DSA Assignment for MTE

1. What is a data structure? Explain different types with examples.

Answer:
A data structure is a way to organize, manage, and store data efficiently. Types:
- Linear: Arrays, Linked Lists, Stacks, Queues
- Non-linear: Trees, Graphs
- Hash-based: Hash Tables

Example:
Array: int arr[] = {1, 2, 3};
Linked List: Node with pointers
Stack: LIFO structure using push/pop
Queue: FIFO structure using enqueue/dequeue

2. Write a program to implement a stack using arrays. Include push, pop, and display operations.

C++ Program:

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

#define MAX 100
class Stack {
   int top;
   int arr[MAX];
public:
   Stack() { top = -1; }

   void push(int value) {
      if (top >= MAX - 1) {
         cout << "Stack Overflow\n";
```

```
            return;
        }
        arr[++top] = value;
    }

    void pop() {
        if (top < 0) {
            cout << "Stack Underflow\n";
            return;
        }
        top--;
    }

    void display() {
        for (int i = top; i >= 0; i--)
            cout << arr[i] << " ";
        cout << endl;
    }
};
```

3. Write a program to implement a queue using arrays. Include enqueue, dequeue, and display operations.

C++ Program:

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

#define SIZE 100
class Queue {
    int front, rear;
    int arr[SIZE];
public:
    Queue() {
        front = rear = -1;
    }
```

```
    void enqueue(int value) {
        if (rear == SIZE - 1) {
            cout << "Queue Overflow\n";
            return;
        }
        if (front == -1) front = 0;
        arr[++rear] = value;
    }

    void dequeue() {
        if (front == -1 || front > rear) {
            cout << "Queue Underflow\n";
            return;
        }
        front++;
    }

    void display() {
        for (int i = front; i <= rear; i++)
            cout << arr[i] << " ";
        cout << endl;
    }
};
```

4. Explain time and space complexity with examples.

Answer:
Time complexity: Time taken by an algorithm as a function of input size (n).
Example: for (int i = 0; i < n; i++) — O(n)

Space complexity: Extra memory required.
Example: Using an array of size n — O(n)

5. What is a linked list? Write a program to implement singly linked list with insert and display operations.

C++ Program:

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

struct Node {
   int data;
   Node* next;
};

void insertAtEnd(Node*& head, int value) {
   Node* newNode = new Node{value, nullptr};
   if (!head) {
      head = newNode;
      return;
   }
   Node* temp = head;
   while (temp->next) temp = temp->next;
   temp->next = newNode;
}

void displayList(Node* head) {
   while (head) {
      cout << head->data << " ";
      head = head->next;
   }
   cout << endl;
}
```

6. Write a program to implement a circular queue using arrays.

C++ Program:

```
#include <iostream>
```

```
using namespace std;

#define SIZE 5
class CircularQueue {
   int front, rear;
   int arr[SIZE];
public:
   CircularQueue() {
      front = rear = -1;
   }

   void enqueue(int value) {
      if ((rear + 1) % SIZE == front) {
         cout << "Queue Overflow\n";
         return;
      }
      if (front == -1) front = 0;
      rear = (rear + 1) % SIZE;
      arr[rear] = value;
   }

   void dequeue() {
      if (front == -1) {
         cout << "Queue Underflow\n";
         return;
      }
      if (front == rear) {
         front = rear = -1;
      } else {
         front = (front + 1) % SIZE;
      }
   }

   void display() {
      if (front == -1) {
```

```
            cout << "Queue is empty\n";
            return;
        }
        int i = front;
        while (true) {
            cout << arr[i] << " ";
            if (i == rear) break;
            i = (i + 1) % SIZE;
        }
        cout << endl;
    }
};
```

7. Write a program to implement stack using linked list.

C++ Program:

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

struct StackNode {
    int data;
    StackNode* next;
};

class StackLL {
    StackNode* top;
public:
    StackLL() { top = nullptr; }

    void push(int value) {
        StackNode* newNode = new StackNode{value, top};
        top = newNode;
    }

    void pop() {
```

```
        if (!top) {
            cout << "Stack Underflow\n";
            return;
        }
        StackNode* temp = top;
        top = top->next;
        delete temp;
    }

    void display() {
        StackNode* temp = top;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
```

8. Write a program to implement queue using linked list.

C++ Program:

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

struct QueueNode {
    int data;
    QueueNode* next;
};

class QueueLL {
    QueueNode* front;
    QueueNode* rear;
public:
    QueueLL() {
```

```
        front = rear = nullptr;
    }

    void enqueue(int value) {
        QueueNode* newNode = new QueueNode{value, nullptr};
        if (!rear) {
            front = rear = newNode;
            return;
        }
        rear->next = newNode;
        rear = newNode;
    }

    void dequeue() {
        if (!front) {
            cout << "Queue Underflow\n";
            return;
        }
        QueueNode* temp = front;
        front = front->next;
        if (!front) rear = nullptr;
        delete temp;
    }

    void display() {
        QueueNode* temp = front;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
```

9. What is recursion? Write a recursive function to calculate factorial of a number.

Answer:
Recursion is when a function calls itself to solve a problem in smaller subproblems.

C++ Program:

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

int factorial(int n) {
   if (n <= 1) return 1;
   return n * factorial(n - 1);
}
```

10. Write a program to reverse a linked list.

C++ Program:

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

struct ListNode {
   int data;
   ListNode* next;
};

void insertNode(ListNode*& head, int value) {
   ListNode* newNode = new ListNode{value, nullptr};
   newNode->next = head;
   head = newNode;
}

void reverseLinkedList(ListNode*& head) {
   ListNode* prev = nullptr;
```

```
    ListNode* curr = head;
    while (curr) {
        ListNode* nextNode = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextNode;
    }
    head = prev;
}

void printLinkedList(ListNode* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
```

11. Find two numbers in a sorted array that add up to a target. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use two-pointer technique:
1. Start with two pointers: one at the beginning, one at the end.
2. If sum is less than target, move the left pointer.
3. If sum is greater, move the right pointer.

C++ Program:

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

```
pair<int, int> findTwoSum(vector<int>& arr, int target) {
   int left = 0, right = arr.size() - 1;
   while (left < right) {
      int sum = arr[left] + arr[right];
      if (sum == target) return {left, right};
      else if (sum < target) left++;
      else right--;
   }
   return {-1, -1};
}

int main() {
   vector<int> arr = {1, 2, 3, 4, 6};
   int target = 7;
   auto result = findTwoSum(arr, target);
   cout << "Indices: " << result.first << ", " << result.second <<
endl;
   return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(1)

12. Rearrange numbers into the lexicographically next greater permutation. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
1. Find the first decreasing element from the right.
2. Swap with the next larger element to the right.
3. Reverse the right portion of the array.

C++ Program:

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

void nextPermutation(vector<int>& nums) {
   int i = nums.size() - 2;
   while (i >= 0 && nums[i] >= nums[i + 1]) i--;
   if (i >= 0) {
      int j = nums.size() - 1;
      while (nums[j] <= nums[i]) j--;
      swap(nums[i], nums[j]);
   }
   reverse(nums.begin() + i + 1, nums.end());
}

int main() {
   vector<int> nums = {1, 2, 3};
   nextPermutation(nums);
   for (int n : nums) cout << n << " ";
   return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(1)

13. How to merge two sorted linked lists into one sorted list. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use a dummy node and iterate while comparing nodes from both lists.

C++ Program:

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

struct Node {
   int data;
   Node* next;
   Node(int x) : data(x), next(nullptr) {}
};

Node* mergeLists(Node* l1, Node* l2) {
   Node dummy(0);
   Node* tail = &dummy;
   while (l1 && l2) {
      if (l1->data < l2->data) {
         tail->next = l1;
         l1 = l1->next;
      } else {
         tail->next = l2;
         l2 = l2->next;
      }
      tail = tail->next;
   }
   tail->next = l1 ? l1 : l2;
   return dummy.next;
}
```

Time Complexity: O(n + m)
Space Complexity: O(1)

14. Find the median of two sorted arrays using binary search. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use binary search on the smaller array to partition both arrays.

C++ Program:

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

double findMedianSortedArrays(vector<int>& A, vector<int>&
B) {
   if (A.size() > B.size()) return findMedianSortedArrays(B, A);
   int x = A.size(), y = B.size();
   int low = 0, high = x;
   while (low <= high) {
      int partitionX = (low + high) / 2;
      int partitionY = (x + y + 1) / 2 - partitionX;

      int maxLeftX = (partitionX == 0) ? INT_MIN : A[partitionX -
1];
      int minRightX = (partitionX == x) ? INT_MAX : A[partitionX];

      int maxLeftY = (partitionY == 0) ? INT_MIN : B[partitionY -
1];
      int minRightY = (partitionY == y) ? INT_MAX : B[partitionY];

      if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
         if ((x + y) % 2 == 0)
            return (max(maxLeftX, maxLeftY) + min(minRightX,
minRightY)) / 2.0;
         else
            return max(maxLeftX, maxLeftY);
      } else if (maxLeftX > minRightY) {
```

```
            high = partitionX - 1;
        } else {
            low = partitionX + 1;
        }
    }
    return -1;
}
```

