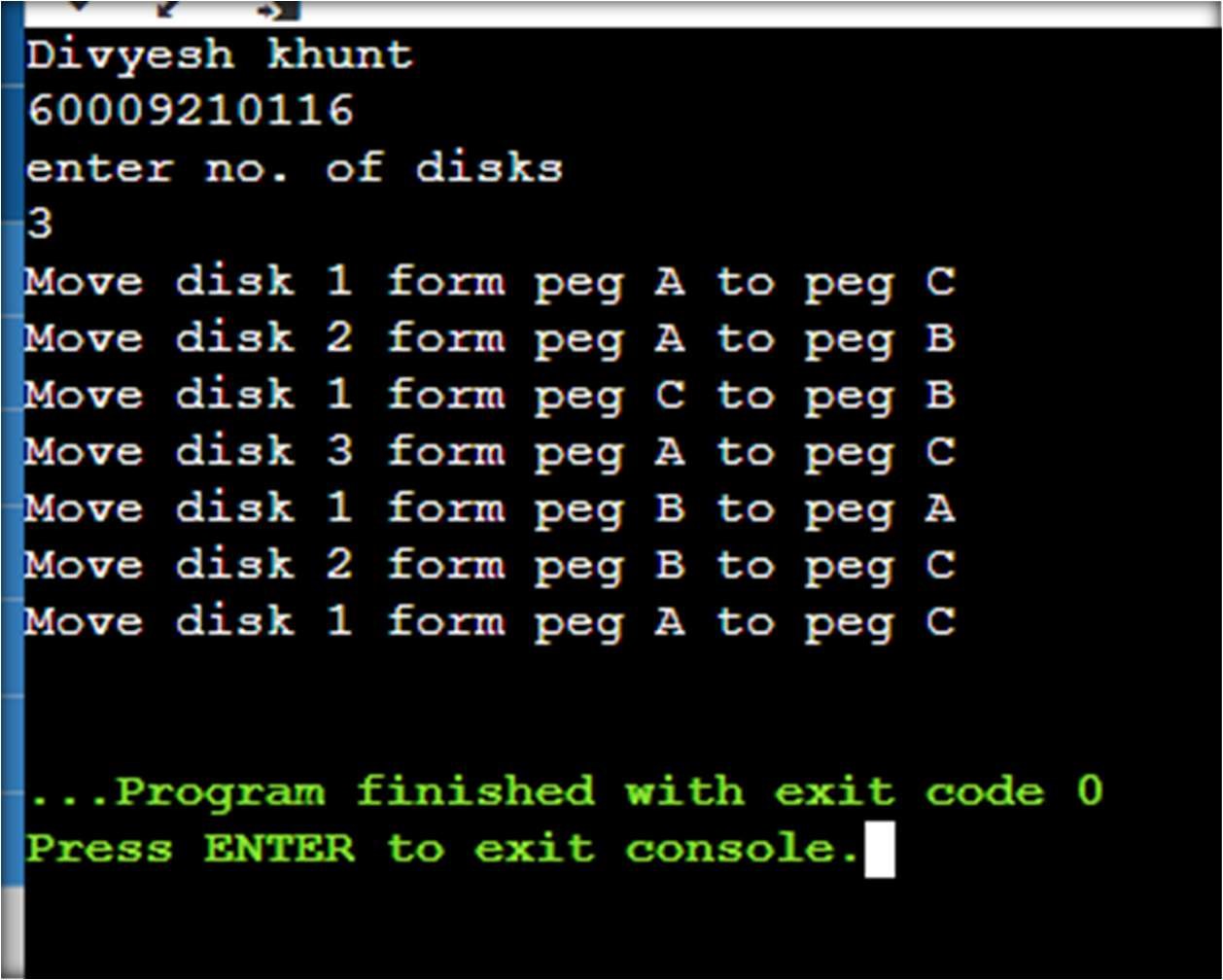
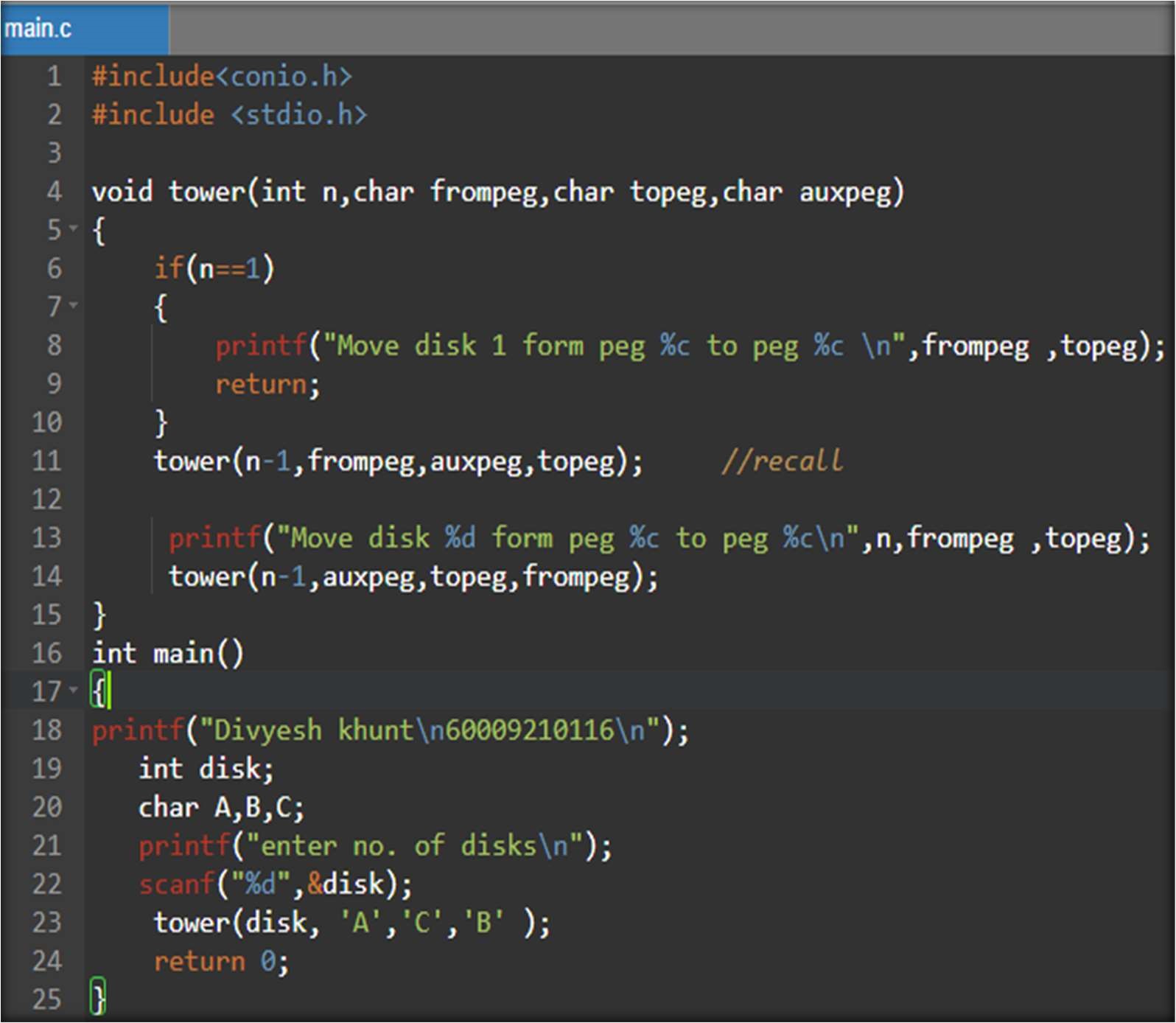
**DSA - Experiment 1**

