DESIGN DOCUMENT

Introduction

This project implements a recursive algorithm to find the **smallest kth element** in an unsorted array using partitioning. The algorithm always uses the **first element as the pivot**, partitions the array into three regions (≤ pivot, pivot, > pivot) and recursively searches the correct partition until the kth element is found.
The purpose is to gain experience with **recursive problem solving**, **array manipulation with pointers**, and **dynamic memory management**.

Data Structures

- **Dynamic Array**: The array is allocated at runtime using the new operator.
- **Integers**: Used for pivot index, partitioning indices, and tracking the kth rank.

Functions:

partition (int *arr, int left, int right)

- Rearrange elements so that values ≤ pivot are on the left, pivot is placed in its correct position, and values > pivot are on the right.
- Returns the pivot's final index.
- Handles the special case when **S1 (≤ pivot)** is empty.

Pseudocode:

```
pivot = arr[left]
i = left + 1
j = right
while true:
   while i <= right and arr[i] <= pivot: i++
   while j >= left+1 and arr[j] > pivot: j--
   if i > j: break
   swap arr[i], arr[j]
swap arr[left], arr[j]
return j
```

1. **kSmall(int *arr, int left, int right, int k)**
    - Recursive function to find the kth smallest element.
    - Compares k with pivot's rank and recurses accordingly.

Pseudocode:

```
if left <= right:
   pivotIndex = partition(arr, left, right)
   rank = pivotIndex - left + 1
   if k == rank:
      return arr[pivotIndex]
   else if k < rank:
      return kSmall(arr, left, pivotIndex - 1, k)
   else:
      return kSmall(arr, pivotIndex + 1, right, k - rank)
else:
   error: k is out of range
```

2. **main()**
    - Reads number of elements, array elements, and k.
    - Allocates array dynamically.
    - Calls kSmall and prints result.
    - Frees memory with delete[].

Structure Chart: