer> freq = new HashMap<>();

for (char c : s.toCharArray()) {

freq.put(c, freq.getOrDefault(c, 0) + 1);

}

System.out.println(freq.get('a'));

A. 3  
B. 2  
C. 1  
D. 0  
**Answer: B**  
**Explanation:** The letter 'a' occurs twice in the string "abcabc".

**Q10. [Definition] What is the time complexity of inserting a character in a HashMap in Java?**  
A. O(1) on average  
B. O(log n)  
C. O(n)  
D. O(n log n)  
**Answer: A**  
**Explanation:** HashMap provides average-case O(1) time for put/get operations.

**Q11. [Moderate] What would a HashSet be used for in string problems?**  
A. Counting characters  
B. Storing all characters uniquely  
C. Finding duplicates  
D. B and C  
**Answer: D**  
**Explanation:** HashSet helps in uniqueness and duplicate detection.

**Q12. [Difficult] Given two strings, determine if they are anagrams. Which data structure is best suited for this?**  
A. ArrayList  
B. Stack  
C. HashMap or frequency array  
D. TreeMap  
**Answer: C**  
**Explanation:** Anagrams are verified by comparing character frequencies using HashMap or arrays.

**🔹 Part 3: Frequency Array**

**Q13. [Definition] How many indices are needed to store the frequency of lowercase English letters using an array?**  
A. 52  
B. 128  
C. 26  
D. 256  
**Answer: C**  
**Explanation:** Lowercase letters range from 'a' to 'z' — 26 characters.

**Q14. [Moderate] What is the output of this code for string s = "banana"?**

java

CopyEdit

int[] freq = new int[26];

for (char c : s.toCharArray()) {

freq[c - 'a']++;

}

System.out.println(freq['a' - 'a']);

A. 2  
B. 3  
C. 1  
D. 0  
**Answer: B**  
**Explanation:** 'a' occurs 3 times in "banana".

**Q15. [Moderate] What is the benefit of using a frequency array over HashMap in some string problems?**  
A. Faster access due to fixed size  
B. More memory efficient  
C. Better for limited character sets  
D. All of the above  
**Answer: D**  
**Explanation:** Frequency arrays are constant-time, compact, and ideal for small fixed sets like alphabets.

**Q16. [Difficult] Two strings are given. How can you check if they are anagrams using frequency arrays?**  
A. Count and compare  
B. Sort and compare  
C. Recursively check  
D. Use regex  
**Answer: A**  
**Explanation:** Counting characters and comparing frequency arrays is optimal for anagram checking.

**Q17. [Moderate] What is the output of the code below?**

String s = "teststring";

int[] freq = new int[26];

for (char c : s.toCharArray()) {

freq[c - 'a']++;

}

System.out.println(freq['t' - 'a']);

A. 3  
B. 2  
C. 1  
D. 4  
**Answer: A**  
**Explanation:** 't' appears three times in "teststring".

**🔹 Part 4: Difficult / Scenario-Based**

**Q18. [Difficult] You are given a string and asked to return the index of the first non-repeating character. What’s the best approach?**  
A. Use two loops  
B. Use HashMap to store counts, then linear scan  
C. Sort and scan  
D. Use a stack  
**Answer: B**  
**Explanation:** HashMap gives frequency; then a single pass finds the first with count 1.

**Q19. [Difficult] Which problem is not efficiently solvable using frequency arrays?**  
A. Find first unique character  
B. Check anagram  
C. Substring pattern matching  
D. Count vowels  
**Answer: C**  
**Explanation:** Pattern matching often needs hashing or KMP, not just frequency.

**Q20. [Difficult] What’s the best structure to use when characters can be any Unicode symbol?**  
A. Frequency array of size 26  
B. Frequency array of size 256  
C. HashMap<Character, Integer>  
D. Bit array  
**Answer: C**  
**Explanation:** Unicode symbols vary widely; HashMap adapts to all key types.