**Name: Divyesh Khunt Sapid:60009210116 Batch:A/3**

|  |  |
| --- | --- |
|  | **AIM:** Implement and Analyse Tower of Hannoi    **THEORY**: Tower of Hanoi is a mathematical puzzle where we have three rods (A ,B ,C and ) and N disks. The smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack to another rod in the same manner as it was in the start.    RULES OF THE GAME:   * Only one disk can be moved at a time. * Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack. * No disk may be placed on top of a smaller disk.       **Time complexity**: O(2n)  T(n)=2T(n-1)+1  Backward substitution:  T(n–1) = 2T(n–2) + 2  T(n–2) = 2T(n–3) + 2  2T(n–2) = 22 T(n–3) + 4  22 T(n–3) = 23 T(n–3) + 8  T(1)=0+2n  General equation is T(n)=2n      **Space complexity:O(n)**  Space of recursive stack is of order n so space complexity is O(n) |

**CODE:**

O**UTPUT:**

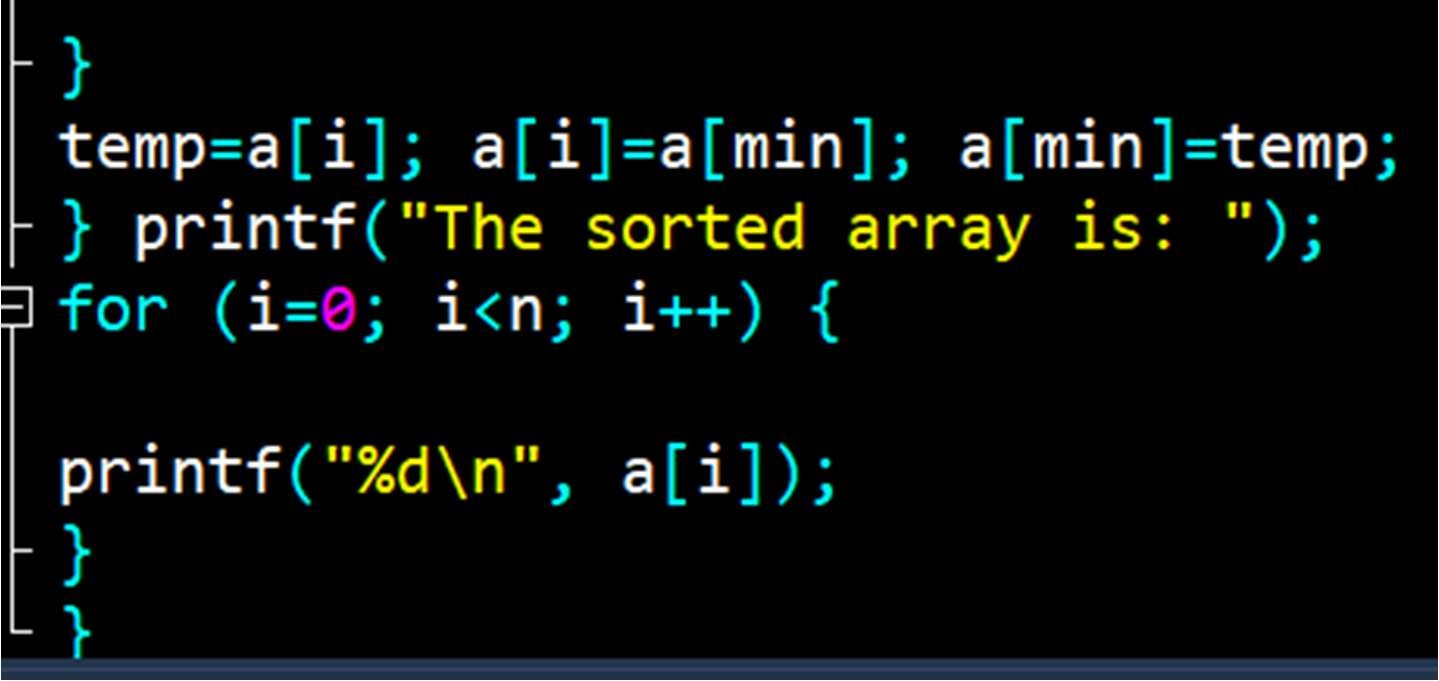
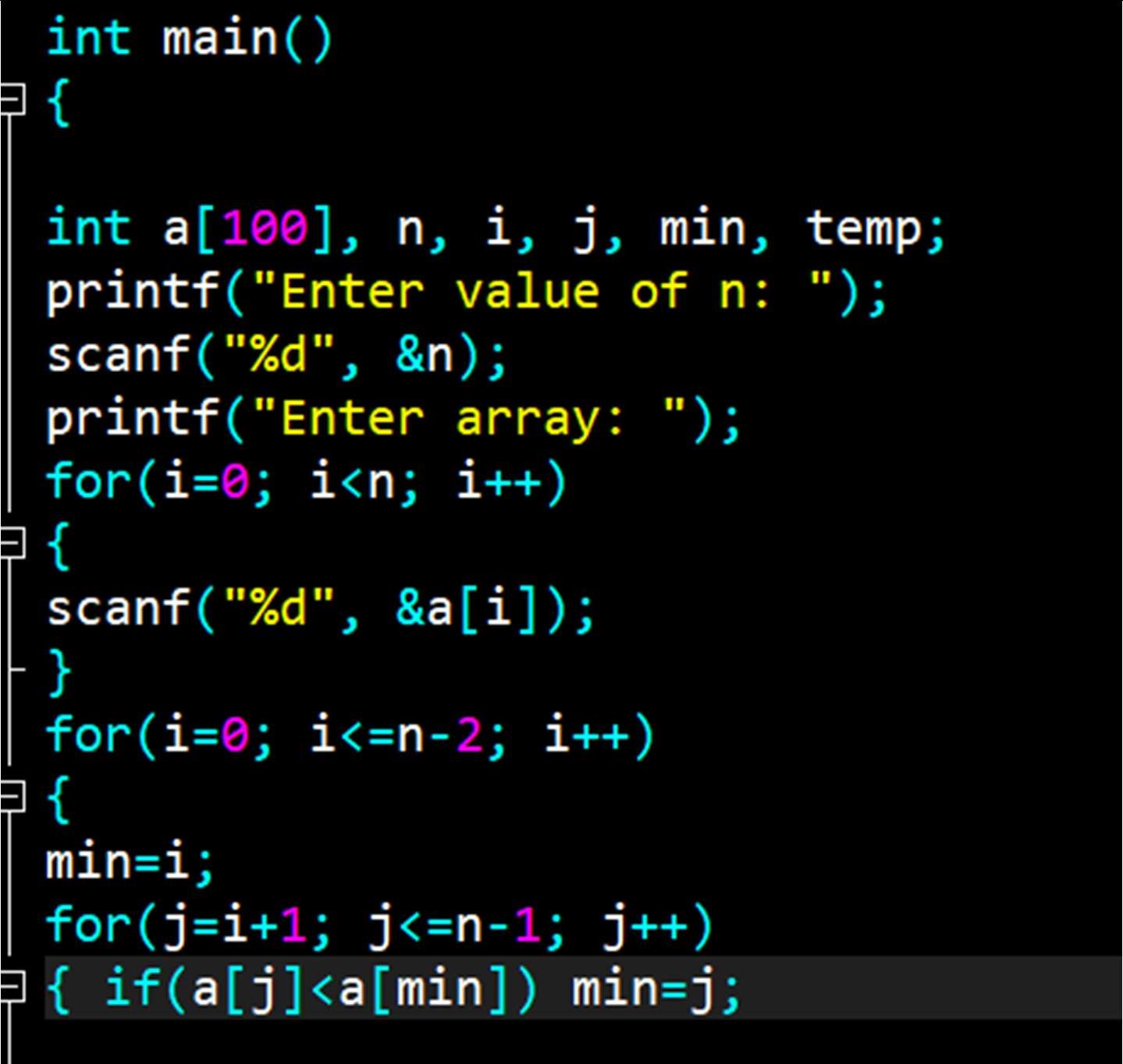


