100 MCQs on DSA Topics:

Arrays, Prefix Sum, Sliding Window, Two Pointer, Kadane's Algorithm, Merge Sort Problems

Part 1: Definition-Based Questions (40)

Q1. [Definition] What is the time complexity to access an element at a specific index in an array?
A. O(n)
B. O(log n)
C. O(1)
D. O(n log n)
Answer: C
Explanation: Arrays allow random access via indices, making element access O(1).

Q2. [Definition] What does the prefix sum technique aim to optimize?
A. Sorting an array
B. Reducing time complexity of range sum queries
C. Finding subarrays
D. Checking for duplicates
Answer: B
Explanation: Prefix sum arrays are precomputed to quickly calculate the sum of elements in a given range.

Q3. [Definition] Which algorithm is used to find the maximum sum of a contiguous subarray in linear time?
A. Merge Sort
B. Binary Search
C. Kadane's Algorithm
D. Quick Sort
Answer: C
Explanation: Kadane’s Algorithm efficiently finds the maximum subarray sum in O(n) time.

Q4. [Definition] What does a two pointer technique typically involve?
A. Two nested loops
B. Recursion
C. Two variables iterating through the array
D. Hashing elements
Answer: C
Explanation: Two pointers usually iterate from different ends or points in the array to solve problems efficiently.

Q5. [Definition] What is the sliding window technique commonly used for?
A. Sorting arrays

B. Calculating subarray sums or properties
C. Searching in trees
D. Counting inversions
Answer: B
Explanation: Sliding window technique is useful for calculating properties (like sum, max) of subarrays in linear time.

Q6. [Definition] What is the main advantage of the two pointer technique?
A. Reduces space complexity
B. Avoids recursion
C. Reduces nested loops in some problems
D. Guarantees optimal result
Answer: C
Explanation: The two pointer technique is often used to avoid brute force nested loop solutions.

Q7. [Definition] What is the worst-case time complexity of Merge Sort?
A. O(n)
B. O(n^2)
C. O(n log n)
D. O(log n)
Answer: C
Explanation: Merge Sort consistently performs at O(n log n) in all cases.

Q8. [Definition] In the prefix sum array, what does prefix[i] represent?
A. Sum of all elements from index i to end
B. Sum of all elements from start to index i
C. Average of first i elements
D. Maximum of first i elements
Answer: B
Explanation: prefix[i] usually stores the sum of array[0] to array[i].

Q9. [Definition] Which data structure is preferred when you need fast lookup and constant-time access?
A. Queue
B. Stack
C. Array
D. Linked List
Answer: C
Explanation: Arrays provide O(1) access time via indexing.

Q10. [Definition] What is the primary goal of Kadane's Algorithm?
A. Sorting arrays
B. Finding subarrays with maximum product
C. Finding maximum sum subarray
D. Calculating prefix sum

Answer: C
Explanation: Kadane's Algorithm finds the subarray with the maximum sum in linear time.

Q11. [Definition] What is the primary use of a prefix sum array?
A. Reduce memory consumption
B. Simplify string matching
C. Enable quick range sum queries
D. Speed up sorting
Answer: C
Explanation: Prefix sums allow fast computation of sums between any two indices.

Q12. [Definition] What is the time complexity of creating a prefix sum array?
A. O(1)
B. O(log n)
C. O(n)
D. O(n^2)
Answer: C
Explanation: One pass through the array is required to compute the prefix sums.

Q13. [Definition] Which approach does the sliding window technique improve upon?
A. Binary search
B. Nested loops
C. Recursion
D. Merge sort
Answer: B
Explanation: Sliding window avoids brute-force nested loops for subarray problems.

Q14. [Definition] What is a common condition for applying the two pointer technique?
A. The array is sorted
B. The array contains only positive integers
C. The array is multi-dimensional
D. The array contains duplicates
Answer: A
Explanation: Many two pointer problems require sorted arrays to efficiently find pairs.