**Q21. [Moderate] Which code correctly initializes a frequency array for digits (0–9)?**

int[] freq = new int[?];

A. 26  
B. 10  
C. 128  
D. 256  
**Answer: B**  
**Explanation:** 10 digits (0 to 9) → array of size 10 is needed.

**Q22. [Difficult] You want to check if one string is a permutation of another. Best logic?**  
A. Sort and compare  
B. HashSet  
C. Compare frequency arrays  
D. Recursion  
**Answer: C**  
**Explanation:** If both strings have same length and frequency arrays, they’re permutations.

**Q23. [Difficult] Which of the following operations is not O(1) with frequency arrays?**  
A. Increment count  
B. Set count  
C. Search for min frequency  
D. Access index  
**Answer: C**  
**Explanation:** Finding min in array takes O(n) time even if access is O(1).

**Q24. [Difficult] Which scenario will lead to a HashMap being preferred over frequency arrays?**  
A. Checking character frequency for only lowercase  
B. Validating 0–9 digits  
C. Frequency of emojis in a string  
D. Comparing two lowercase strings  
**Answer: C**  
**Explanation:** HashMaps support wide/unpredictable character sets like emojis or Unicode.

**Q25. [Difficult] You want to find the longest substring without repeating characters. Which combination is best?**  
A. Brute force + nested loop  
B. HashMap + Sliding Window  
C. Frequency array  
D. Sort and match  
**Answer: B**  
**Explanation:** Sliding window with a HashMap tracks characters and ensures max length efficiently.

**Q26. [Definition] What is a palindrome?**  
A. A string with unique characters  
B. A string that reads the same backward as forward  
C. A string with repeating characters  
D. A string with only vowels  
**Answer: B**  
**Explanation:** Palindromes are symmetrical around their center.

**Q27. [Definition] Which of the following is a palindrome?**  
A. "hello"  
B. "level"  
C. "string"  
D. "world"  
**Answer: B**  
**Explanation:** "level" reads the same forward and backward.

**Q28. [Definition] What is the minimum length of a non-empty palindromic substring?**  
A. 0  
B. 2  
C. 1  
D. Depends on string  
**Answer: C**  
**Explanation:** Any single character is a palindrome of length 1.

**Q29. [Definition] What is the time complexity of checking if a string is a palindrome?**  
A. O(n log n)  
B. O(1)  
C. O(n²)  
D. O(n)  
**Answer: D**  
**Explanation:** Comparing first and last characters toward the center takes O(n) time.

**Q30. [Definition] In total, how many palindromic substrings exist in "aaa"?**  
A. 3  
B. 4  
C. 6  
D. 7  
**Answer: C**  
**Explanation:** Substrings: "a", "a", "a", "aa", "aa", "aaa" → 6 total palindromic substrings.

**🔹 Part 2: Moderate-Level – Expand Around Center, Brute Force**

**Q31. [Moderate] Which approach is most common to count all palindromic substrings efficiently?**  
A. Two Pointers  
B. HashSet  
C. Expand Around Center  
D. Sorting  
**Answer: C**  
**Explanation:** Expanding from each center in a string is a widely used and optimal approach.

**Q31. [Moderate] How many centers exist for palindromic substring expansion in a string of length n?**  
A. n  
B. 2n  
C. n²  
D. n/2  
**Answer: B**  
**Explanation:** Each character and the gap between characters can be centers → 2n - 1 total centers.

**Q33. [Moderate] Which method finds all palindromic substrings in O(n²) time and O(1) space?**  
A. KMP Algorithm  
B. Manacher’s Algorithm  
C. Expand Around Center  
D. Trie  
**Answer: C**  
**Explanation:** Expand Around Center is simple and efficient with O(n²) time and O(1) space.

**Q34. [Moderate] What is the output of this function when s = "aba"?**

boolean isPalindrome(String s) {

int i = 0, j = s.length() - 1;

while (i < j) {

if (s.charAt(i++) != s.charAt(j--)) return false;

}

return true;

}

A. true  
B. false  
C. Compilation Error  
D. Runtime Exception  
**Answer: A**  
**Explanation:** The string "aba" is a valid palindrome.