|  |  |
| --- | --- |
|  | **CONCLUSION:**  Thus the code of tower of hannoi was analysed and implemented.  The time complexity of tower of hannoi is 2n -1 |

Graphical user interface, text, application, email

Description automatically generated

**OUTPUTS:**



Text

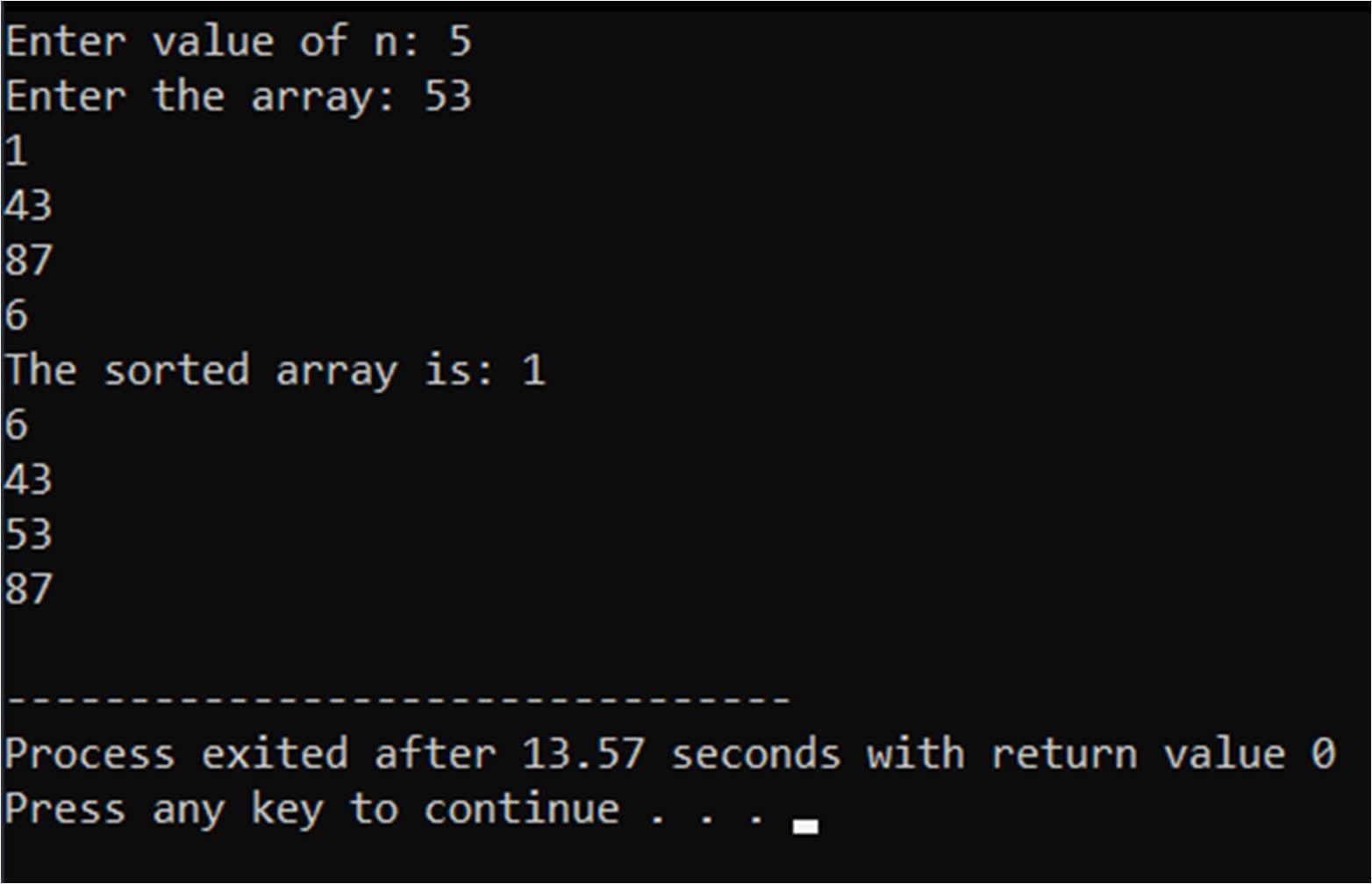
Description automatically generated

|  |
| --- |
| Insertion sort  **Theory:**  Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.  **Time Complexity :**  Since there is a while loop enclosed by the for loop the time complexity adds up to O(N2)=O(N\*N) CODE:  A screenshot of a computer  Description automatically generated with medium confidence |

