**ICS2105 - Data Structures and Algorithms Assignment**

## 1. Find the largest element in an integer array

**int findLargest(int arr[], int n) {  
 int max = arr[0];  
 for (int i = 1; i < n; ++i) {  
 if (arr[i] > max)  
 max = arr[i];  
 }  
 return max;  
}**

## 2. Shift all zeros to the end without changing the order of non-zero elements

**void shiftZerosToEnd(int arr[], int n) {  
 int index = 0;  
 for (int i = 0; i < n; ++i)  
 if (arr[i] != 0)  
 arr[index++] = arr[i];  
 while (index < n)  
 arr[index++] = 0;  
}**

## 3. Remove duplicates from a vector

**#include <vector>  
#include <unordered\_set>  
  
std::vector<int> removeDuplicates(std::vector<int>& vec) {  
 std::unordered\_set<int> seen;  
 std::vector<int> result;  
 for (int num : vec) {  
 if (seen.find(num) == seen.end()) {  
 seen.insert(num);  
 result.push\_back(num);  
 }  
 }  
 return result;  
}**

## 4. Reverse a singly linked list

**struct Node {  
 int data;  
 Node\* next;  
};  
  
Node\* reverseList(Node\* head) {  
 Node\* prev = nullptr;  
 Node\* curr = head;  
 while (curr != nullptr) {  
 Node\* nextNode = curr->next;  
 curr->next = prev;  
 prev = curr;  
 curr = nextNode;  
 }  
 return prev;  
}**

## 5. Implement binary search on a sorted array

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

## 6. Implement Depth-First Search (DFS) for a graph

**#include <iostream>  
#include <vector>  
using namespace std;  
  
void dfs(int v, vector<bool>& visited, vector<vector<int>>& adj) {  
 visited[v] = true;  
 cout << v << " ";  
 for (int u : adj[v]) {  
 if (!visited[u])  
 dfs(u, visited, adj);  
 }  
}**

## 7. Implement a queue using two stacks

**#include <stack>  
  
class QueueUsingStacks {  
 std::stack<int> s1, s2;  
  
public:  
 void enqueue(int x) {  
 s1.push(x);  
 }  
  
 int dequeue() {  
 if (s2.empty()) {  
 while (!s1.empty()) {  
 s2.push(s1.top());  
 s1.pop();  
 }  
 }  
 if (s2.empty()) return -1;  
 int front = s2.top();  
 s2.pop();  
 return front;  
 }  
};**

## 8. Reverse the elements of a queue

**#include <queue>  
#include <stack>  
  
void reverseQueue(std::queue<int>& q) {  
 std::stack<int> s;  
 while (!q.empty()) {  
 s.push(q.front());  
 q.pop();  
 }  
 while (!s.empty()) {  
 q.push(s.top());  
 s.pop();  
 }  
}**

## 9. How are arrays stored in memory and how does indexing work in constant time?

**Arrays are stored in contiguous memory. The address of arr[i] is:  
base\_address + (i \* size\_of\_element)  
This allows constant time (O(1)) access using indexing.**

## 10. Limitations of arrays and dynamic alternatives

**Limitations:  
- Fixed size  
- Inefficient insertions/deletions  
- Wasted memory if not fully used  
  
Alternative: std::vector in C++ – it resizes dynamically and manages memory automatically.**

## 11. Structure of a singly linked list and benefits

**Each node contains:  
- data  
- next pointer  
  
Advantages:  
- Dynamic size  
- Efficient insertions/deletions  
  
Compared to arrays:  
- Better for frequent inserts/removals  
- Slower for random access**

## 12. Binary Search vs Linear Search

|  |  |  |
| --- | --- | --- |
| **feature** | **Binary search** | **Linear search** |
| **Time complexity** | **O(log n)** | **O(n)** |
| **Data requirement** | **sorted** | **Unsorted** |
| **Speed on large Data** | **faster** | **slower** |
|  |  |  |

## 13. Graph representation in memory

**1. Adjacency Matrix: 2D array – fast but uses more memory  
2. Adjacency List: Array of vectors – memory efficient and commonly used**

## 14. FIFO property of queues + real-life analogy

**FIFO: First In, First Out  
Analogy: People waiting in line — the first person to join is the first to be served.**

## 15. Sparse matrix and linked list representation

**Sparse Matrix: Matrix with mostly zeros  
  
Representation:  
Store only non-zero elements as:  
struct SparseNode {  
 int row, col, value;  
 SparseNode\* next;  
};  
  
Example:  
Matrix:  
[0 0 5]  
[0 0 0]  
[7 0 0]  
Linked list:  
(0,2,5) → (2,0,7)**