Time Complexity: O(log(min(n, m)))
Space Complexity: O(1)

15. Find the k-th smallest element in a sorted matrix. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use a min-heap or binary search over the matrix value range.

C++ Program (Binary Search):

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

int countLessEqual(vector<vector<int>>& matrix, int mid, int n)
{
    int count = 0, j = n - 1;
    for (int i = 0; i < n; i++) {
        while (j >= 0 && matrix[i][j] > mid) j--;
        count += (j + 1);
    }
    return count;
}
```

```
int kthSmallest(vector<vector<int>>& matrix, int k) {
   int n = matrix.size();
   int low = matrix[0][0], high = matrix[n - 1][n - 1];
   while (low < high) {
      int mid = (low + high) / 2;
      if (countLessEqual(matrix, mid, n) < k)
         low = mid + 1;
      else
         high = mid;
   }
   return low;
}
```

Time Complexity: O(n * log(max - min))
Space Complexity: O(1)

16. Find the majority element in an array that appears more than n/2 times. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use Boyer-Moore Voting Algorithm.

C++ Program:

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

int majorityElement(vector<int>& nums) {
   int count = 0, candidate = 0;
   for (int num : nums) {
      if (count == 0) candidate = num;
      count += (num == candidate) ? 1 : -1;
```

```
    }
    return candidate;
}

int main() {
    vector<int> nums = {2, 2, 1, 1, 2, 2, 2};
    cout << "Majority Element: " << majorityElement(nums) <<
endl;
    return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(1)

17. Calculate how much water can be trapped between the bars of a histogram. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use two-pointer approach with left and right max heights.

C++ Program:

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

int trap(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int leftMax = 0, rightMax = 0, water = 0;
    while (left < right) {
        if (height[left] < height[right]) {
            height[left] >= leftMax ? (leftMax = height[left]) : (water
+= leftMax - height[left]);
```

```
            left++;
        } else {
            height[right] >= rightMax ? (rightMax = height[right]) :
(water += rightMax - height[right]);
            right--;
        }
    }
    return water;
}

int main() {
    vector<int> height = {0,1,0,2,1,0,1,3,2,1,2,1};
    cout << "Trapped water: " << trap(height) << endl;
    return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(1)

18. Find the maximum XOR of two numbers in an array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use a trie or greedy bit-mask strategy.

C++ Program (using bit masking):

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

int findMaximumXOR(vector<int>& nums) {
```

```
    int maxXor = 0, mask = 0;
    for (int i = 31; i >= 0; i--) {
        mask |= (1 << i);
        unordered_set<int> s;
        for (int num : nums) {
            s.insert(num & mask);
        }
        int candidate = maxXor | (1 << i);
        for (int prefix : s) {
            if (s.count(prefix ^ candidate)) {
                maxXor = candidate;
                break;
            }
        }
    }
    return maxXor;
}

int main() {
    vector<int> nums = {3, 10, 5, 25, 2, 8};
    cout << "Maximum XOR: " << findMaximumXOR(nums) <<
endl;
    return 0;
}
```

Time Complexity: O(n * 32)
Space Complexity: O(n)

19. How to find the maximum product subarray. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:

Track both max and min products at each position.