OUTPUTS:

**Conclusion:**

Thus insertion and selection sort were implemented.



|  |  |
| --- | --- |
|  | **EXP 3**  **NAME: DIVYESH KHUNT SAPID:60009210116**  **Aim:** To implement and analyse Merge sort and Quick sort.  Merge Sort    **Theory:**  The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.  **Time Complexity:**  O (N log(N)), Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.  T(n) = 2T(n/2) + θ(n) CODE: |

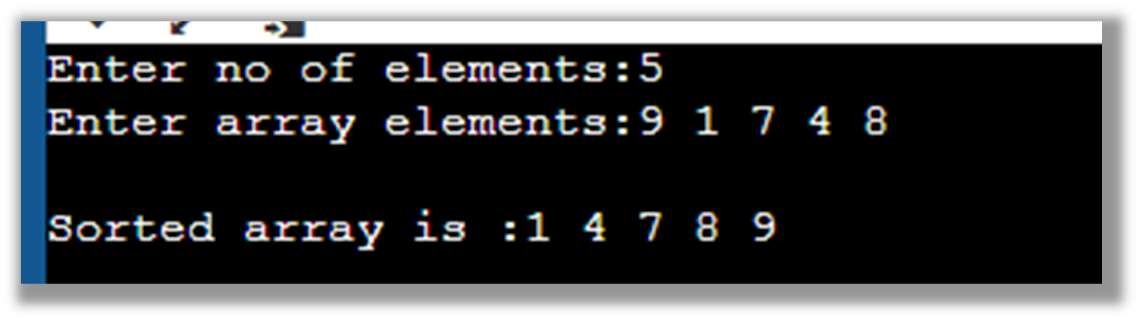
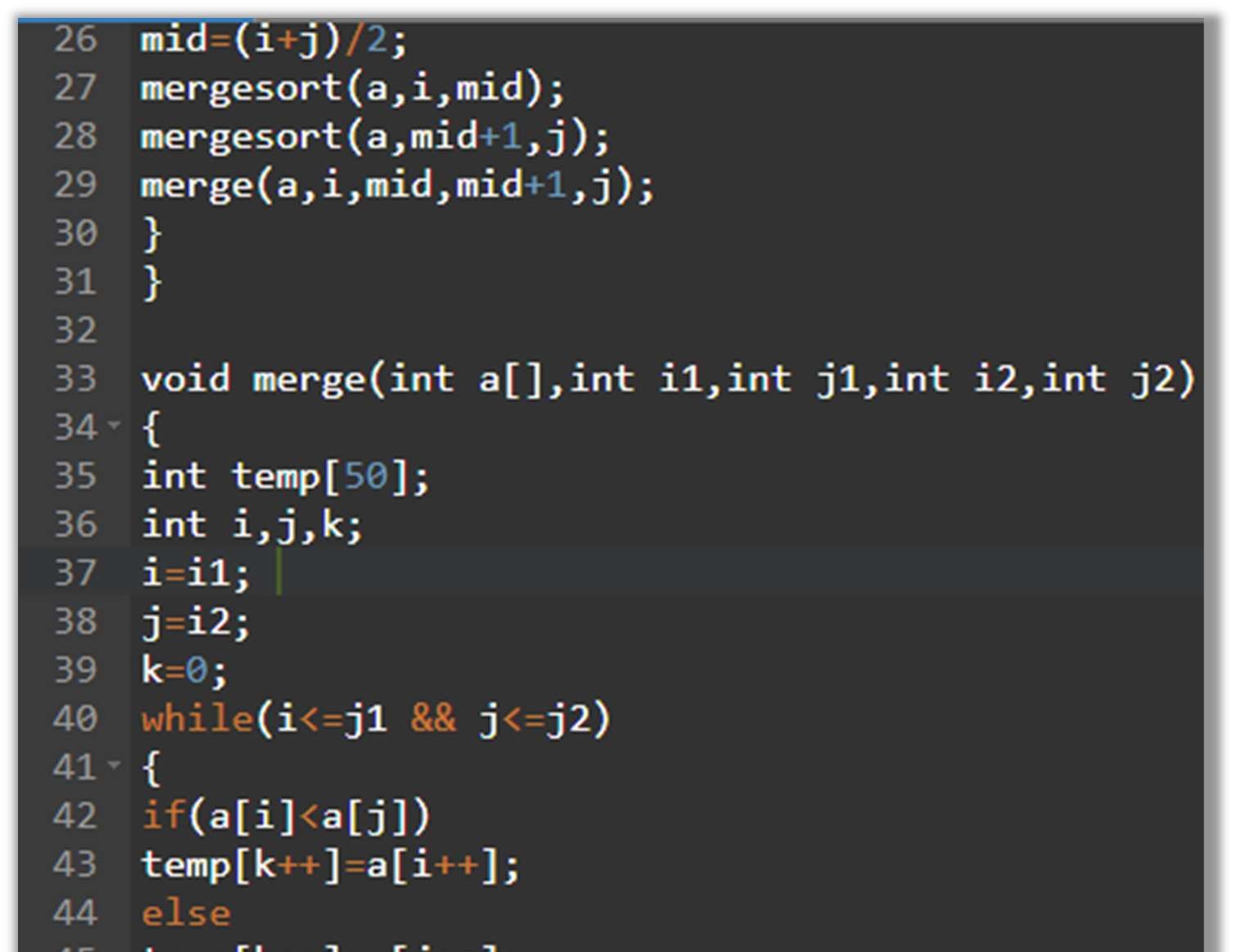
|  |  |
| --- | --- |
|  | OUTPUTS: |