Q15. [Definition] In the context of merge sort, what is a key step that enables sorting?
A. Random partitioning
B. Merging two sorted halves
C. Using a stack
D. Applying prefix sums
Answer: B
Explanation: Merge sort divides arrays and merges sorted halves recursively.

Q16. [Definition] How are inversion counts related to merge sort?
A. They are not related
B. Merge sort's merge step can be adapted to count inversions

C. Inversions are counted during sorting
D. Only quick sort can be used
Answer: B
Explanation: The merge step is ideal for counting inversions by tracking swaps.

Q17. [Definition] What is the time complexity of the two pointer technique on a sorted array?
A. O(n^2)
B. O(n log n)
C. O(n)
D. O(log n)
Answer: C
Explanation: Both pointers traverse the array linearly, so it's O(n).

Q18. [Definition] What problem does Kadane's algorithm solve?
A. Count sort
B. Maximum subarray sum
C. Shortest path
D. Array reversal
Answer: B
Explanation: Kadane's algorithm finds the contiguous subarray with the largest sum.

Q19. [Definition] Which operation is NOT allowed on fixed-size arrays in Java?
A. Reading an element
B. Modifying an element
C. Adding a new element
D. Initializing the array
Answer: C
Explanation: Arrays in Java have a fixed size and cannot grow dynamically.

Q20. [Definition] Which of the following algorithms is divide-and-conquer based?
A. Insertion Sort
B. Bubble Sort
C. Merge Sort
D. Linear Search
Answer: C
Explanation: Merge Sort is a classic example of a divide-and-conquer algorithm.

Q21. [Definition] Which DSA concept is best suited to solve the 2-sum problem in a sorted array?
A. HashMap
B. Binary Search
C. Two Pointer Technique
D. Stack
Answer: C
Explanation: Two pointers from both ends can be used efficiently in sorted arrays.

Q22. [Definition] What does an inversion in an array mean?
A. A sorted pair
B. A pair (i, j) such that i < j and arr[i] > arr[j]
C. A duplicate pair
D. A prefix pair
Answer: B
Explanation: Inversion indicates disorder where a larger element appears before a smaller one.

Q23. [Definition] What is the base case for Kadane's algorithm initialization?
A. 0
B. Integer.MIN_VALUE
C. The first array element
D. The maximum prefix sum
Answer: C
Explanation: Initialization starts with the first element for both current and max values.

Q24. [Definition] When is a sliding window most useful?
A. When elements must be removed from the middle
B. When dealing with fixed-size or dynamic subarrays
C. For recursive tree traversal
D. In graphs
Answer: B
Explanation: Sliding windows allow processing of continuous subarrays efficiently.

Q25. [Definition] What is the best time complexity to find the max sum subarray?
A. O(n log n)
B. O(n^2)
C. O(n)
D. O(n!)
Answer: C
Explanation: Kadane's algorithm accomplishes it in O(n) time.

Q26. [Definition] Which of these algorithms divides the array into two halves and processes them recursively?
A. Linear Search
B. Bubble Sort
C. Merge Sort
D. Prefix Sum
Answer: C
Explanation: Merge Sort recursively divides the array and then merges sorted halves.

Q27. [Definition] What is a prefix sum array used for in coding interviews?
A. Searching
B. Reducing loop complexity for sum ranges
C. Finding duplicates
D. Heap operations

Answer: B
Explanation: It precomputes cumulative sums for efficient subarray queries.

Q28. [Definition] The main benefit of merge sort over bubble sort is:
A. Simpler logic
B. In-place operation
C. Consistent performance
D. Better worst-case complexity
Answer: D
Explanation: Merge sort has O(n log n) in the worst case, better than O(n^2) of bubble sort.

Q29. [Definition] Which algorithm is suitable for maximum sum of k consecutive elements?
A. Kadane's Algorithm
B. Binary Search
C. Sliding Window
D. DFS
Answer: C
Explanation: Sliding window efficiently tracks sum of fixed-length subarrays.

Q30. [Definition] What does the term 'window' in sliding window refer to?
A. Binary partitioning
B. A fixed-size subarray
C. Dynamic programming array
D. Graph node set
Answer: B
Explanation: It represents a subarray of the input array over which computation is done.