**Q35. [Moderate] Which of the following substrings of “ababa” is the longest palindrome?**  
A. "aba"  
B. "bab"  
C. "ababa"  
D. "aa"  
**Answer: C**  
**Explanation:** The entire string is symmetric and the longest palindromic substring.

**🔹 Part 3: Advanced/Difficult (Dynamic Programming, Hashing, Manacher’s)**

**Q36. [Difficult] What is the time and space complexity of the DP approach to find the longest palindromic substring?**  
A. O(n²) time, O(n²) space  
B. O(n log n), O(n)  
C. O(n), O(n²)  
D. O(n), O(1)  
**Answer: A**  
**Explanation:** 2D table dp[i][j] stores whether s[i..j] is a palindrome → O(n²) space and time.

**Q37. [Difficult] Which condition is used in dynamic programming to check s[i..j] is a palindrome?**  
A. s[i] == s[j] && j - i <= 1  
B. s[i] == s[j] && dp[i+1][j-1] == true  
C. Both A and B  
D. Only A  
**Answer: C**  
**Explanation:** For base case and recursive check, both are used.

**Q38. [Difficult] Which of the following is the most time-efficient algorithm to find the longest palindromic substring?**  
A. Hashing  
B. Manacher’s Algorithm  
C. Z-Algorithm  
D. KMP  
**Answer: B**  
**Explanation:** Manacher’s Algorithm finds the longest palindrome in O(n) time.

**Q39. [Difficult] Which algorithm uses modified string with ‘#’ delimiters to handle even-length palindromes?**  
A. Expand Around Center  
B. DP Table  
C. Manacher’s Algorithm  
D. Z-Algorithm  
**Answer: C**  
**Explanation:** Manacher’s uses delimiters like '#' to unify even/odd palindrome processing.

**Q40. [Difficult] What is the return value of longestPalindrome("abccbaabcd")?**  
A. "abccba"  
B. "bccbaab"  
C. "cc"  
D. "abcd"  
**Answer: A**  
**Explanation:** "abccba" is the longest palindromic substring in the given string.

**🔹 Part 4: Scenario-Based / Interview**

**Q41. [Difficult] Given a string of length 100,000, which algorithm should you use to find the longest palindromic substring efficiently?**  
A. DP  
B. Hashing  
C. Manacher’s Algorithm  
D. Expand Around Center  
**Answer: C**  
**Explanation:** Only Manacher’s achieves linear time suitable for large input sizes.

**Q42. [Difficult] In an interview, you're asked to find the total number of palindromic substrings. What’s your best strategy?**  
A. HashMap with index pairs  
B. DP with 2D matrix  
C. Expand Around Center  
D. Recursion with memoization  
**Answer: C**  
**Explanation:** Expand Around Center counts all substrings efficiently with O(n²) time and O(1) space.

**Q43. [Difficult] What should be the base case in a DP table for palindromic substrings?**  
A. dp[i][i] = true  
B. dp[i][i+1] = true if s[i] == s[i+1]  
C. dp[i][j] = false  
D. A and B  
**Answer: D**  
**Explanation:** Single character substrings are palindromes and so are even-length pairs if matched.

**Q44. [Difficult] What is the value of count for s = "abcd" using palindromic substrings count function?**  
A. 4  
B. 6  
C. 2  
D. 1  
**Answer: A**  
**Explanation:** All 4 single characters are individual palindromes.

**Q45. [Difficult] What causes the DP method to fail for very large strings (e.g., 10⁵)?**  
A. It is inaccurate  
B. It consumes too much memory  
C. Time complexity is too high  
D. Both B and C  
**Answer: D**  
**Explanation:** O(n²) space and time make DP unusable for strings of size >10⁴–10⁵.

**Part 1: Anagram Grouping**

**Q46. [Definition] Two strings are anagrams if:**  
A. They contain the same characters in different order  
B. They are palindromes  
C. One is substring of another  
D. They start with the same letter  
**Answer: A**  
**Explanation:** Anagrams contain the same characters with same frequency.

