Final Exam Sample Assignment

Question 1: Given the following Binary Search Tree…

```
      1
     / \
    2   3
   / \   \
  4   5   6
```

What is the pre-order traversal of the tree?

a) 1, 2, 4, 5, 6, 3
b 1, 2, 4, 5, 3, 6
c) 1, 2, 3, 4, 5, 6
d) 1, 3, 6, 2, 4, 5
Answer: B

What is the post-order traversal of the tree?:

a) 5, 4, 2, 6, 3, 1
b) 4, 5, 2, 3, 6, 1
c) 4, 5, 2, 6, 3, 1
d) 5, 4, 6, 2, 3, 1
Answer: C

What is the in-order traversal of the tree?:

a) 1, 2, 3, 4, 5, 6
b) 2, 1, 3, 4, 6, 5
c) 1, 2, 3, 4, 6, 5
d) 1, 2, 3, 5, 4, 6
Answer: A

Choose the correct direction for each traversal:

Pre-Order:

a) Left, Right, Visit

b) Left, Visit, Right

c) Right, Visit, Left

d) Visit, Left, Right

Answer: D

Post-Order:

a) Left, Right, Visit

b) Left, Visit, Right

c) Right, Visit, Left

d) Right, Left, Visit

Answer: A

In-Order:

a) Visit, Left, Right

b) Left, Visit, Right

c) Left, Left, Right

d) Right, Visit, Left

Answer: B

Question 2:

- **List all the different/main Big O notation types**
- **Explain how they work, their uniqueness, etc.**

Answer:
Big O notation is a way to describe the efficiency or complexity of an algorithm in terms of its input size. It helps us analyze how an algorithm's runtime or memory usage scales as the input size grows. There are several types of Big O notation, each representing a different level of algorithmic efficiency. Here are the most common types:
Big O-Types:

● O(1)	Constant Time Complexity
● O(log n)	Logarithmic Time Complexity
● O(n)	Linear Time Complexity
● O(n log n)	Linearithmic Time Complexity
● O(n^2)	Quadratic Time Complexity
● O(n^k)	Polynomial Time Complexity
● O(2^n)	Exponential Time Complexity
● O(n!)	Factorial Time Complexity

How they work, their uniqueness, etc.:

- O(1) - Constant Time Complexity:
  An algorithm is said to have constant time complexity if its execution time or memory usage remains constant, regardless of the input size. It's the most efficient complexity since the algorithm's performance doesn't change as the input grows.
- O(log n) - Logarithmic Time Complexity:
  Algorithms with logarithmic complexity have runtimes that increase logarithmically as the input size grows. This means that as the input size increases, the increase in execution time becomes slower. Binary search algorithms are a classic example of algorithms with logarithmic time complexity.

- O(n) - Linear Time Complexity:
  Algorithms with linear time complexity have runtimes that grow linearly with the size of the input. As the input size increases by a certain factor, the execution time also increases by a proportional factor. Simple array traversal is an example of linear time complexity.
- O(n log n) - Linearithmic Time Complexity:
  Algorithms with linearithmic complexity have runtimes that are a product of linear and logarithmic growth. This complexity often appears in efficient sorting algorithms like Merge Sort and Heap Sort.
- O(n^2) - Quadratic Time Complexity:
  Quadratic time complexity means that the runtime grows quadratically with the input size. Nested loops that iterate over the input are common causes of quadratic complexity. Examples include selection sort and insertion sort.
- O(n^k) - Polynomial Time Complexity:
  Polynomial time complexities, where k is a constant greater than 1, indicate that the runtime grows as a polynomial function of the input size. This category includes cubic time (O(n^3)), quartic time (O(n^4)), and so on.
- O(2^n) - Exponential Time Complexity:
  Algorithms with exponential complexity have runtimes that double with each additional element in the input. These algorithms quickly become impractical for larger inputs due to their exponential growth.
- O(n!) - Factorial Time Complexity:
  Algorithms with factorial complexity have runtimes that are proportional to the factorial of the input size. These algorithms are extremely inefficient and are generally not usable for any practical inputs.

When analyzing algorithmic complexity, it's common to focus on the worst-case scenario, as this gives an upper bound on how the algorithm will perform under any circumstances.