Data Structures and Algorithms in C++ - Fundamental Concepts

Hitarth Patel
150096724046
Jensen Huang

#Part 1: Theory Questions

Introduction to DSA

1. What are the basic principles of Object-Oriented Programming in C++?
→

- Encapsulation : Bundling data and methods that operate on that data within a single unit (class).
- Abstraction : Hiding implementation details and showing only necessary information to the outside world.
- Inheritance : Creating a new class based on an existing class, inheriting its properties and behavior.
- Polymorphism : Ability of an object to take on multiple forms, depending on the context

2. Explain the difference between arrays and vectors in C++.
→

- Fixed Size: Arrays have a fixed size that must be specified at compile time. Vectors, on the other hand, can grow or shrink dynamically at runtime.
- Memory Allocation: Arrays are allocated memory on the stack or statically, whereas vectors allocate memory on the heap.
- Operations: Vectors provide more operations like push_back, insert, erase, etc., making them more flexible and easier to use.

Arrays and Memory

3. Explain the difference between stack and heap memory allocation in C++.
→

- Stack Allocation: Memory is allocated and deallocated automatically when a function is called and returns. Variables allocated on the stack have a fixed size and scope.
- Heap Allocation: Memory is allocated and deallocated manually using operators new and delete. Heap allocation allows for dynamic memory allocation, but it requires manual memory management.

4. How does array indexing work in C++? What happens when we access an out-of-bounds Index?
→

- Valid Indexing: Accessing an array element within its bounds (from index 0 to size - 1) is valid and returns the corresponding element.

- Out-of-Bounds Access: Accessing an array element outside its bounds (less than 0 or greater than or equal to size) results in undefined behavior.

Time and Space Complexity

5. Analyze the space complexity difference between:

```
int arr[1000]; // Stack allocation
int* arr = new int[1000]; // Heap allocation
```

→

- The space complexity of stack allocation is O(1) because the memory is allocated and deallocated automatically, and the size is fixed.
- The space complexity of heap allocation is O(n) because the memory is allocated dynamically, and the size can vary.

6. What is the time complexity of vector operations in C++:

```
push_back()
insert() at beginning
pop_back()
```

→

- push_back: Amortized O(1)
- insert at beginning: O(n)
- pop_back: O(1)

Part 2: Practical Questions

Arrays and Vectors

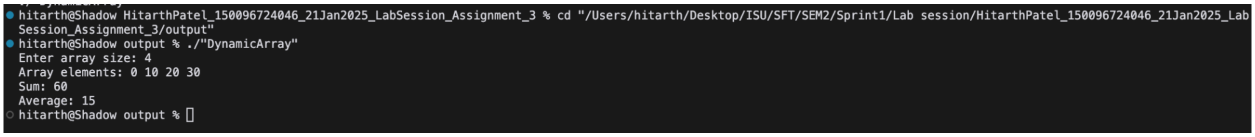
1. Write a program demonstrating dynamic array manipulation:

```
// 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;
}
```

→

```
#include <iostream>
```

```cpp
using namespace std;

int main() {
   int size;
   cout << "Enter array size: ";
   cin >> size;

   int* arr = new int[size];

   for (int i = 0; i < size; ++i) {
       arr[i] = i * 10;
   }

   int sum = 0;
   for (int i = 0; i < size; ++i) {
       sum += arr[i];
   }
   double average = static_cast<double>(sum) / size;

   cout << "Array elements: ";
   for (int i = 0; i < size; ++i) {
       cout << arr[i] << " ";
   }
   cout << "\nSum: " << sum << "\nAverage: " << average << endl;

   delete[] arr;

   return 0;}
```



```
● hitarth@Shadow HitarthPatel_150096724046_21Jan2025_LabSession_Assignment_3 % cd "/Users/hitarth/Desktop/ISU/SFT/SEM2/Sprint1/Lab session/HitarthPatel_150096724046_21Jan2025_Lab
  Session_Assignment_3/output"
● hitarth@Shadow output % ./"DynamicArray"
  Enter array size: 4
  Array elements: 0 10 20 30
  Sum: 60
  Average: 15
○ hitarth@Shadow output % []
```