**Q47. [Definition] Which of the following pairs are anagrams?**  
A. "listen", "silent"  
B. "night", "thing"  
C. "loop", "pool"  
D. All of the above  
**Answer: D**  
**Explanation:** All these word pairs have matching characters and frequencies.

**Q48. [Moderate] What is the best way to check if two strings are anagrams (ignoring sort)?**  
A. Compare strings directly  
B. Use HashMap/Array to count character frequency  
C. Use recursion  
D. Use Set  
**Answer: B**  
**Explanation:** Character count matching is optimal for anagram verification.

**Q49. [Moderate] What is the time complexity of sorting strings to group anagrams?**  
A. O(n)  
B. O(n log n)  
C. O(m log m) per string  
D. O(1)  
**Answer: C**  
**Explanation:** Each string of length m takes O(m log m) to sort.

**Q50. [Moderate] Which data structure is most used in anagram grouping problems?**  
A. Queue  
B. HashMap<String, List<String>>  
C. Stack  
D. TreeMap  
**Answer: B**  
**Explanation:** HashMap stores sorted/frequency signature as key and list of anagrams as value.

**Q51. [Difficult] Which key is best for grouping anagrams using frequency counts (without sorting)?**  
A. Raw string  
B. Sorted string  
C. Frequency array converted to string (e.g. "2#0#1...")  
D. HashCode  
**Answer: C**  
**Explanation:** Frequency signature string is a compact and hashable key for anagram grouping.

**Q52. [Difficult] Why is sorting each string inefficient for large strings in anagram grouping?**  
A. Sorting is O(n²)  
B. Sorting changes original string  
C. Sorting is slower than frequency count (O(m log m) vs O(m))  
D. None  
**Answer: C**  
**Explanation:** Frequency array solution is linear (O(m)) while sorting is log-linear (O(m log m)).

**Q53. [Difficult] What will be the output group for input: ["eat", "tea", "tan", "ate", "nat", "bat"]?**  
A. 6 groups  
B. 3 groups: [["eat","tea","ate"], ["tan","nat"], ["bat"]]  
C. 2 groups  
D. All same group  
**Answer: B**  
**Explanation:** Grouping based on sorted or frequency-matching keys.

**Q54. [Difficult] What is the worst-case time complexity for grouping n strings of length m using sorting?**  
A. O(n)  
B. O(nm log m)  
C. O(n²)  
D. O(n log n)  
**Answer: B**  
**Explanation:** Each string of length m takes O(m log m), for n strings → O(nm log m).

**Q55. [Difficult] How can we reduce space usage in grouping anagrams without using extra list?**  
A. Use TreeMap  
B. Sort in-place  
C. Modify input array  
D. You cannot avoid extra space  
**Answer: D**  
**Explanation:** Grouping always requires extra space to group/track elements.

**🔹 Part 2: Sliding Window on Strings**

**Q56. [Definition] Sliding window technique is used when:**  
A. We need to find repeating elements  
B. We process substrings of fixed or variable length efficiently  
C. We sort strings  
D. We reverse substrings  
**Answer: B**  
**Explanation:** Sliding window helps scan over substrings with limited movement and memory.

**Q57. [Definition] Which problem is best solved using sliding window technique?**  
A. Reverse a string  
B. Count total anagrams of pattern in a string  
C. Convert string to integer  
D. Find all palindromes  
**Answer: B**  
**Explanation:** Fixed-size sliding window is ideal for pattern/anagram matching.

**Q58. [Moderate] What is the window size for finding all anagrams of "abc" in a string s?**  
A. 1  
B. 2  
C. 3  
D. Variable  
**Answer: C**  
**Explanation:** Anagrams must match the pattern’s length.

**Q59. [Moderate] How do you check if the current window is an anagram of pattern p?**  
A. Compare string hash  
B. Compare frequency array of size 26  
C. Sort window  
D. Convert to set  
**Answer: B**  
**Explanation:** Character frequency arrays can be compared in O(1) time if size is fixed.