|  |  |
| --- | --- |
|  | Quick sort  Theory:  Quicksort is a Divide and Conquer algorithm. It picks an Like Merge Sort element as a pivot and partitions the given array around the picked pivot. There are many different versions of quicksort that pick pivot in different ways.   1. Always pick the first element as a pivot. 2. Always pick the last element as a pivot (implemented below) 3. Pick a random element as a pivot. 4. Pick median as the pivot.   Time Complexity :  T(n) = T(k)+T(n-k-1)+O(n)Where k is no. of elements that are smaller than pivot.  CODE: |

OUTPUTS:

Conclusion:

Thus insertion and selection sort were implemented.



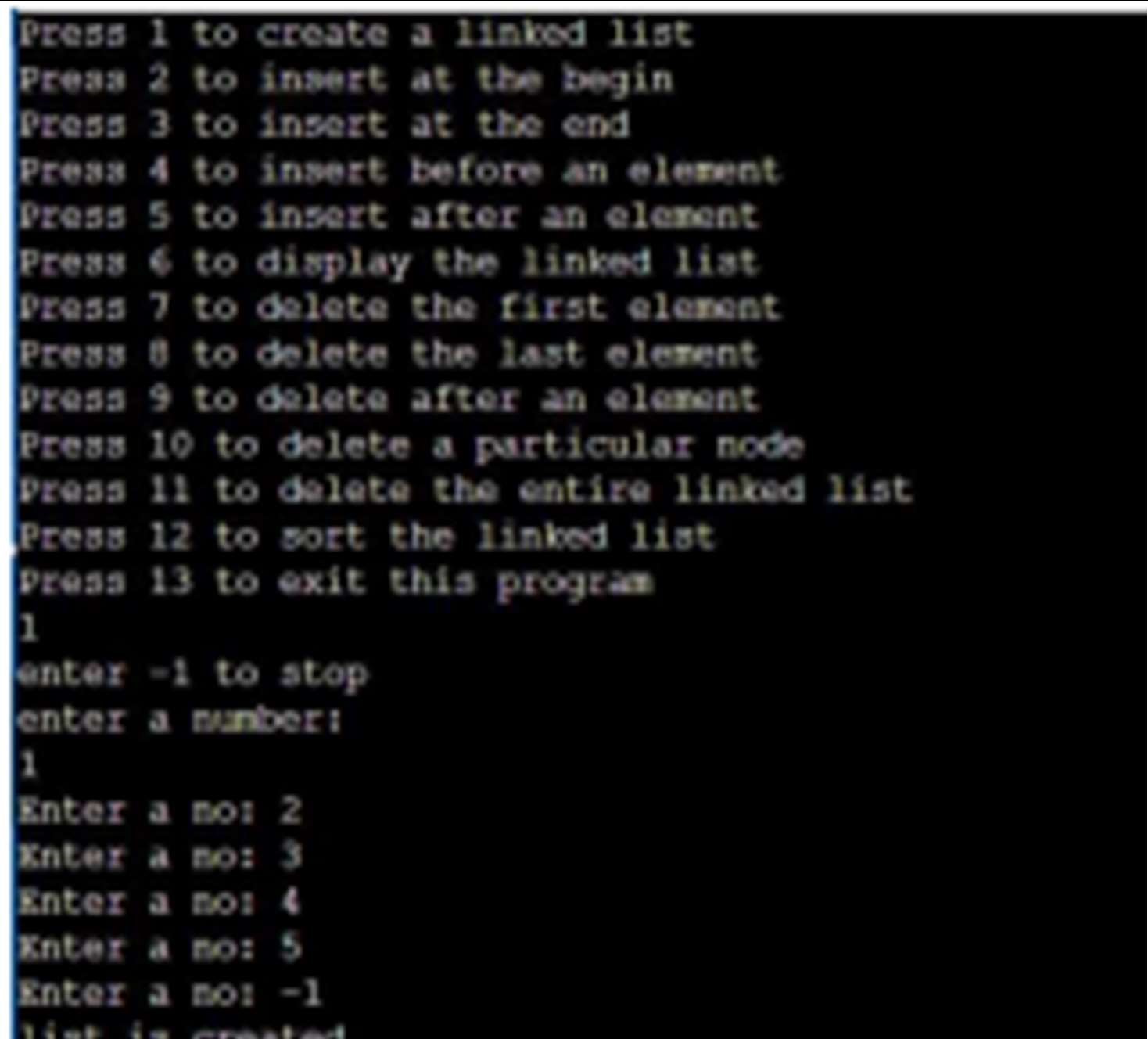
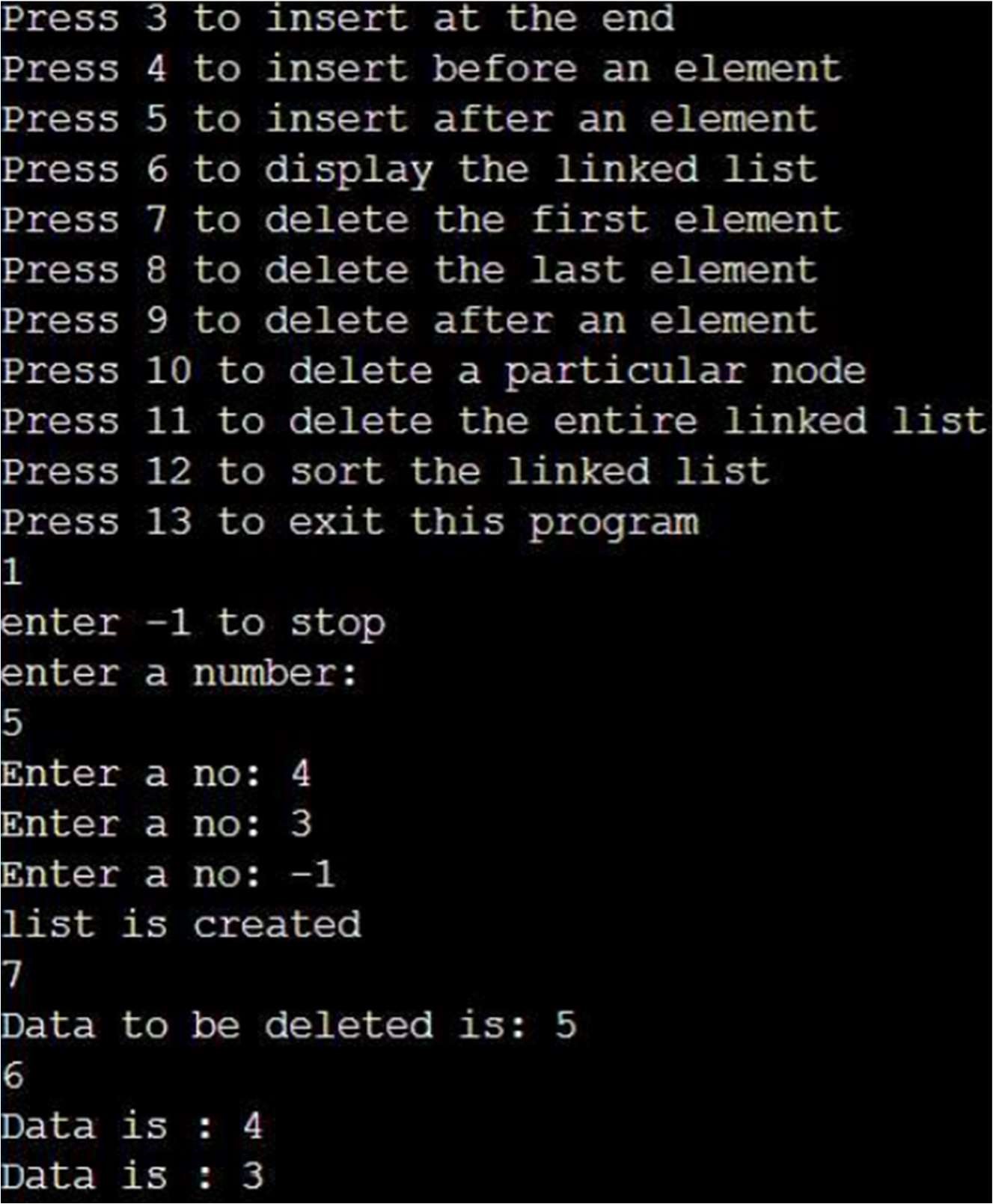
|  |
| --- |
| **EXP 4**  **LINKED LIST**  **NAME: DIVYESH KHUNT SAPID:60009210116**  **Aim:** To create and implement a linked list in c programming    **Theory:**  Linked is a data structure which uses dynamic memory allocation. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers. They include a series of connected nodes.  Here, each node stores the data and the address of the next node.  Advantages of Linked Lists over arrays:   * Dynamic Array. * Ease of Insertion/Deletion.   Drawbacks of Linked Lists:   * Random access is not allowed. We have to access elements sequentially starting from the first node(head node). So we cannot do a binary search with linked lists efficiently with its default implementation. * Extra memory space for a pointer is required with each element of the list. * Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.   **Time Complexity :**  For searching is O(n)  For insertion and deletion is O(1)    **CODE:** |

