DAA ASSIGNMENT

Name :- Fida khan **UID No:- 23BCS10345**
SECTION:- 621/A Submitted by:- Richa Dhiman mam

Subject:- DAA Assignment

1. Code to find the ignored successor in a Binary Search Tree with complexity analysis

Question:
Write a program to find the inorder successor of a given node in a Binary Search Tree (BST).

Answer:
Inorder successor of a node in a BST is the node with the smallest key greater than the key of the given node.

Algorithm:
1. If the node has a right subtree, the successor is the leftmost node in the right subtree.
2. If there's no right subtree, move up the tree until you find a node that is the left child of its parent; the parent is the successor.

Code (C++):

```
struct Node {
int data;
Node* left;
   Node* right;
};

Node* minValue(Node* node) {
Node* current = node;
   while (current && current->left != nullptr)
current = current->left;     return current;
}
```

```
Node* inorderSuccessor(Node* root, Node* target) {
Node* succ = nullptr;     while (root) {
      if (target->data < root->data) {
succ = root;           root = root-
>left;
      } else if (target->data > root->data)
         root = root->right;
else          break;
   }
   return succ;
}
```

Complexity Analysis:
Time Complexity: O(h), where h = height of BST
Space Complexity: O(1)

2. Reverse a linked list

Question:
Given the pointer to the head node of a linked list, reverse the list.

Code (C++):

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

Node* reverse(Node* head) {
   Node* prev = nullptr;
   Node* current = head;
Node* next = nullptr;
while (current != nullptr) {
next = current->next;
current->next = prev;
prev = current;       current
= next;
   }
   head = prev;
return head;
}
```

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

3. Heap Sort and Time Analysis

Question:
Sort a given set of elements using Heap Sort and determine the time required to sort.

Code (C++):

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

void heapify(int arr[], int n, int i) {
int largest = i;     int l = 2*i + 1;
int r = 2*i + 2;
   if (l < n && arr[l] > arr[largest])
      largest = l;
   if (r < n && arr[r] > arr[largest])
largest = r;     if (largest != i) {
swap(arr[i], arr[largest]);
heapify(arr, n, largest);
   }
}

void heapSort(int arr[], int n) {
for (int i = n/2 - 1; i >= 0; i--)
heapify(arr, n, i);     for (int i =
n-1; i > 0; i--) {
swap(arr[0], arr[i]);
heapify(arr, i, 0);
   }
}

int main() {
int n = 1000;
int arr[n];
```

```
   for (int i = 0; i < n; i++)
arr[i] = rand() % 1000;

   clock_t start = clock();
heapSort(arr, n);     clock_t
end = clock();

   double time_taken = double(end - start) / CLOCKS_PER_SEC;
cout << "Time taken: " << time_taken << " seconds" << endl;
return 0;
}
```

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

4. Postorder Traversal of a Binary Tree

Question:
Write the postOrder function to print the values of a binary tree in postorder traversal.

Code (C++):

```
struct Node {
int data;
Node* left;
   Node* right;
};

void postOrder(Node* root) {
if (root == nullptr) return;
postOrder(root->left);
postOrder(root->right);     cout
<< root->data << " ";
}
```

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

5. Queue Operations using Templates

Question:
Write code for enqueue, dequeue, isFull, and isEmpty operations using templates.

Code (C++):

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

template <class T> class
Queue {
   int front, rear, size;
T* arr; public:
Queue(int s) {
front = rear = -1;
      size = s;       arr =
new T[size];
   }
   bool isFull() { return rear == size - 1; }
   bool isEmpty() { return front == -1 || front > rear; }

   void enqueue(T x) {
      if (isFull()) {
         cout << "Queue is Full\n";
return;
      }
      if (front == -1) front = 0;
arr[++rear] = x;
   }

   void dequeue() {       if
(isEmpty()) {          cout <<
"Queue is Empty\n";
         return;
      }
      cout << "Dequeued: " << arr[front++] << endl;
   }
};
```

Complexity Analysis:

Enqueue: O(1) Dequeue:
O(1)
Space Complexity: O(n)