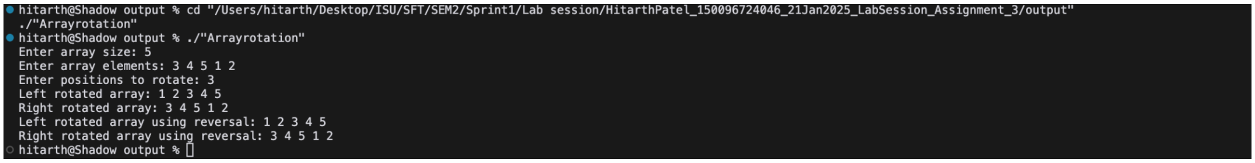
2. Implement array rotation using different techniques:

```cpp
// 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);
};
```

→

```
#include <iostream>
using namespace std;

class ArrayRotation {
public:
   void leftRotate(int arr[], int size, int positions) {
       int* temp = new int[positions];
       for (int i = 0; i < positions; ++i) {
           temp[i] = arr[i];
       }
       for (int i = positions; i < size; ++i) {
           arr[i - positions] = arr[i];
       }
       for (int i = 0; i < positions; ++i) {
           arr[size - positions + i] = temp[i];
       }
       delete[] temp;
   }

    void rightRotate(int arr[], int size, int positions) {
       for (int i = 0; i < positions; ++i) {
           int temp = arr[size - 1];
           for (int j = size - 1; j > 0; --j) {
               arr[j] = arr[j - 1];
           }
           arr[0] = temp;
       }
   }

   void leftRotateReversal(int arr[], int size, int positions) {
       reverse(arr, 0, positions - 1);
       reverse(arr, positions, size - 1);
       reverse(arr, 0, size - 1);
   }

   void rightRotateReversal(int arr[], int size, int positions) {
       reverse(arr, 0, size - 1);
```

```
        reverse(arr, 0, positions - 1);
        reverse(arr, positions, size - 1);
    }

    void printArray(int arr[], int size) {
        for (int i = 0; i < size; ++i) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }

private:
    void reverse(int arr[], int start, int end) {
        while (start < end) {
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }
};

int main() {
    int size;
    cout << "Enter array size: ";
    cin >> size;

    int* arr = new int[size];

    cout << "Enter array elements: ";
    for (int i = 0; i < size; ++i) {
        cin >> arr[i];
    }

    int positions;
    cout << "Enter positions to rotate: ";
    cin >> positions;

    ArrayRotation rotation;
    rotation.leftRotate(arr, size, positions);
    cout << "Left rotated array: ";
    rotation.printArray(arr, size);

    rotation.rightRotate(arr, size, positions);
    cout << "Right rotated array: ";
    rotation.printArray(arr, size);

    rotation.leftRotateReversal(arr, size, positions);
    cout << "Left rotated array using reversal: ";
    rotation.printArray(arr, size);

    rotation.rightRotateReversal(arr, size, positions);
    cout << "Right rotated array using reversal: ";
    rotation.printArray(arr, size);

    delete[] arr;

    return 0;
}
```



```
hitarth@Shadow output % cd "/Users/hitarth/Desktop/ISU/SFT/SEM2/Sprint1/Lab session/HitarthPatel_150096724046_21Jan2025_LabSession_Assignment_3/output"
./"Arrayrotation"
hitarth@Shadow output % ./"Arrayrotation"
Enter array size: 5
Enter array elements: 3 4 5 1 2
Enter positions to rotate: 3
Left rotated array: 1 2 3 4 5
Right rotated array: 3 4 5 1 2
Left rotated array using reversal: 1 2 3 4 5
Right rotated array using reversal: 3 4 5 1 2
hitarth@Shadow output % 
```