C++ Program:

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

int maxProduct(vector<int>& nums) {
   int currMax = nums[0], currMin = nums[0], result = nums[0];
   for (int i = 1; i < nums.size(); i++) {
      if (nums[i] < 0) swap(currMax, currMin);
      currMax = max(nums[i], currMax * nums[i]);
      currMin = min(nums[i], currMin * nums[i]);
      result = max(result, currMax);
   }
   return result;
}

int main() {
   vector<int> nums = {2, 3, -2, 4};
   cout << "Maximum product subarray: " << maxProduct(nums)
<< endl;
   return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(1)

20. Count all numbers with unique digits for a given number of digits. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use combinatorics: For n = 0 → 1, n = 1 → 10, and beyond, multiply possibilities.

C++ Program:

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

int countNumbersWithUniqueDigits(int n) {
   if (n == 0) return 1;
   int res = 10, uniqueDigits = 9, available = 9;
   while (n-- > 1 && available > 0) {
      uniqueDigits *= available;
      res += uniqueDigits;
      available--;
   }
   return res;
}

int main() {
   int n = 2;
   cout << "Count of unique digit numbers: " <<
countNumbersWithUniqueDigits(n) << endl;
   return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(1)

21. How to count the number of 1s in the binary representation of numbers from 0 to n. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use dynamic programming:
- For i > 0, bits[i] = bits[i / 2] + (i % 2)

C++ Program:

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

vector<int> countBits(int n) {
   vector<int> bits(n + 1, 0);
   for (int i = 1; i <= n; i++) {
      bits[i] = bits[i >> 1] + (i & 1);
   }
   return bits;
}

int main() {
   int n = 5;
   vector<int> result = countBits(n);
   for (int i = 0; i <= n; i++) {
      cout << "Number of 1s in " << i << ": " << result[i] << endl;
   }
   return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(n)

22. How to check if a number is a power of two using bit manipulation. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
If n > 0 and (n & (n - 1)) == 0, then it’s a power of two.

C++ Program:

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

bool isPowerOfTwo(int n) {
   return n > 0 && (n & (n - 1)) == 0;
}

int main() {
   int num = 16;
   cout << num << (isPowerOfTwo(num) ? " is" : " is not") << " a
power of two." << endl;
   return 0;
}
```

Time Complexity: O(1)
Space Complexity: O(1)

23. How to find the maximum XOR of two numbers in an array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

(Repeated from Q18)

Algorithm:
Use greedy + hashset for bitmask checking.

C++ Program:

```
#include <iostream>
```

```
#include <vector>
#include <unordered_set>
using namespace std;

int findMaximumXOR(vector<int>& nums) {
   int maxXor = 0, mask = 0;
   for (int i = 31; i >= 0; i--) {
      mask |= (1 << i);
      unordered_set<int> s;
      for (int num : nums) s.insert(num & mask);
      int candidate = maxXor | (1 << i);
      for (int prefix : s) {
         if (s.count(prefix ^ candidate)) {
            maxXor = candidate;
            break;
         }
      }
   }
   return maxXor;
}

int main() {
   vector<int> nums = {3, 10, 5, 25, 2, 8};
   cout << "Maximum XOR: " << findMaximumXOR(nums) <<
endl;
   return 0;
}
```

Time Complexity: O(n * 32)
Space Complexity: O(n)

24. Explain the concept of bit manipulation and its advantages in algorithm design.

Answer:

Bit manipulation is a technique where binary operators are used directly on bits to perform operations such as AND, OR, XOR, NOT, shifts, and masks.

Advantages:
- Extremely fast (low-level operations)
- Useful in space-efficient algorithms
- Helps solve problems involving toggling states, counting bits, subsets, masks, etc.

Examples:
- Power of two check
- Counting set bits
- Swapping variables without temp

25. Solve the problem of finding the next greater element for each element in an array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use a stack to track decreasing elements from right to left.

C++ Program:

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

vector<int> nextGreaterElements(vector<int>& nums) {
    vector<int> res(nums.size(), -1);
    stack<int> st;
    for (int i = nums.size() - 1; i >= 0; i--) {
```

```
        while (!st.empty() && st.top() <= nums[i]) st.pop();
        if (!st.empty()) res[i] = st.top();
        st.push(nums[i]);
    }
    return res;
}

int main() {
    vector<int> nums = {4, 5, 2, 10, 8};
    vector<int> result = nextGreaterElements(nums);
    for (int val : result) cout << val << " ";
    return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(n)

26. Remove the n-th node from the end of a singly linked list. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use two pointers: move one pointer n steps ahead, then move both until the end.

C++ Program:

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
```

```
};

ListNode* removeNthFromEnd(ListNode* head, int n) {
   ListNode dummy(0);
   dummy.next = head;
   ListNode *first = &dummy, *second = &dummy;
   for (int i = 0; i <= n; i++) first = first->next;
   while (first) {
      first = first->next;
      second = second->next;
   }
   second->next = second->next->next;
   return dummy.next;
}
```

Time Complexity: O(n)
Space Complexity: O(1)

27. Find the node where two singly linked lists intersect. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Use two pointers. Switch lists when a pointer reaches the end.

