4th QUESTION EXPLAINATION ASSIGNMENT-2

THE FUNCTIONS I HAVE USED FOR THE 4th QUESTION(IN THIS EXPLAINATION I HAVE NOT EXPLAINED ABOUT THE MAIN):

`makenode(int val)`: Creates a new node with the given value, sets its left and right pointers to NULL, and returns a pointer to the new node.

- `makequeue(int capacity)`: Creates a new queue with the given capacity, allocates memory for the queue, and returns a pointer to the new queue.
- `isfull(queue *head)`: Returns 1 if the queue is full (i.e. the counter is equal to the capacity), and 0 otherwise.
- `isempty(queue *head)`: Returns 1 if the queue is empty (i.e. the rear pointer is -1), and 0 otherwise.
- `enqueue(queue *head, int val)`: Adds a new element to the rear of the queue (if the queue is not full) and increments the rear pointer and counter accordingly.
- `dequeue(queue *head)`: Removes and returns the element at the front of the queue (if the queue is not empty) and increments the front pointer and decrements the counter accordingly.
- `ListToBST(Node *root, int val)`: Inserts a new node with the given value into a binary search tree rooted at `root`, and returns a pointer to the root of the modified tree.
- `Modifiedvalue(Node *root,long long int*a,queue *q,long long int *sum)`: Modifies the values of the nodes in a binary search tree by replacing each node's value with the sum of all values less than or equal to that node's value. The modified tree is returned.

- `totalsum(Node*head)`: Calculates and returns the total sum of all the nodes in a binary search tree.
- `printatgivennode(Node*head, int i,int pos)`: Prints the value of the node at position `i` in a binary tree, where the root node is at position 1 and each child node is at a position one greater than its parent. The `pos` parameter keeps track of the current position while traversing the tree.
- `height(Node*head)`: Calculates and returns the height of a binary tree.
- `printlevel(Node*head,long long int n)`: Prints the values of all the nodes in a binary tree level by level, starting with the root node and moving down one level at a time.

Time complexity:

1. Building the binary search tree using the given list of numbers takes O(n*log n) time complexity on average. In the worst case, it can take O(n^2) time complexity if the tree is skewed.
2. Enqueuing and dequeuing nodes from the queue takes O(n) time complexity in the worst case.
3. Traversing the tree to modify the node values and printing the nodes at each level takes O(n^2) time complexity in the worst case, if the tree is skewed.
4. Calculating the total sum of node values takes O(n) time complexity.

Therefore, the overall time complexity of the program is O(n^2) in the worst case.

Space complexity:

1. Creating the binary search tree takes O(n) space complexity.
2. Creating the queue takes O(1) space complexity.
3. The maximum number of nodes that can be stored in the queue at any given time is n, so the space complexity of the queue is O(n).
4. The recursive function Modifyiedvalue() uses the call stack, which can take up to O(n) space complexity.
5. Therefore, the overall space complexity of the program is O(n).

EXPLAINATION OF MAIN FUNCTION:
It starts by declaring a pointer `head` to the root of the binary search tree and initializing it to `NULL`. It then reads an integer `n` from the input, which represents the number of nodes in the tree.

After that, it reads `n` integers from the input and inserts them into the binary search tree using the `ListToBST()` function. This function takes the root of the tree and the current value as arguments and returns a pointer to the modified tree with the new node inserted.

Next, it creates a queue using the `makequeue()` function to perform a level order traversal of the tree. It then initializes the variable `sum` to zero and calls the `Modifiedvalue()` function, which modifies the values of the nodes of the binary search tree as per the given problem statement. The function takes the root of the tree, the array `a`, the queue `q`, and the variable `sum` as arguments, and returns a pointer to the modified tree.

After modifying the values of the nodes, the program calls the `printlevel()` function to print the nodes of the modified tree level by level. This function takes the root of the tree and the total number of nodes as arguments and prints the nodes of the tree level by level.

Finally, it calls the `totalsum()` function to calculate the sum of all the values in the modified binary search tree and prints it to the output.