ALL Complexities

String Matching:

1. **Naive**:
- Time: O(m*n)
- Space: O(1)
2. **Rabin Karp**:
- Time: Average case O(n+m), worst case O(n*m)
- Space: O(m)
3. **Knuth-Morris-Pratt (KMP)**:
- Time: O(n+m)
- Space: O(m)
4. **Finite Automata**:
- Time: O(n)
- Space: O(m*Σ), where Σ is the size of the alphabet

Divide n Conquer:

1. **Optimal Merge Pattern**:
- Time: O(n log n)
- Space: O(n)

Graphs:

1. **Breadth-First Search (BFS)**:
- Time: O(V+E)
- Space: O(V)
2. **Depth-First Search (DFS)**:
- Time: O(V+E)
- Space: O(V)
3. **Prim's Algorithm**:
- Time: $O(V^2)$, O(E log V) for a binary heap
- Space: O(V)
4. **Dijkstra's Algorithm**:
- Time: $O(V^2)$, O(E log V) for a binary heap
- Space: O(V)

Greedy:

1. **Job Scheduling**:
- Time: O(n log n)
- Space: O(n)
2. **Fractional Knapsack**:
- Time: O(n log n)
- Space: O(n)
3. **Huffman Coding**:
- Time: O(n log n)
- Space: O(n)

Sorting:

1. **Insertion Sort**:
- Time: $O(n^2)$
- Space: O(1)
2. **Merge Sort**:
- Time: O(n log n)
- Space: O(n)
3. **Quick Sort**:
- Time: Average case O(n log n), worst case $O(n^2)$
- Space: O(log n)
4. **Counting Sort**:
- Time: O(n+k)
- Space: O(n+k)
5. **Bucket Sort**:
- Time: Average case O(n+k), worst case $O(n^2)$
- Space: O(n+k)
6. **Radix Sort**:
- Time: O(nk)
- Space: O(n+k)

Heap:

1. **Build-Heap**:
- Time: O(n)
- Space: O(1)
2. **Heapify**:
- Time: O(log n)

Space: O(1)

3. **Heap Sort**:
- Time: O(n log n)
- Space: O(1)

Dynamic Programming

1. **Rod Cutting**:
- Time: $O(n^2)$
- Space: O(n)
2. **Matrix Chain Multiplication**:
- Time: $O(n^3)$
- Space: $O(n^2)$
3. **Longest Common Subsequence (LCS)**:
- Time: O(n*m)
- Space: O(n*m)
4. **Optimal Binary Search Tree (OBST)**:
- Time: $O(n^3)$
- Space: $O(n^2)$
5. **0/1 Knapsack**:
- Time: O(nW), where W is the capacity of the knapsack
- Space: O(nW)
6. **Bellman Ford Algorithm**:
- Time: O(VE)