OUTPUTS:

Creation and display of LL



Deletion of data



**Conclusion**:

Thus linked list was created and manny operations on it were performed successfully

**EXPERIMENT 5**

|  |
| --- |
| **NAME: DIVYESH KHUNT SAPID:60009210116**  **PARENTESES**  **AIM:** To implement parentheses checker i.e. to check the expression is valid or not.  **THEORY:**  The balanced parentheses problem is one of the common programming problems that is also known as Balanced brackets. This problem is commonly asked by the interviewers where we have to validate whether the brackets in a given string are balanced on not.  Characters such as "(", ")", "[", "]", "{", and "}" are considered brackets  **Time Complexity**  The **time complexity** of the parenthesis checker implementation using stack is O(n) where n is the length of the input expression, as we are traversing the string character by character using for loop.  **Space Complexity**  The space complexity of the parenthesis checker implementation using stack is O(n) where n is the length of the input expression, as we are storing the opening parenthesis characters in a stack.  **CODE:**      **OUTPUT:**      **CONCLUSION:**  Thus we can check if the expression is valid or not with the help of stack.  **EXPERIMENT 6**  **NAME: DIVYESH KHUNT SAPID:60009210116**  **INFIX TO POSTFIX**  **AIM:** To convert an infix expression to postfix expression  **Theory:**  **Infix Expression:** In infix expression, an operator is placed between the two operands. Example: x + y, here operator + is placed between operands x and y.  **Postfix Expression:** In postfix expression, an operator is placed after the operands. Example: xy+, here operator + is placed after the operands x and y.    The precedence of these operators can be given as follows:  Higher priority \*, /, %  Lower priority +, –  **CODE:**          **OUTPUT**    **CONCLUSION:**  Thus, an application of stack to convert infix to postfix was performed successfully  **EXP 7**  **NAME: DIVYESH KHUNT SAPID:60009210116**  **Circular Queue**  **AIM:** To create and implement circular queue as an abstract data type  **THEORY:**  A Circular Queue is a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle.    **Time Complexity:** Time complexity of dequeue() operation is O(1) as there is no loop in any of the operation.  **CODE:**        **OUTPUT:**      **CONCLUSION:** Thus circular queue was implemented successfully  **EXP 8**  **NAME: DIVYESH KHUNT SAPID:60009210116**  **Priority Queue**  **Aim:** To create an type of queue (priority queue)  **THEORY:**  Priority Queue is an abstract data type that is similar to a queue, and every element has some priority value associated with it. The priority of the elements in a priority queue determines the order in which elements are served (i.e., the order in which they are removed).      **CODE:**        **OUTPUT:**      **EXP 9**  **NAME: DIVYESH KHUNT SAPID:60009210116**  **Binary Search Tree**  **AIM:** To implement an hierarchical data structure known as TREE  **THEORY:**  A binary search tree, also known as an ordered binary tree, is a variant of binary trees in which the nodes are arranged in an order. In a binary search tree, all the nodes in the left sub-tree have a value less than that of the root node. Correspondingly, all the nodes in the right sub-tree have a value either equal to or greater than the root node. The same rule is applicable to every sub-tree in the tree.      **CODE:**            **OUTPUT:**  Inserting a value    Traversals    Count nodes    Find smallest and largest    **Conclusion:**  In this experiment we saw how to implement binary search tree using linked list  **EXP 10**  **NAME: DIVYESH KHUNT SAPID:60009210116**  **KRUSKAL ALGO**  **Aim:** To implement and analyse Kruskal’s Algorithm  **Theory:** In Kruskal's algorithm, we start from edges with the lowest weight and keep adding the edges until the goal is reached. The steps to implement Kruskal's algorithm are listed as follows –   * First, sort all the edges from low weight to high. * Now, take the edge with the lowest weight and add it to the spanning tree. If the edge to be added creates a cycle, then reject the edge. * Continue to add the edges until we reach all vertices, and a minimum spanning tree is created   **CODE:**    **S**  **OUTPUTS:**      **Conclusion:**  In this experiment we analysed and implemented Kruskal’s algorithm code.  **EXP 11**  **NAME: DIVYESH KHUNT SAPID:60009210116**  **LINEAR HASHING**  **AIM:** To insert elements in array with help of linear hasing  **Theory**  Linear hashing is a dynamic data structure which implements a hash table and grows or shrinks one bucket at a time. The file structure of a dynamic hashing data structure adapts itself to changes in the size of the file, so expensive periodic file reorganization is avoided. A hash function is simply a mathematical function which then applied to a key, produces an integer which can be used as an index for the key in the hash table.  **CODE:**        **OUTPUTS:**        **CONCLUSION:**  Thus the elements were stored in array with help of linear hashing. |