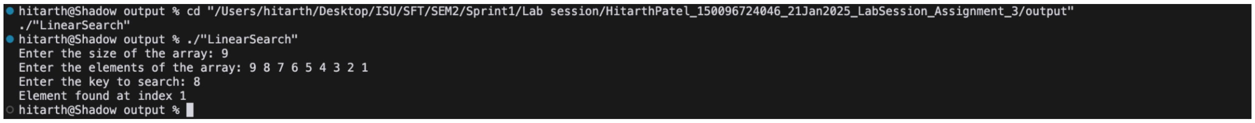
Searching

3. Implement Linear Search with templates:

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

→

```
#include <iostream>
using namespace std;

template <typename T>
int linearSearch(T arr[], int size, T key) {
    for (int i = 0; i < size; ++i) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}

int main() {
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;

    int* arr = new int[size];
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < size; ++i) {
        cin >> arr[i];
    }

    int key;
    cout << "Enter the key to search: ";
    cin >> key;

    int result = linearSearch(arr, size, key);
    if (result != -1) {
        cout << "Element found at index " << result << endl;
    } else {
        cout << "Element not found" << endl;
    }

    delete[] arr;

    return 0;
}
```



```
● hitarth@Shadow output % cd "/Users/hitarth/Desktop/ISU/SFT/SEM2/Sprint1/Lab session/HitarthPatel_150096724046_21Jan2025_LabSession_Assignment_3/output"
  ./"LinearSearch"
● hitarth@Shadow output % ./"LinearSearch"
  Enter the size of the array: 9
  Enter the elements of the array: 9 8 7 6 5 4 3 2 1
  Enter the key to search: 8
  Element found at index 1
○ hitarth@Shadow output % █
```

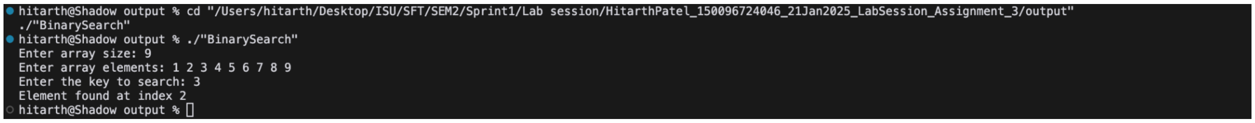
4. Implement Binary Search with error handling:

```
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);};
```

→

```
#include <iostream>
#include <stdexcept>
using namespace std;

class BinarySearch {
public:
   int search(int arr[], int size, int key) {
       if (!isSorted(arr, size)) {
           throw invalid_argument("Array is not sorted");
       }

       int low = 0;
       int high = size - 1;
       while (low <= high) {
           int mid = low + (high - low) / 2;
           if (arr[mid] == key) {
               return mid;
           } else if (arr[mid] < key) {
               low = mid + 1;
           } else {
               high = mid - 1;
           }
       }
       return -1;
   }

private:
   bool isSorted(int arr[], int size) {
       for (int i = 0; i < size - 1; ++i) {
           if (arr[i] > arr[i + 1]) {
               return false;
           }
       }
       return true;
   }
};
```

```
int main() {
   int size;
   cout << "Enter array size: ";
   cin >> size;

   int* arr = new int[size];
   cout << "Enter array elements: ";
   for (int i = 0; i < size; ++i) {
       cin >> arr[i];
   }

   int key;
   cout << "Enter the key to search: ";
   cin >> key;

   BinarySearch search;
   try {
       int result = search.search(arr, size, key);                 if (result != -1) {
           cout << "Element found at index " << result << std::endl;
       } else {
           cout << "Element not found" << std::endl;
       }
   } catch (const std::exception& e) {
       cerr << "Error: " << e.what() << std::endl;
   }

   delete[] arr;

   return 0;
}
```



```
● hitarth@Shadow output % cd "/Users/hitarth/Desktop/ISU/SFT/SEM2/Sprint1/Lab session/HitarthPatel_150096724046_21Jan2025_LabSession_Assignment_3/output"
  ./"BinarySearch"
● hitarth@Shadow output % ./"BinarySearch"
  Enter array size: 9
  Enter array elements: 1 2 3 4 5 6 7 8 9
  Enter the key to search: 3
  Element found at index 2
○ hitarth@Shadow output % ▯
```

Sorting

5. Implement Selection Sort with comparison operator overloading:

```
class Student {
string name;
int score;
public:
```

```
// Constructor
// Overload < operator for comparison

// Getters and setters
};
void selectionSort(Student arr[], int size);
```

