Recitation Material: Computational Complexity and Algorithm Analysis

Problem 1: Understanding Big-O Notation

Question: Prove that $f(n) = 3n^2 + 2n + 1$ is $O(n^2)$.

Problem 2: Analyzing Running Time

Question: Given the algorithm that runs in time $T(n) = 4n^3 + 3n^2 + 5$, show that it is $O(n^3)$.

Problem 3: Counting Operations

Question: Count the number of basic operations in the following code snippet:

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        sum += i * j;
    }
}
```

Problem 4: Comparing Functions

Question: Show that $f(n) = n^2$ is better than $g(n) = n^3$ asymptotically.

Problem 5: Understanding Big-Omega

Question: Prove that $f(n) = 2n^2$ is $\Omega(n^2)$.

Problem 6: Big-Theta Notation

Question: Prove that $f(n) = 5n^2 + 3n$ is $\Theta(n^2)$.

Problem 7: Analyzing the Complexity of Linear Search

Question: Consider a linear search algorithm that searches for an element in an unsorted array of length n. The algorithm works as follows:

```
function linearSearch(arr, target):
    for i = 0 to arr.length - 1:
        if arr[i] == target:
            return i
    return -1
```

Analyze the time complexity of this algorithm.

Problem 8: Complexity of Matrix Addition

Question: Consider the algorithm for adding two $n \times n$ matrices:

```
function addMatrices(A, B):
    C = new matrix of size n x n
    for i = 0 to n-1:
        for j = 0 to n-1:
            C[i][j] = A[i][j] + B[i][j]
    return C
```

Analyze the computational complexity of this algorithm.

Problem 9: Complexity Analysis of Merge Sort

Question: Merge Sort is a divide-and-conquer algorithm that works as follows:

1. Divide the array into two halves.
2. Recursively sort each half.
3. Merge the two sorted halves.

The code for Merge Sort can be written as:

```
function mergeSort(arr):
    if arr.length <= 1:
        return arr
    mid = arr.length / 2
    left = mergeSort(arr[0:mid])
    right = mergeSort(arr[mid:])
    return merge(left, right)
```

Prove that the time complexity of the Merge Sort algorithm is $O(n \log n)$.

Problem 10: Complexity of Training a Perceptron

Question: A single-layer perceptron is trained using the following basic algorithm:

1. Initialize the weights to small random values.
2. For each training sample (x_i, y_i):
   - Compute the output $\hat{y} = \text{sign}(w \cdot x_i)$.
   - Update the weights if the prediction is incorrect:

     $$w = w + \eta(y_i - \hat{y})x_i$$

     where η is the learning rate.

Given a dataset with m samples, each with d features, analyze the computational complexity of training the perceptron.

Problem 11: Complexity Analysis of k-Means Clustering

Question: The k-Means clustering algorithm works as follows:

1. Initialize k cluster centroids randomly.
2. Repeat until convergence:
   - Assign each of the m data points to the nearest cluster (distance computation).
   - Update the cluster centroids based on the assigned points.

Given k clusters, m data points, and d features, analyze the computational complexity of k-Means clustering.

Problem 12: Complexity of Decision Tree Training

Question: A decision tree is trained as follows:

1. For each node, evaluate all d features to find the best split based on a criterion (e.g., Gini impurity, information gain).
2. Split the node based on the best feature and recursively repeat for the child nodes.
3. Stop when a stopping condition is met (e.g., maximum depth or minimum samples).

Given m samples and d features, analyze the computational complexity of training a decision tree.