C++ Program:

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

struct ListNode {
   int val;
   ListNode* next;
   ListNode(int x) : val(x), next(nullptr) {}
```

```
};

ListNode* getIntersectionNode(ListNode* headA, ListNode*
headB) {
   ListNode* a = headA;
   ListNode* b = headB;
   while (a != b) {
      a = a ? a->next : headB;
      b = b ? b->next : headA;
   }
   return a;
}
```

Time Complexity: O(n + m)
Space Complexity: O(1)

28. Implement two stacks in a single array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Start stack1 from 0 and stack2 from end. Move inwards.

C++ Program:

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

class TwoStacks {
   int* arr;
   int top1, top2, size;
public:
   TwoStacks(int n) {
      size = n;
```

```
        arr = new int[n];
        top1 = -1;
        top2 = n;
    }

    void push1(int x) {
        if (top1 + 1 < top2) arr[++top1] = x;
    }

    void push2(int x) {
        if (top1 + 1 < top2) arr[--top2] = x;
    }

    int pop1() {
        return (top1 >= 0) ? arr[top1--] : -1;
    }

    int pop2() {
        return (top2 < size) ? arr[top2++] : -1;
    }
};

int main() {
    TwoStacks ts(10);
    ts.push1(5); ts.push2(10);
    cout << ts.pop1() << " " << ts.pop2();
    return 0;
}
```

Time Complexity: O(1)
Space Complexity: O(n)

29. Write a program to check if an integer is a palindrome without converting it to a string. Write its algorithm, program.

Find its time and space complexities. Explain with suitable example.

Answer:

Algorithm:
Reverse half of the number and compare.

C++ Program:

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

bool isPalindrome(int x) {
   if (x < 0 || (x % 10 == 0 && x != 0)) return false;
   int rev = 0;
   while (x > rev) {
      rev = rev * 10 + x % 10;
      x /= 10;
   }
   return x == rev || x == rev / 10;
}

int main() {
   int num = 121;
   cout << num << (isPalindrome(num) ? " is" : " is not") << " a
palindrome." << endl;
   return 0;
}
```

Time Complexity: $O(\log_{10}(n))$
Space Complexity: O(1)

30. Explain the concept of linked lists and their applications in algorithm design.

Answer:

A linked list is a linear data structure where elements are stored in nodes and each node points to the next.

Types:
- Singly Linked List
- Doubly Linked List
- Circular Linked List

Applications:
- Dynamic memory allocation
- Efficient insert/delete at head/middle
- Implementing stacks, queues, graphs, and hash tables
- Used in file systems and memory management

31. Use a deque to find the maximum in every sliding window of size K. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
Maintain a deque that stores indices of elements in a way such that the front always contains the index of the largest element for the current window. Remove elements that are out of the current window or smaller than the current element.

C++ Program:

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

vector<int> maxSlidingWindow(vector<int>& nums, int k) {
   deque<int> dq;
   vector<int> res;
```

```
    for (int i = 0; i < nums.size(); i++) {
        if (!dq.empty() && dq.front() == i - k)
            dq.pop_front();
        while (!dq.empty() && nums[dq.back()] < nums[i])
            dq.pop_back();
        dq.push_back(i);
        if (i >= k - 1)
            res.push_back(nums[dq.front()]);
    }
    return res;
}
```

Time Complexity: O(n)
Space Complexity: O(k)

32. How to find the largest rectangle that can be formed in a histogram. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
Use a stack to store indices of the bars. For each bar, calculate the area with the height of the bar at the top of the stack as the smallest bar.