**Q60. [Moderate] What is the time complexity of finding all anagrams of pattern p in string s using sliding window?**  
A. O(n × m)  
B. O(n log m)  
C. O(n + m)  
D. O(n)  
**Answer: D**  
**Explanation:** Frequency array comparison and movement are done in linear time.

**Q61. [Moderate] What is the output of this code for s = "cbaebabacd", p = "abc"?**

// find start indices of anagrams of p in s

A. [0, 6]  
B. [1, 2, 5]  
C. [2, 4]  
D. [1, 3, 5]  
**Answer: A**  
**Explanation:** "cba" at index 0 and "bac" at index 6 are valid anagrams.

**Q62. [Difficult] Which algorithm uses two frequency arrays for matching pattern in sliding window problems?**  
A. KMP  
B. Rabin-Karp  
C. Anagram finder  
D. LPS Matcher  
**Answer: C**  
**Explanation:** Two frequency arrays—one for pattern, one for window—are compared during the scan.

**Q63. [Difficult] Which trick allows constant time frequency comparison when sliding the window?**  
A. Full sort every time  
B. Hash comparison  
C. Add 1 to entering char, subtract 1 from exiting char  
D. Binary Search  
**Answer: C**  
**Explanation:** Update char counts incrementally to avoid re-scanning entire window.

**Q67. [Difficult] What is the space complexity for sliding window anagram check on lowercase letters?**  
A. O(1)  
B. O(n)  
C. O(log n)  
D. O(n²)  
**Answer: A**  
**Explanation:** Frequency array of size 26 → constant space usage.

**Q68. [Difficult] You want to find the minimum window substring that contains all characters of a target string. What technique is best?**  
A. Expand Around Center  
B. Sliding Window + HashMap  
C. Sorting  
D. DP  
**Answer: B**  
**Explanation:** Variable size sliding window with character map tracks shortest valid substring.

**Part 1: Manacher’s Algorithm**

**Q69. [Definition] What is the primary use of Manacher’s Algorithm?**  
A. String sorting  
B. Finding longest palindromic substring in linear time  
C. Finding anagrams  
D. Pattern matching  
**Answer: B**  
**Explanation:** Manacher’s efficiently finds the longest palindromic substring in O(n) time.

**Q70. [Definition] What modification is made to the input string in Manacher’s Algorithm?**  
A. Characters are reversed  
B. Special characters like ‘#’ are added between characters  
C. String is sorted  
D. Extra whitespace is trimmed  
**Answer: B**  
**Explanation:** Delimiters like # are added to handle even and odd-length palindromes uniformly.

**Q73. [Definition] What is the time complexity of Manacher’s Algorithm?**  
A. O(n²)  
B. O(n log n)  
C. O(n)  
D. O(1)  
**Answer: C**  
**Explanation:** Manacher’s Algorithm runs in linear time.

**Q74. [Moderate] Which of the following strings will become "^#a#b#a#$" in preprocessing?**  
A. "aba"  
B. "abac"  
C. "ab"  
D. "abcba"  
**Answer: A**  
**Explanation:** "^" and "$" are boundaries, and '#' separates each character.

**Q75. [Moderate] What role does the array P[i] play in Manacher’s Algorithm?**  
A. Stores prefix sums  
B. Stores lengths of palindromes centered at i  
C. Stores hash values  
D. Stores substring indexes  
**Answer: B**  
**Explanation:** P[i] indicates the radius of palindrome centered at position i.

**Q76. [Moderate] What happens when the palindrome centered at i expands beyond the current right?**  
A. Update left and right  
B. Terminate the loop  
C. Re-initialize P[]  
D. Skip to next odd index  
**Answer: A**  
**Explanation:** left and right are updated when a longer palindrome is found.

**Q77. [Difficult] What is the length of the longest palindromic substring in "abcba"?**  
A. 3  
B. 5  
C. 4  
D. 2  
**Answer: B**  
**Explanation:** The entire string is a palindrome.