Q31. [Definition] In 3-sum problem, which approach reduces time from O(n^3) to O(n^2)?
A. Brute force
B. Hashing
C. Sorting + Two Pointer
D. DFS
Answer: C
Explanation: Sorting the array and using two pointers reduces complexity.

Q32. [Definition] Merge sort is preferred over quick sort when:
A. Space is limited
B. Worst-case time is critical
C. Median is needed
D. Array is already sorted
Answer: B
Explanation: Merge sort offers predictable O(n log n) even in worst-case scenarios.

Q33. [Definition] What defines the end of a subarray?
A. Beginning of another
B. Any valid index >= start index

C. End of array only
D. Where a 0 is encountered
Answer: B
Explanation: Subarrays can start and end at any valid indices.

Q34. [Definition] What is required for an efficient 2-pointer solution?
A. Duplicates
B. Sorted array
C. Queue implementation
D. Reversed array
Answer: B
Explanation: Sorted arrays help two pointers converge optimally.

Q35. [Definition] The maximum subarray sum problem is solved optimally by:
A. Greedy Algorithm
B. Kadane's Algorithm
C. Dynamic Programming
D. Divide and Conquer
Answer: B
Explanation: Kadane's algorithm is a greedy approach for maximum subarray sum.

Q36. [Definition] What's the space complexity of Kadane's Algorithm?
A. O(n)
B. O(log n)
C. O(1)
D. O(n^2)
Answer: C
Explanation: It uses only two variables to track current and max sums.

Q37. [Definition] In array problems, what does 'in-place' mean?
A. Using recursion
B. Using extra array
C. Without using extra space
D. Running inside a function
Answer: C
Explanation: In-place means modifying the original array without extra space.

Q38. [Definition] Which of these algorithms is stable?
A. Quick Sort
B. Merge Sort
C. Heap Sort
D. Selection Sort
Answer: B
Explanation: Merge Sort maintains the order of equal elements.

Q39. [Definition] Which sorting algorithm is not comparison-based?
A. Merge Sort
B. Quick Sort
C. Count Sort
D. Bubble Sort
Answer: C
Explanation: Count Sort uses counting, not comparisons.

Q40. [Definition] What's the basic idea of prefix sum optimization?
A. Reduce recursion depth
B. Precompute cumulative sums to avoid repeated work
C. Create new arrays for all ranges
D. Use stack for storage
Answer: B
Explanation: It speeds up queries by avoiding repetitive summation loops.

Part 2: Moderate Difficulty (Error Identification & Debugging - 35)

Q41. [Moderate] What will be the output of the following code?

```
int[] arr = {1, 2, 3};
System.out.println(arr[3]);
```

A. 3
B. 0
C. ArrayIndexOutOfBoundsException
D. Compilation Error
Answer: C
Explanation: Index 3 is out of bounds for a 3-element array (indices 0 to 2).

Q42. [Moderate] Which line contains the error in the following code snippet to calculate prefix sums?

```
int[] arr = {1, 2, 3};
int[] prefix = new int[arr.length];
prefix[0] = arr[0];
for (int i = 0; i < arr.length; i++) {
    prefix[i] = prefix[i - 1] + arr[i];
}
```

A. Line 3
B. Line 4
C. Line 5
D. Line 6
Answer: D
Explanation: When i = 0, prefix[i - 1] results in accessing prefix[-1], which is invalid.

Q43. [Moderate] What is the output of the following code?

```
int[] arr = {3, 2, 1};
Arrays.sort(arr);
System.out.println(arr[0]);
```

A. 3
B. 2
C. 1
D. Compilation Error
Answer: C
Explanation: Arrays.sort() sorts the array in ascending order, making the first element the smallest.

Q44. [Moderate] Identify the issue in this code for max subarray sum:

```
int maxSum = 0;
for (int i = 0; i < arr.length; i++) {
    maxSum = Math.max(maxSum, arr[i]);
}
```

A. Logic for max subarray sum is incorrect
B. Runtime error
C. Compilation error
D. Works fine
Answer: A
Explanation: This computes the max element, not the max subarray sum.

Q45. [Moderate] What does this code do?

```
int[] arr = {1, 2, 3};
for (int i = 0; i <= arr.length; i++) {
    System.out.print(arr[i]);
}
```

A. Prints 123
B. Compilation error
C. Runtime error
D. Prints 12
Answer: C
Explanation: Accessing arr[arr.length] causes ArrayIndexOutOfBoundsException.

Q46. [Moderate] What will this print?

```
int[] a = {1, 2, 3};
a[1] = a[1] + a[2];
System.out.println(a[1]);
```

A. 3
B. 5
C. 2
D. 1
Answer: B
Explanation: a[1] becomes 2 + 3 = 5.

Q47. [Moderate] Find the error:

```
int[] prefix = new int[n];
prefix[0] = arr[0];
for (int i = 1; i < n; i++)
    prefix[i] = arr[i] + prefix[i];
```

A. No error
B. prefix[i] should use prefix[i-1]
C. Loop should start from 0
D. arr not initialized
Answer: B
Explanation: The correct computation is prefix[i] = arr[i] + prefix[i - 1].

Q48. [Moderate] Find the error in this sliding window implementation:

```
int maxSum = 0;
for (int i = 0; i < arr.length - k; i++) {
    int sum = 0;
    for (int j = i; j < i + k; j++) {
        sum += arr[j];
    }
    maxSum = Math.max(maxSum, sum);
}
```

A. Loop bounds are incorrect
B. sum is not reset
C. k must be less than array size
D. Missing print statement
Answer: A
Explanation: The outer loop should go till arr.length - k + 1 to cover the last valid window.

Q49. [Moderate] What is the output of the following?

```
int[] arr = {1, 2, 3, 4};
int sum = 0;
for (int i = 1; i < arr.length; i++) {
    sum += arr[i] - arr[i - 1];
}
System.out.println(sum);
```

A. 3
B. 6

C. 0
D. 1
Answer: A
Explanation: Sum is (2-1) + (3-2) + (4-3) = 1 + 1 + 1 = 3.

Q50. [Moderate] What's the bug in this merge sort implementation?

```
void mergeSort(int[] arr) {
    if (arr.length <= 1) return;
    int mid = arr.length / 2;
    int[] left = Arrays.copyOfRange(arr, 0, mid);
    int[] right = Arrays.copyOfRange(arr, mid, arr.length);
    mergeSort(left);
    mergeSort(right);
    merge(arr, left, right);
}
```

A. Infinite recursion
B. No base case
C. merge function undefined
D. Wrong range
Answer: C
Explanation: merge function must be implemented to merge the sorted halves.

Q51. [Moderate] Identify the issue in this code for calculating the maximum subarray sum using Kadane's Algorithm:

```
int maxSum = 0;
int currSum = 0;
for (int i = 0; i < arr.length; i++) {
    currSum += arr[i];
    if (currSum < 0) currSum = 0;
    maxSum = Math.max(maxSum, currSum);
}
```

A. Works correctly
B. maxSum initialization is wrong
C. Fails if all numbers are negative
D. Loop bound is off
Answer: C
Explanation: This version fails when all numbers are negative; a proper Kadane's should initialize with the first element.

Q52. [Moderate] What is the error in this 2-pointer code for finding a pair sum?

```
int l = 0, r = arr.length - 1;
while (l < r) {
    int sum = arr[l] + arr[r];
    if (sum == target) System.out.println("Found");
    else if (sum < target) r--;
```

```
    else l++;
}
```

A. r-- and l++ conditions are reversed
B. Infinite loop
C. Array not sorted
D. No return after print
Answer: A
Explanation: If sum < target, we should move left pointer up; if sum > target, move right pointer down.