C++ Program:

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

int largestRectangleArea(vector<int>& heights) {
    stack<int> s;
    heights.push_back(0);
    int maxArea = 0;
    for (int i = 0; i < heights.size(); ++i) {
```

```
        while (!s.empty() && heights[i] < heights[s.top()]) {
            int h = heights[s.top()];
            s.pop();
            int w = s.empty() ? i : i - s.top() - 1;
            maxArea = max(maxArea, h * w);
        }
        s.push(i);
    }
    return maxArea;
}
```

Time Complexity: O(n)
Space Complexity: O(n)

33. Explain the sliding window technique and its applications in array problems.

Answer:
The sliding window technique is used to optimize problems that involve linear sequences like arrays or strings. It uses two pointers to create a window and slide it over the data to reduce time complexity.

Applications:
- Maximum/minimum sum of subarrays
- Longest substring with unique characters
- Count of anagrams in a string
- Subarrays with given sum

34. Solve the problem of finding the subarray sum equal to K using hashing. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:

Use a hashmap to store cumulative sum frequencies. For each sum, check if (sum - k) exists in the map to count valid subarrays.

C++ Program:

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

int subarraySum(vector<int>& nums, int k) {
    unordered_map<int, int> sumFreq;
    sumFreq[0] = 1;
    int sum = 0, count = 0;
    for (int num : nums) {
        sum += num;
        if (sumFreq.count(sum - k))
            count += sumFreq[sum - k];
        sumFreq[sum]++;
    }
    return count;
}
```

Time Complexity: O(n)
Space Complexity: O(n)

35. Find the k-most frequent elements in an array using a priority queue. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
Count the frequency of elements using a hashmap, then use a max-heap (priority queue) to get the top k frequent elements.

C++ Program:

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <queue>
using namespace std;

vector<int> topKFrequent(vector<int>& nums, int k) {
   unordered_map<int, int> freq;
   for (int num : nums) freq[num]++;
   priority_queue<pair<int, int>> pq;
   for (auto& [num, count] : freq)
      pq.push({count, num});
   vector<int> res;
   for (int i = 0; i < k; i++) {
      res.push_back(pq.top().second);
      pq.pop();
   }
   return res;
}
```

Time Complexity: O(n log n)
Space Complexity: O(n)

36. Generate all subsets of a given array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
Start with the empty subset. For each element, add it to all existing subsets and append those to the result.

C++ Program:

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

```
vector<vector<int>> subsets(vector<int>& nums) {
   vector<vector<int>> res = {{}};
   for (int num : nums) {
      int n = res.size();
      for (int i = 0; i < n; i++) {
         res.push_back(res[i]);
         res.back().push_back(num);
      }
   }
   return res;
}
```

Time Complexity: O(2^n)
Space Complexity: O(2^n)

37. Find all unique combinations of numbers that sum to a target. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
Use backtracking to explore all combinations that sum to the target, reusing elements.

C++ Program:

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

void helper(vector<int>& c, int target, vector<int>& curr, int idx,
vector<vector<int>>& res) {
   if (target == 0) {
      res.push_back(curr);
      return;
   }
```

```
    for (int i = idx; i < c.size(); ++i) {
        if (c[i] <= target) {
            curr.push_back(c[i]);
            helper(c, target - c[i], curr, i, res);
            curr.pop_back();
        }
    }
}

vector<vector<int>> combinationSum(vector<int>& candidates,
int target) {
    vector<vector<int>> res;
    vector<int> curr;
    helper(candidates, target, curr, 0, res);
    return res;
}
```

Time Complexity: O(2^target)
Space Complexity: O(target)

38. Generate all permutations of a given array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
Use backtracking and swap elements to generate all possible permutations.

C++ Program:

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

void permuteHelper(vector<int>& nums, int start,
vector<vector<int>>& res) {
```

```
    if (start == nums.size()) {
        res.push_back(nums);
        return;
    }
    for (int i = start; i < nums.size(); i++) {
        swap(nums[start], nums[i]);
        permuteHelper(nums, start + 1, res);
        swap(nums[start], nums[i]);
    }
}

vector<vector<int>> permute(vector<int>& nums) {
    vector<vector<int>> res;
    permuteHelper(nums, 0, res);
    return res;
}
```

Time Complexity: O(n!)
Space Complexity: O(n)

39. Explain the difference between subsets and permutations with examples.

Answer:
- Subsets are combinations where the order doesn't matter.
  Example: [1, 2] → [], [1], [2], [1, 2]
- Permutations are arrangements where the order matters.
  Example: [1, 2] → [1, 2], [2, 1]

40. Solve the problem of finding the element with maximum frequency in an array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:

Use a hashmap to count the occurrences of each element.
Track the maximum frequency while iterating.

C++ Program:

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

int maxFrequencyElement(vector<int>& nums) {
   unordered_map<int, int> freq;
   int maxFreq = 0, element = nums[0];
   for (int num : nums) {
      freq[num]++;
      if (freq[num] > maxFreq) {
         maxFreq = freq[num];
         element = num;
      }
   }
   return element;
}
```