**Q78. [Difficult] Which optimization makes Manacher’s linear?**  
A. Avoiding recomputation of palindrome lengths using mirrored indices  
B. Using two pointers  
C. Sorting characters  
D. Brute force substring comparison  
**Answer: A**  
**Explanation:** Manacher’s leverages symmetry by using previously computed results.

**Q79. [Difficult] After preprocessing "racecar", how many characters are in the transformed string?**  
A. 13  
B. 15  
C. 9  
D. 5  
**Answer: A**  
**Explanation:** "^#r#a#c#e#c#a#r#$" → 13 characters.

**Q80. [Difficult] In Manacher’s algorithm, what is returned as the final result?**  
A. Max value in array P[]  
B. Index of center  
C. P[], L[], and R[]  
D. Count of odd palindromes  
**Answer: A**  
**Explanation:** The max value in P[] gives the radius of the longest palindromic substring.

**🔹 Part 2: Z-Algorithm (Pattern Matching)**

**Q81. [Definition] What is the Z-array in the Z-algorithm?**  
A. Stores hash values  
B. Stores LPS values  
C. Stores length of longest substring starting at i that matches the prefix  
D. Stores frequency count  
**Answer: C**  
**Explanation:** Z[i] = longest substring starting at i which is also a prefix of the entire string.

**Q82. [Definition] What is the time complexity of the Z-algorithm?**  
A. O(n log n)  
B. O(n)  
C. O(n²)  
D. O(1)  
**Answer: B**  
**Explanation:** Z-algorithm computes all Z-values in linear time using windowing.

**Q83. [Moderate] What is the output Z-array for the string "aabxaabxcaabxaabxay"?**  
A. Problem-specific, depends on match lengths  
B. All zeros  
C. All n  
D. All ones  
**Answer: A**  
**Explanation:** The Z-array is unique to each pattern and reflects its prefix structure.

**Q84. [Moderate] How do you use Z-algorithm for pattern searching in string S with pattern P?**  
A. Build Z-array of P + $ + S  
B. Build Z-array of S only  
C. Build suffix array  
D. Use Trie  
**Answer: A**  
**Explanation:** Concatenate P + "$" + S, then scan for Z[i] = P.length().

**Q85. [Moderate] What is the purpose of the special separator character ($ or #) in pattern + text?**  
A. Remove whitespace  
B. Separate pattern from text to avoid overlap  
C. Improve time complexity  
D. Add to prefix  
**Answer: B**  
**Explanation:** Prevents false Z-matches across pattern-text boundaries.

**Q86. [Moderate] Which of the following can Z-algorithm solve efficiently?**  
A. Prefix queries  
B. Substring pattern search  
C. Repetition finding  
D. All of the above  
**Answer: D**  
**Explanation:** Z-values can be reused to solve multiple string problems.

**Q87. [Difficult] If Z[i] = pattern.length() in a Z-array for "P$S" → what does it mean?**  
A. Mismatch  
B. Partial match  
C. Full match of pattern starting at index i - (pattern.length() + 1) in S  
D. Error  
**Answer: C**  
**Explanation:** That index in S marks the start of a complete pattern match.

**Q88. [Difficult] In Z-algorithm, what does maintaining a [L, R] window help with?**  
A. Avoids recomputation  
B. Speeds up hash comparison  
C. Avoids overflow  
D. Ensures string is reversed  
**Answer: A**  
**Explanation:** Reuse previous matches to extend current match without restarting.

**Q89. [Difficult] When is Z-algorithm better than KMP for pattern matching?**  
A. When preprocessing is costly  
B. When patterns repeat heavily  
C. When all prefix matches are needed  
D. Never  
**Answer: C**  
**Explanation:** Z-algorithm is excellent for multiple prefix or substring matches.

**Q90. [Difficult] What happens if the pattern is equal to the entire string? Z[1] = ?**  
A. n - 1  
B. n  
C. 0  
D. n/2  
**Answer: A**  
**Explanation:** Since Z[0] is always 0, Z[1] holds the length of the full match (n - 1).