Q53. [Moderate] Why might this prefix sum implementation be incorrect?

```
int[] prefix = new int[n];
prefix[0] = 0;
for (int i = 1; i < n; i++) {
    prefix[i] = prefix[i - 1] + arr[i];
}
```

A. prefix[0] should be arr[0]
B. Loop should start from 0
C. prefix array too small
D. arr not initialized
Answer: A
Explanation: Prefix sums should start with the first value of the original array.

Q54. [Moderate] What will this sliding window code print for arr = {1,2,3,4,5}, k = 3?

```
int maxSum = 0;
int sum = 0;
for (int i = 0; i < k; i++) sum += arr[i];
maxSum = sum;
for (int i = k; i < arr.length; i++) {
    sum += arr[i] - arr[i - k];
    maxSum = Math.max(maxSum, sum);
}
System.out.println(maxSum);
```

A. 9
B. 10
C. 12
D. 15
Answer: B
Explanation: Max window sum of size 3 in {1,2,3,4,5} is {3,4,5} = 12. But the logic correctly gives 12.

Q55. [Moderate] What is wrong with this code to check if array is sorted?

```
boolean sorted = true;
for (int i = 0; i <= arr.length; i++) {
```

```
    if (arr[i] > arr[i+1]) sorted = false;
}
```

A. i <= arr.length should be i < arr.length - 1
B. sorted should be initialized as false
C. Missing return statement
D. Works fine
Answer: A
Explanation: arr[i+1] will be out of bounds on the last iteration.

Q56. [Moderate] What does this code print for arr = {4,3,2,1}?

```
int count = 0;
for (int i = 0; i < arr.length; i++) {
    for (int j = i+1; j < arr.length; j++) {
        if (arr[i] > arr[j]) count++;
    }
}
System.out.println(count);
```

A. 4
B. 6
C. 3
D. 10
Answer: B
Explanation: It counts all inversions. For reverse sorted array of 4 elements, there are 6.

Q57. [Moderate] Which line causes error in this two-sum approach?

```
int l = 0, r = arr.length;
while (l < r) {
    int sum = arr[l] + arr[r];
    if (sum == target) return true;
    else if (sum < target) l++;
    else r--;
}
```

A. l = 0
B. r = arr.length
C. return true
D. l < r
Answer: B
Explanation: Index arr.length is out of bounds. Use r = arr.length - 1.

Q58. [Moderate] What is the issue with this prefix sum range query?

```
int[] prefix = new int[n];
for (int i = 1; i < n; i++) {
    prefix[i] = prefix[i-1] + arr[i];
}
```

```
int sum = prefix[j] - prefix[i];
```

A. Index out of bounds for prefix[j]
B. sum calculation is incorrect for i = 0
C. Prefix initialization is invalid
D. Loop is incorrect
Answer: B
Explanation: For i = 0, prefix[i-1] becomes prefix[-1], which is invalid.

Q59. [Moderate] What is the error in this merge function?

```
void merge(int[] arr, int[] left, int[] right) {
    int i = 0, j = 0, k = 0;
    while (i < left.length && j < right.length) {
        if (left[i] <= right[j]) arr[k++] = left[i++];
        else arr[k++] = right[j++];
    }
    while (i < left.length) arr[k++] = left[i++];
    while (j < right.length) arr[k++] = right[j++];
}
```

A. merge doesn't sort correctly
B. Array index error
C. arr is not large enough
D. Works perfectly
Answer: C
Explanation: arr should be of size left.length + right.length, or merging will fail.

Q60. [Moderate] Identify the logical issue in this maximum subarray code:

```
int max = Integer.MIN_VALUE;
for (int i = 0; i < arr.length; i++) {
    int sum = 0;
    for (int j = i; j < arr.length; j++) {
        sum += arr[j];
        max = Math.max(max, sum);
    }
}
```

A. Logic is wrong
B. Initialization of sum
C. Inefficient but correct
D. sum never resets
Answer: C
Explanation: This is a brute-force method; it works but takes O(n^2) time.

