COP4531 Spring 2025 Project-1 using C++ (due date 10 March 2025)

Implementation of Quicksort with Animation and Performance Analysis

1. Introduction and Overview

- **Objective**: The objective of this project is to implement Quicksort algorithm and animate the sorting process using a stack of stars (*) for each cell of the array storing the numbers to be sorted. This animation will help visualize the algorithm's execution at each step while displaying the time taken for completing the sort. The project will also analyze the performance of Quicksort for three pivot strategies by repeating the process for different array sizes (n = 10, 20, 30, 40), calculating average sort times, plotting the results using MS Excel and analyzing the results.

2. Project Requirements

- **Input**:
  - An array of size n, where n starts from 10 and increases in increments of 10 up to 40.
  - Randomly generated integers between 1 and 10 stored in each array cell.
  - Using three different pivots i.e. first, last, and middle value.
  - Timer to control speed of animation.
- **Output**:
  - An animation of the sorting process in ascending order, showing the array as a stack/column of stars (*) corresponding to the number in the array cell.
  - The value selected as pivot shown by red stack of stars * (as shown in course material).
  - Complete sorting time for each iteration of the Quicksort algorithm for each pivot value.
  - MS Excel graphs plotting the relationship between array size (n) on x-axis and the average sorting time taken on y-axis for each value of pivot.

3. Detailed Steps

Step 1: Set up Random Numbers

- Generate a random array of n integers between 1 and 10 (size of array starts at n=10, increments by 10 to a maximum of n=40).
- **Task**:
  - Generate an array of n random numbers.
  - Display the numbers as a stack of *, where the number in each array position determines the count of stars drawn.

Step 2: Visual Representation of Array

- Create a visual representation of the array by drawing stacks/column of stars (*).
- For example, for an array element of value 7, 3, 6, 4, 5, 5, 9, 1, 4, and 6 would display as follows:

							*			
							*			
	*						*			
	*		*				*			*
	*		*		*	*	*			*
	*		*	*	*	*	*		*	*
	*	*	*	*	*	*	*		*	*
	*	*	*	*	*	*	*		*	*
	*	*	*	*	*	*	*	*	*	*
contents	**7**	**3**	**6**	**4**	**5**	**5**	**9**	**1**	**4**	**6**
index	**[0]**	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**

Fig-1: Contents of array size 10 displayed as a stack of stars, red color for the pivot selected

Step 3: Implement Quicksort

- Implement the **Quicksort algorithm** with pivot selection strategies:
  - **Pivot as the first value** in the array.
  - **Pivot as the last value** in the array.
  - **Pivot as the middle value** in the array.
- **Task**:
  - Create a function that sorts the array using Quicksort and displays the array contents at each iteration as shown in Fig-1.
  - For each pivot strategy, ensure that each partitioning step is visualized by updating the screen with the current state of the array (stacks of *).

Step 4: Animation of Quicksort Iterations

- **Task**: After each partitioning step, update the array on the screen to show the new arrangement of values (Fig-1).
- Update the display dynamically by updating the array contents by overwriting on Fig-1. Set a timer for displaying results of iteration on the screen, giving impression of an animation where the order of array values and pivot changes for each iteration of Quicksort.

Step 5: Timing Each Iteration

- **Task**: Record the time taken for each iteration (partitioning step) of the Quicksort algorithm.

- Determine the elapsed time for each partitioning step and display the complete sorting time when Quicksort terminates.

Step 6: Performance Analysis

- **Task**:
    - For a certain value of n (say) 10, and pivot (say) first value, run the application thrice and take the average of the three execution times. This is done for three pivot values so nine iterations with n= 10. When done for n = 20, 30 and 40, total 36 iterations.
    - Take the average time taken for each complete run of Quicksort with each value of n and each value of pivot.
    - For each value of n, plot the average time on the graphs with n on the x-axis and average time taken on the y-axis and identify the pivot used.

4. Experimental Setup

- **Range of n**: Start with n=10 and increment by 10, testing up to n=40 (i.e., n = 10, 20, 30, 40).
- **Run the application three times for each n**:
    - This will allow for averaging out the time results and mitigating any fluctuations due to system performance.

5. Graphical Representation

- After the Quicksort algorithm runs for each value of n and all three pivot strategies, use the collected time data to generate Excel graphs.
- **Graph Details**:
    - **X-axis**: The array size n (values 10, 20, 30, 40).
    - **Y-axis**: The average time taken for sorting (measured in milliseconds).
    - **Graph Types**: Line chart showing the performance comparison for the three pivot strategies (first, last, and middle).

6. Code Structure Outline

- **Step 1: Random Number Generation**
    - Generate random numbers between 1 and 10.
    - Display these numbers as stack/column of stars.
- **Step 2: Quicksort Algorithm**
    - Implement Quicksort with three pivot strategies (first, last, middle).
    - Update the array visualization on each partitioning iteration.
    - Record the time for each partitioning iteration.
- **Step 3: Time Recording**
    - Measure time for each iteration and calculate the average time for each run of the application.

- **Step 4: Graphing the Results**
  - After running the application number of time as required, use MS Excel to plot graphs showing n vs. average time for each pivot value.

7. Expected Results

- **Visualization**: The user should see a smooth animated visualization of the Quicksort algorithm in action, with each step of the sorting process displayed.
- **Performance**: The average time taken for each run will give insights into the efficiency of different pivot strategies.
- **Graph**: Excel graphs should be generated showing the relationship between time taken, input size and pivot strategy.

8. Answer each of the following question using 200 words and attach screen-shots to support your analysis.

1. What is the relationship between the size of the input (n) and the average completion time for quicksort?

2. How does the choice of pivot affect the performance of the quicksort algorithm for different values of n?

3. How consistent are the results across multiple runs, and what might account for any variability in the average completion time?

9. Conclusions

- The project will help demonstrate how Quicksort works through animation and will offer insights into its performance with different pivot strategies. By plotting the average time taken for each array size, the project will highlight how Quicksort scales with increasing input size and pivot choice.