C++ Assignments | Problems on sorting | Week 9

1.What is an in-place sorting algorithm?

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

b) The input is already sorted and in-place

c) It requires additional storage

d) It requires additional space

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

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

a) The input is already sorted

b) A large file has to be sorted

c) Large values need to be sorted with small keys

d) Small values need to be sorted with large keys

Answer:-b) A large file has to be sorted

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.

Answer:-To find the kth smallest element in an array efficiently, you have several options:

- **Sorting**: Sort the array and directly access the kth element. This takes O(n log n) time complexity.
- **Min-Heap**: Use a min-heap to extract the minimum k times, which also takes O(n log k) time complexity.
- **Quickselect algorithm**: This algorithm is like quicksort but focuses on partitioning around the kth smallest element, achieving average O(n) time complexity.

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.

Answer:-

To sort the array with minimum operations where each operation sets occurrences of one element to 0, consider:

- **Counting Sort**: Count occurrences of each element and reconstruct the array. This can be done in O(n + k) time, where k is the range of elements.
- **Frequency Array Approach**: Similar to counting sort, but directly operates on the original array to reduce elements to 0.

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

int findKForSortedTransformation(std::vector<int>& arr) {
   int n = arr.size();
   if (n == 0) return -1;

   std::sort(arr.begin(), arr.end());

   int medianIndex = n / 2;
   return arr[medianIndex];
}
```

5.Given an array, arr[] containing n integers, the task is to find an integer (say 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.

To find the integer K such that replacing each element by |ai - K| results in a sorted array:

- **Middle Element**: If the array is sorted in non-decreasing order, K should ideally be the median of the array. This ensures that after transformation, the array remains sorted.

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

int findKForSortedTransformation(std::vector<int>& arr) {
   int n = arr.size();
   if (n == 0) return -1;

   std::sort(arr.begin(), arr.end());

   int medianIndex = n / 2;
   return arr[medianIndex];
}
```