Time Complexity: O(n)
Space Complexity: O(n)

41. Write a program to generate all valid parentheses combinations for a given number of pairs. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
Use backtracking by keeping count of open and close parentheses used. Only add open if open < n, and close if close < open.

C++ Program:

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

void generateParenthesisHelper(int open, int close, int n, string
current, vector<string>& result) {
   if (current.length() == n * 2) {
      result.push_back(current);
      return;
   }
   if (open < n)
      generateParenthesisHelper(open + 1, close, n, current + "(",
result);
   if (close < open)
      generateParenthesisHelper(open, close + 1, n, current + ")",
result);
}

vector<string> generateParentheses(int n) {
   vector<string> result;
   generateParenthesisHelper(0, 0, n, "", result);
   return result;
}
```

Time Complexity: O(2^n)
Space Complexity: O(n)

42. Solve the N-Queens problem and print all solutions. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
Use backtracking. Place a queen row by row, checking column, main diagonal, and anti-diagonal safety.

C++ Program:

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

bool isSafeQueen(vector<string>& board, int row, int col, int n)
{
   for (int i = 0; i < row; i++)
      if (board[i][col] == 'Q') return false;
   for (int i = row-1, j = col-1; i >= 0 && j >= 0; i--, j--)
      if (board[i][j] == 'Q') return false;
   for (int i = row-1, j = col+1; i >= 0 && j < n; i--, j++)
      if (board[i][j] == 'Q') return false;
   return true;
}

void solveNQueensHelper(int row, int n, vector<string>& board,
vector<vector<string>>& result) {
   if (row == n) {
      result.push_back(board);
      return;
   }
   for (int col = 0; col < n; col++) {
      if (isSafeQueen(board, row, col, n)) {
         board[row][col] = 'Q';
         solveNQueensHelper(row + 1, n, board, result);
         board[row][col] = '.';
      }
   }
}

vector<vector<string>> solveNQueens(int n) {
   vector<vector<string>> result;
   vector<string> board(n, string(n, '.'));
```

```
    solveNQueensHelper(0, n, board, result);
    return result;
}
```

Time Complexity: O(n!)
Space Complexity: O(n^2)

43. Explain the concept of backtracking with suitable examples.

Answer:
Backtracking is a technique to explore all possibilities recursively and undo changes (backtrack) when needed.

Examples:
- N-Queens problem
- Sudoku solver
- Generating permutations, combinations
- Word search in grid

44. Solve the problem of finding the shortest path in a binary matrix using BFS. Write its algorithm, program. Find its time and space complexities.

Algorithm:
Use BFS starting from (0, 0), moving in all 8 directions. Return the path length when reaching the bottom-right cell.

C++ Program:

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

int shortestPathBinaryMatrix(vector<vector<int>>& grid) {
```

```
    int n = grid.size();
    if (grid[0][0] || grid[n-1][n-1]) return -1;
    queue<pair<int, int>> q;
    q.push({0, 0});
    grid[0][0] = 1;

    vector<pair<int, int>> directions = {
        {1,0},{0,1},{-1,0},{0,-1},{1,1},{1,-1},{-1,1},{-1,-1}
    };

    while (!q.empty()) {
        auto [x, y] = q.front(); q.pop();
        int dist = grid[x][y];
        if (x == n-1 && y == n-1) return dist;
        for (auto [dx, dy] : directions) {
            int nx = x + dx, ny = y + dy;
            if (nx >= 0 && ny >= 0 && nx < n && ny < n &&
grid[nx][ny] == 0) {
                q.push({nx, ny});
                grid[nx][ny] = dist + 1;
            }
        }
    }
    return -1;
}
```

Time Complexity: O(n^2)
Space Complexity: O(n^2)

45. Implement a trie and perform insert, search, and startsWith operations.

C++ Program:

```
#include <iostream>
#include <unordered_map>
```

```
using namespace std;

class TrieNode {
public:
   bool isWord;
   unordered_map<char, TrieNode*> children;
   TrieNode() : isWord(false) {}
};

class Trie {
   TrieNode* root;
public:
   Trie() {
      root = new TrieNode();
   }

   void insert(string word) {
      TrieNode* node = root;
      for (char c : word) {
         if (!node->children[c])
            node->children[c] = new TrieNode();
         node = node->children[c];
      }
      node->isWord = true;
   }

   bool search(string word) {
      TrieNode* node = root;
      for (char c : word) {
         if (!node->children[c])
            return false;
         node = node->children[c];
      }
      return node->isWord;
   }
```

```
    bool startsWith(string prefix) {
        TrieNode* node = root;
        for (char c : prefix) {
            if (!node->children[c])
                return false;
            node = node->children[c];
        }
        return true;
    }
};
```

Time Complexity: O(m) per operation (m = length of word)
Space Complexity: O(n * m)

46. Solve the word search problem in a grid using backtracking. Write its algorithm, program. Find its time and space complexities.

Algorithm:
Start DFS from each cell, check 4 directions, mark visited using temporary change, backtrack.

C++ Program:

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

bool dfs(vector<vector<char>>& board, string word, int i, int j,
int idx) {
    if (idx == word.size()) return true;
    if (i < 0 || j < 0 || i >= board.size() || j >= board[0].size() ||
board[i][j] != word[idx]) return false;
    char temp = board[i][j];
    board[i][j] = '#';
```

```
    bool found = dfs(board, word, i+1, j, idx+1) ||
                 dfs(board, word, i-1, j, idx+1) ||
                 dfs(board, word, i, j+1, idx+1) ||
                 dfs(board, word, i, j-1, idx+1);
    board[i][j] = temp;
    return found;
}

bool exist(vector<vector<char>>& board, string word) {
    for (int i = 0; i < board.size(); i++)
        for (int j = 0; j < board[0].size(); j++)
            if (dfs(board, word, i, j, 0)) return true;
    return false;
}
```

Time Complexity: O(m * n * 4^L)
Space Complexity: O(L)

47. Explain the differences between BFS and DFS with examples.

Answer:
- BFS explores level by level using a queue.
  - Example: Shortest path in unweighted graph.
- DFS explores depth-wise using a stack or recursion.
  - Example: Topological sort, cycle detection.

48. Implement depth-first search (DFS) in a graph. Write its algorithm and program.

C++ Program:

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

```
void dfs(int node, vector<vector<int>>& adj, vector<bool>&
visited) {
   visited[node] = true;
   cout << node << " ";
   for (int neighbor : adj[node])
      if (!visited[neighbor])
         dfs(neighbor, adj, visited);
}

int main() {
   int n = 5;
   vector<vector<int>> adj(n);
   adj[0] = {1, 2};
   adj[1] = {0, 3};
   adj[2] = {0, 4};
   adj[3] = {1};
   adj[4] = {2};

   vector<bool> visited(n, false);
   dfs(0, adj, visited);
   return 0;
}
```

Time Complexity: O(V + E)
Space Complexity: O(V)

49. Implement breadth-first search (BFS) in a graph. Write its algorithm and program.

C++ Program:

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

```
void bfs(int start, vector<vector<int>>& adj, vector<bool>&
visited) {
    queue<int> q;
    q.push(start);
    visited[start] = true;

    while (!q.empty()) {
        int node = q.front(); q.pop();
        cout << node << " ";
        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
}

int main() {
    int n = 5;
    vector<vector<int>> adj(n);
    adj[0] = {1, 2};
    adj[1] = {0, 3};
    adj[2] = {0, 4};
    adj[3] = {1};
    adj[4] = {2};

    vector<bool> visited(n, false);
    bfs(0, adj, visited);
    return 0;
}
```

Time Complexity: O(V + E)
Space Complexity: O(V)

50. What is topological sorting? How is it implemented using DFS?

Answer:
Topological sorting is a linear ordering of vertices such that for every directed edge u → v, u comes before v.

Implementation (DFS based):
- Perform DFS, push node to stack after visiting all descendants.
- Reverse stack to get topological order.