Q61. [Moderate] What issue exists in this code to compute inversion count?

```
int count = 0;
for (int i = 0; i < n; i++) {
```

```
    for (int j = i+1; j < n; j++) {
        if (arr[i] < arr[j]) count++;
    }
}
```

A. Missing equals check
B. Should be arr[i] > arr[j] for inversion
C. Inner loop should start at 0
D. Loop indices are invalid
Answer: B
Explanation: Inversion requires arr[i] > arr[j] with i < j.

Q62. [Moderate] Which case causes sliding window sum to break?

```
for (int i = 0; i <= arr.length - k; i++) {
    int sum = 0;
    for (int j = i; j < i + k; j++) {
        sum += arr[j];
    }
}
```

A. If k > arr.length
B. If arr has negative elements
C. If arr is sorted
D. If k = 1
Answer: A
Explanation: The range goes out of bounds when k > arr.length.

Q63. [Moderate] What happens if the prefix sum array is not initialized properly?

```
int[] prefix = new int[n];
for (int i = 1; i < n; i++) {
    prefix[i] = prefix[i-1] + arr[i];
}
```

A. prefix[0] remains 0
B. prefix[0] must be arr[0]
C. prefix will not represent correct sums
D. All of the above
Answer: D
Explanation: Prefix array must start with prefix[0] = arr[0] for correct computation.

Q64. [Moderate] What's the issue with this two pointer loop?

```
int i = 0, j = 0;
while (i < n && j < n) {
    if (arr[i] + arr[j] == target && i != j) return true;
    else if (arr[i] + arr[j] < target) j++;
    else i++;
}
```

A. i and j can both increase indefinitely
B. Possibility of infinite loop
C. Fails if i == j
D. All of the above
Answer: D
Explanation: i and j can skip correct pairs; condition i != j may be redundant.

Q65. [Moderate] What condition will fix the above code?
A. Set j = i+1 and use i < j loop
B. Use HashSet
C. Sort the array
D. Replace target
Answer: A
Explanation: Starting j = i+1 and keeping i < j avoids redundant comparisons and ensures $i \neq j$.

Q66. [Moderate] What is the potential bug in this Kadane's implementation?

```
int max = 0, sum = 0;
for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
    max = Math.max(max, sum);
    if (sum < 0) sum = 0;
}
```

A. Incorrect max initialization
B. sum condition should be before max update
C. Handles only positive elements
D. Works correctly
Answer: A
Explanation: max should be initialized with Integer.MIN_VALUE or first element to work with negative-only arrays.

Q67. [Moderate] Why might this two pointer approach for 3-sum fail?

```
for (int i = 0; i < arr.length - 2; i++) {
    int l = i + 1, r = arr.length - 1;
    while (l < r) {
        int sum = arr[i] + arr[l] + arr[r];
        if (sum == 0) System.out.println("Triplet found");
        else if (sum < 0) r--;
        else l++;
    }
}
```

A. r-- and l++ are reversed
B. Should sort the array first
C. Loop should break after finding triplet
D. Works fine

Answer: B
Explanation: Array must be sorted to use the two-pointer strategy effectively.

Q68. [Moderate] What's the issue if this code runs on an empty array?

```
int max = arr[0];
System.out.println(max);
```

A. Returns incorrect result
B. Prints 0
C. Throws ArrayIndexOutOfBoundsException
D. Compilation error
Answer: C
Explanation: Accessing arr[0] in an empty array will cause runtime exception.

Q69. [Moderate] What could be improved in this merge sort base condition?

```
if (arr.length == 0) return;
```

A. Check for null too
B. Use arr.length <= 1
C. Merge step is missing
D. Works fine
Answer: B
Explanation: A single-element array is already sorted; base case should be arr.length <= 1.

Q70. [Moderate] What is wrong with this max window sum code if k > arr.length?

```
int maxSum = 0;
for (int i = 0; i <= arr.length - k; i++) {
    int sum = 0;
    for (int j = i; j < i + k; j++) {
        sum += arr[j];
    }
    maxSum = Math.max(maxSum, sum);
}
```

A. No error
B. Logic is inefficient
C. Index out of bounds
D. k should be 1
Answer: C
Explanation: When k > arr.length, arr[j] access will exceed array bounds.

