1. What is an in-place sorting algorithm?

An in-place sorting algorithm is one that sorts the data without requiring additional storage space proportional to the size of the input. It modifies the input array directly. The correct answer is:

a) It needs O(1) or O(logn) memory to create auxiliary locations

2. In the following scenarios, when will you use selection sort?

Selection sort is best used in scenarios where you do not require the fastest possible sorting but want a simple algorithm. It is generally not efficient for large arrays, but it can be useful in certain cases:

a) The input is already sorted
(selection sort will still run, but it does not take advantage of the already sorted order)

c) Large values need to be sorted with small keys
(selection sort does not care about the distribution of values, and it might be used when the range of the values is not large)

Selection sort is not particularly useful for large files or when sorting small values with large keys, so the best choices are a) and c).

3. Given an integer array and an integer k where k <= size of array, we need to return the kth smallest element of the array.

To find the kth smallest element efficiently, you can use the Quickselect algorithm, which has an average-case time complexity of O(n). Other methods like sorting the array and accessing the kth element directly will have a time complexity of O(n log n). If you use Quickselect:

Quickselect can be implemented with the following steps:

1. Choose a pivot element.
2. Partition the array around the pivot.
3. Recursively apply Quickselect to the appropriate partition.

4. Find the minimum operations required to sort the array in increasing order. In one operation, you can set each occurrence of one element to 0.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Function to find the length of the longest increasing subsequence
int longestIncreasingSubsequence(vector<int>& arr) {
    if (arr.empty()) return 0;

    vector<int> lis;
    for (int num : arr) {
        auto it = lower_bound(lis.begin(), lis.end(), num);
        if (it == lis.end()) {
            lis.push_back(num);
        } else {
            *it = num;
        }
    }
    return lis.size();
}

// Function to calculate minimum operations required to sort the array
int minOperationsToSort(int arr[], int n) {
    vector<int> vec(arr, arr + n);
    int lisLength = longestIncreasingSubsequence(vec);
    return n - lisLength;
}

int main() {
    int arr[] = {10, 20, 10, 30, 20, 50};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Minimum operations required to sort the array: " <<
minOperationsToSort(arr, n) << endl;
    return 0;
}
```

5. Given an array `arr[]` containing `n` integers, the task is to find an integer `K` such that after replacing each and every index of the array by `|ai − K|` where `( i ∈ [1, n])`, results in a sorted array. If no such integer exists that satisfies the above condition, then return -1.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <set>

using namespace std;

// Function to check if replacing each element with |ai - K| results in a sorted array
bool isSortedAfterTransformation(vector<int>& arr, int K) {
    vector<int> transformed;
    for (int num : arr) {
        transformed.push_back(abs(num - K));
    }
    // Check if transformed array is sorted
    for (size_t i = 1; i < transformed.size(); ++i) {
        if (transformed[i] < transformed[i - 1]) {
            return false;
        }
    }
    return true;
}

// Function to find the integer K that satisfies the condition
int findKForSortedTransformation(vector<int>& arr) {
    if (arr.empty()) return -1;

    // Try each unique value in the array as a candidate for K
    set<int> uniqueValues(arr.begin(), arr.end());
    for (int K : uniqueValues) {
        if (isSortedAfterTransformation(arr, K)) {
            return K;
        }
    }

    return -1;
}

int main() {
    vector<int> arr = {1, 5, 3, 9}; // Example array
    int K = findKForSortedTransformation(arr);
    if (K != -1) {
        cout << "The integer K that makes the transformed array sorted is " << K << endl;
    } else {
        cout << "No such integer K exists." << endl;
    }

    return 0;
}
```