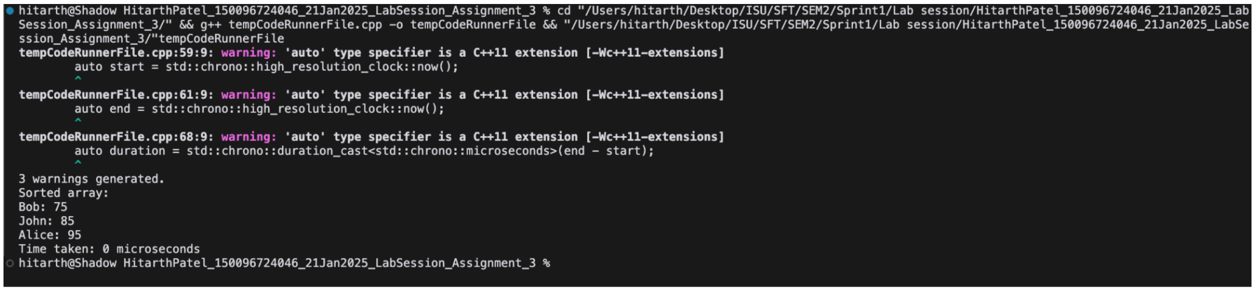
Requirements:

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

→

```
#include <iostream>
#include <chrono>
#include <stdexcept>

class Student {
private:
   std::string name;
   int score;

public:
   Student(std::string name, int score) : name(name), score(score) {}

   bool operator<(const Student& other) const {
       return score < other.score;
   }

   std::string getName() const {
       return name;
   }

   int getScore() const {
       return score;
   }

   void setName(const std::string& name) {
       this->name = name;
   }

   void setScore(int score) {
       this->score = score;
   }
};

void selectionSort(Student arr[], int size) {
   if (arr == nullptr || size <= 0) {
       throw std::invalid_argument("Invalid input");
   }

   for (int i = 0; i < size - 1; ++i) {
       int minIndex = i;
       for (int j = i + 1; j < size; ++j) {
           if (arr[j] < arr[minIndex]) {
               minIndex = j;
           }
       }
       if (minIndex != i) {
           Student temp = arr[i];
           arr[i] = arr[minIndex];
           arr[minIndex] = temp;
       }
   }
}

int main() {
   try {
       Student students[] = {Student("John", 85), Student("Alice", 95), Student("Bob",
75)};
       int size = sizeof(students) / sizeof(students[0]);

       auto start = std::chrono::high_resolution_clock::now();
       selectionSort(students, size);
       auto end = std::chrono::high_resolution_clock::now();

       std::cout << "Sorted array:" << std::endl;
       for (int i = 0; i < size; ++i) {
           std::cout << students[i].getName() << ": " << students[i].getScore() <<
std::endl;
       }

       auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end -
start);
```

```
        std::cout << "Time taken: " << duration.count() << " microseconds" <<
std::endl;
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```



```
hitarth@Shadow HitarthPatel_150096724046_21Jan2025_LabSession_Assignment_3 % cd "/Users/hitarth/Desktop/ISU/SFT/SEM2/Sprint1/Lab session/HitarthPatel_150096724046_21Jan2025_Lab
Session_Assignment_3/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/hitarth/Desktop/ISU/SFT/SEM2/Sprint1/Lab session/HitarthPatel_150096724046_21Jan2025_LabSe
ssion_Assignment_3/"tempCodeRunnerFile
tempCodeRunnerFile.cpp:59:9: warning: 'auto' type specifier is a C++11 extension [-Wc++11-extensions]
        auto start = std::chrono::high_resolution_clock::now();
        ^
tempCodeRunnerFile.cpp:61:9: warning: 'auto' type specifier is a C++11 extension [-Wc++11-extensions]
        auto end = std::chrono::high_resolution_clock::now();
        ^
tempCodeRunnerFile.cpp:68:9: warning: 'auto' type specifier is a C++11 extension [-Wc++11-extensions]
        auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - start);
        ^
3 warnings generated.
Sorted array:
Bob: 75
John: 85
Alice: 95
Time taken: 0 microseconds
hitarth@Shadow HitarthPatel_150096724046_21Jan2025_LabSession_Assignment_3 %
```

- Proper input validation: The selectionSort function checks for valid input, including a non-null array and a positive size.
- Exception handling: The code catches and handles exceptions that may occur during execution.
- Performance measurement: The code uses the chrono library to measure the time taken to perform the selection sort.
- Memory leak prevention: The code does not allocate memory dynamically, eliminating the risk of memory leaks.
- Documentation: The code includes comments to explain its functionality and purpose.