Data Structures and Algorithms in C++ - Fundamental Concepts

#Part 1: Theory Questions

Introduction to DSA
1. What are the basic principles of Object-Oriented Programming in C++?
2. Explain the difference between arrays and vectors in C++.

Arrays and Memory
3. Explain the difference between stack and heap memory allocation in C++.
4. How does array indexing work in C++? What happens when we access an out-of-bounds index?

Time and Space Complexity
5. Analyze the space complexity difference between:
```
cpp_format
int arr[1000];  // Stack allocation
int* arr = new int[1000];  // Heap allocation
```

6. What is the time complexity of vector operations in C++:
   - push_back()
   - insert() at beginning
   - pop_back()

Part 2: Practical Questions

Arrays and Vectors
1. **Write a program demonstrating dynamic array manipulation:**
```
cpp_format
// Create a program that:
// 1. Takes array size from user
// 2. Dynamically allocates an array
// 3. Fills it with values
// 4. Finds sum and average
// 5. Properly deallocates memory

int main() {
    // Your code here
    return 0;
}
```

2. **Implement array rotation using different techniques:**
```
cpp_format
```

```
// Write a class with methods to:
// 1. Rotate array using temporary array
// 2. Rotate array using one by one
// 3. Rotate array using reversal algorithm

class ArrayRotation {
public:
    void leftRotate(int arr[], int size, int positions);
    void rightRotate(int arr[], int size, int positions);
    void printArray(int arr[], int size);
};
```

Searching

3. **Implement Linear Search with templates:**

cpp_format

```
// Create a template function that can search any data type
template<typename T>
int linearSearch(T arr[], int size, T key) {
    // Your code here
}
```

4. **Implement Binary Search with error handling:**

cpp_format

```
class BinarySearch {
public:
    // Return -1 if element not found
    // Throw exception if array is not sorted
    int search(int arr[], int size, int key);
private:
    bool isSorted(int arr[], int size);
};
```

Sorting

5. **Implement Selection Sort with comparison operator overloading:**

cpp_format

```
class Student {
    string name;
    int score;
public:
    // Constructor
    // Overload < operator for comparison
```

```
    // Getters and setters
};

void selectionSort(Student arr[], int size);
```

Requirements:
- Proper input validation
- Exception handling
- Performance measurement using `chrono` library
- Memory leak prevention
- Documentation using comments