Q71. [Moderate] What is the error in using this for inversion count?

```
int count = 0;
for (int i = 0; i < n - 1; i++) {
    if (arr[i] > arr[i + 1]) count++;
```

```
}
```

A. Doesn't count all inversions
B. Outer loop range is incorrect
C. Works for sorted arrays only
D. All of the above
Answer: A
Explanation: This only checks adjacent elements, not all pairs (i < j).

Q72. [Moderate] What is the problem with this logic to find max in prefix sum array?

```
int max = prefix[0];
for (int i = 0; i < prefix.length; i++) {
    max = Math.max(max, prefix[i]);
}
```

A. Redundant first comparison
B. Off-by-one error
C. Index starts wrong
D. Works fine
Answer: A
Explanation: The first comparison is redundant since we start at i = 0.

Q73. [Moderate] What is the bug in using prefix[j] - prefix[i - 1] without check?
A. Invalid if i = 0
B. prefix not initialized
C. j must be greater than i
D. All of the above
Answer: A
Explanation: When i = 0, prefix[i-1] becomes prefix[-1], which is invalid.

Q74. [Moderate] Which is a limitation of two pointer technique?
A. Can't work on sorted arrays
B. Not usable when backtracking is required
C. Doesn't reduce time complexity
D. Only works with strings
Answer: B
Explanation: Two pointer solutions do not backtrack, so they aren't suited for problems requiring reversals.

Q75. [Moderate] Which of these is not an inversion pair?
Array: [2, 4, 1, 3, 5]
A. (2, 1)
B. (4, 1)
C. (4, 3)
D. (3, 5)

Answer: D
Explanation: 3 < 5 and 3 precedes 5, so it's not an inversion..

Part 3: Difficult (Output, Scenario-Based - 25)

Q76. [Difficult] What will be the output of the following Java program using Kadane's Algorithm?

```
int[] arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
int maxSoFar = arr[0], maxEndingHere = arr[0];
for (int i = 1; i < arr.length; i++) {
    maxEndingHere = Math.max(arr[i], maxEndingHere + arr[i]);
    maxSoFar = Math.max(maxSoFar, maxEndingHere);
}
System.out.println(maxSoFar);
```

A. 6
B. 7
C. 4
D. 5
Answer: A
Explanation: The maximum subarray sum is [4, -1, 2, 1], which adds up to 6.

Q77. [Difficult] You are given an array [1, 20, 6, 4, 5]. How many inversions are there?
A. 5
B. 4
C. 3
D. 2
Answer: A
Explanation: The inversions are: (20, 6), (20, 4), (20, 5), (6, 4), (6, 5).

Q78. [Difficult] What will this code output for arr = {2, -1, 2, 3, 4, -5}?

```
int maxSum = Integer.MIN_VALUE;
int currSum = 0;
for (int i = 0; i < arr.length; i++) {
    currSum += arr[i];
    if (currSum > maxSum) maxSum = currSum;
    if (currSum < 0) currSum = 0;
}
System.out.println(maxSum);
```

A. 10
B. 11
C. 9
D. 7
Answer: A
Explanation: The subarray [2, -1, 2, 3, 4] sums to 10, which is the maximum.

Q79. [Difficult] Given the array [5, 3, 2, 4, 1], find the number of inversions.
A. 6
B. 8
C. 7
D. 5
Answer: B
Explanation: The inversion pairs are (5,3), (5,2), (5,4), (5,1), (3,2), (3,1), (2,1), (4,1).

Q80. [Difficult] What will this code print?

```
int[] arr = {2, 1, 5, 1, 3, 2};
int k = 3;
int maxSum = 0;
for (int i = 0; i < k; i++) maxSum += arr[i];
int windowSum = maxSum;
for (int i = k; i < arr.length; i++) {
    windowSum += arr[i] - arr[i - k];
    maxSum = Math.max(maxSum, windowSum);
}
System.out.println(maxSum);
```

A. 8
B. 9
C. 7
D. 6
Answer: B
Explanation: Maximum window sum of size 3 is [5,1,3] = 9.