P1 - Explain the scenarios where you can use linked lists and arrays?

Properties of Array :

1. Array provides constant time for accessing any element using their index. So it can be used in algorithms where the data elements need to have some order and constant access to elements is required.
2. The constant access time of the array is possible due to the fact that it uses contiguous storage and only allows the same kind of data to be stored in this storage space.

Scenarios where Array can be used :

1. Array is used as a container in many sorting algorithms.
2. Binary heap data structure which is used in priority queue abstract data type also uses array as container. The binary heap data structure can be implemented using Tree as well but array is much more concise as it doesn't need extra storage for references to child nodes. The indices of the array are sufficient to refer to the child nodes.
3. Array is used as a building block for many complex data types such as Stack, Queue etc. The ArrayList of Java and vector of C++ uses array as the container for elements.

Properties of Linked List :

1. Conceptually, a linked list can be defined as a linear chain-like structure of nodes having the same data type. A node is any abstract data type.
2. They allow constant time insertion.

Scenarios where Linked-list can be used :

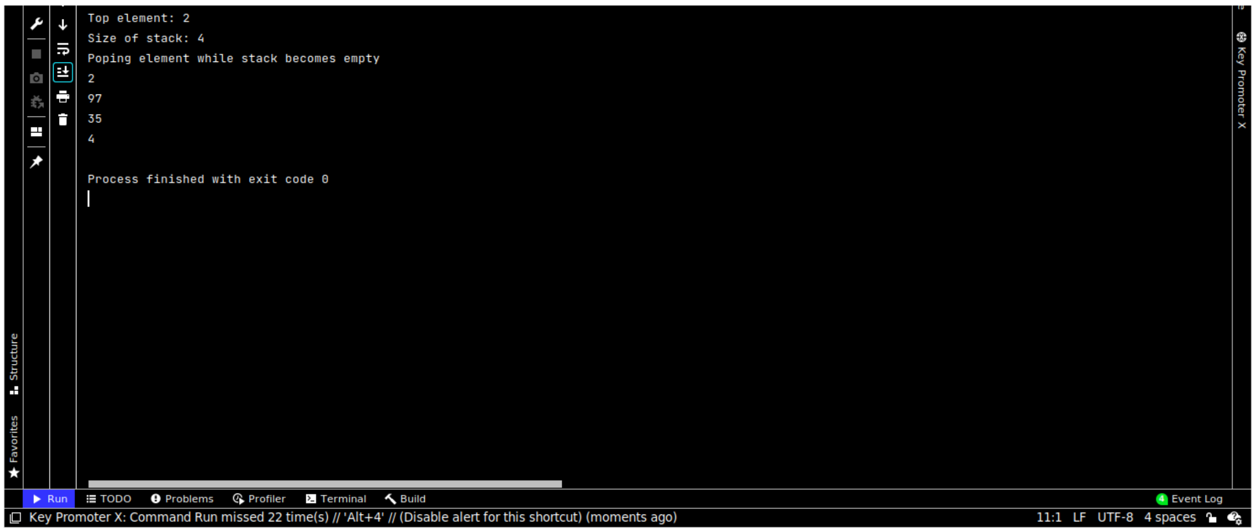
1. If the amount of data that the user will provide is not known a priori then linked-list is better choice than array as they provide constant time insert operation. Using an array will force the programmer to grow its size in case the initial capacity was not enough.
2. Linked-list is used to implement data types like Stack, Queue, Bag etc. The list data type of C++ and LinkedList data type of Java uses linked-list for their implementation.
3. There are many variants of linked-list which can be beneficial in different situations. Some of the variants are doubly linked list, circular linked-list.

P3-Write a program to Implement a stack using a singly linked list?

Stack.java file :

```
class LinkedListNode {
    public int value;
    public LinkedListNode next;

    LinkedListNode(int value, LinkedListNode next) {
        this.value = value;
        this.next = next;
    }
}

public class Stack {
    private LinkedListNode head;
    private int size;

    public void push(int value) {
        if (this.head == null) {
            this.head = new LinkedListNode(value, null);
        } else {
            this.head = new LinkedListNode(value, this.head);
        }
        this.size++;
    }

    public void pop() {
        if (this.isEmpty()) {
            throw new Error("Can't pop from an empty array");
        }
        this.head = this.head.next;
    }

    public int peek() {
        if (this.isEmpty()) {
            throw new Error("Can't peek in an empty array");
        }
        return this.head.value;
    }

    public boolean isEmpty() {
        return this.head == null;
    }
    public int size() {
        return this.size;
    }

    public static void main(String[] args) {
        Stack st = new Stack();

        st.push(4);
        st.push(35);
        st.push(97);
        st.push(2);

        System.out.println("Top element: " + st.peek());
        System.out.println("Size of stack: " + st.size());

        System.out.println("Poping element while stack becomes empty");
        while (!st.isEmpty()) {
            System.out.println(st.peek());
            st.pop();
        }
    }
}
```

Output screenshot :



```
Top element: 2
Size of stack: 4
Poping element while stack becomes empty
2
97
35
4

Process finished with exit code 0
```

Key Promoter X: Command Run missed 22 time(s) // 'Alt+4' // (Disable alert for this shortcut) (moments ago)

P4:-Justify the Time Complexity of give functions with proper reasoning;

a)-

```
int fun(int n) {
        int i, j;
        for(i=1; i<=n; i++) {
                for (j=1; j<n; j+=i){
                        printf("1");
                }
        }
}
```

In the 1st iteration, the inner for loop runs for (n-1) times, then ceil((n-1) / 2), then ceil((n-1) / 3) and so on. So the general term of this series is ceil((n-1) / i) for the ith iteration.

For simplicity and approximation, we can remove the ceil() and then the general term will become (n-1) / i. So the total number of operations will be proportional to $\sum_{i=1}^{n}(n-1)/i$

We can replace the summation by integral to approximate it. $\int_1^n (n-1)/i\, di$

So now it becomes, $(n-1) * \int_1^n 1/i\, di$ which is equal to $(n-1) * logn$.

Hence, its time complexity is equal to O(nlogn).

b):-

```
void fun(int n)
{
        int i = 1, sum =1;
        while (sum <= n)
        {
                i++;
                sum += i;
                printf("1");
        }
 }
```

After k iteration, value of i becomes i + k and value of sum becomes the sum of natural number upto k + 1. Let m be the number of times while loop runs after which value of sum becomes > n.

Then at this point, value of sum will be (m + 1)(m+2) / 2. Because the value of sum variable is the sum of all natural numbers upto m + 1. Solving equation (m + 1)(m+2) / 2 > n using the usual techniques of quadratic equation, we get $m > -3 + \mathrm{sqrt}(1 + 4n) / 2$. Hence its complexity will be $O(m) = O(\mathrm{sqrt}(n))$.

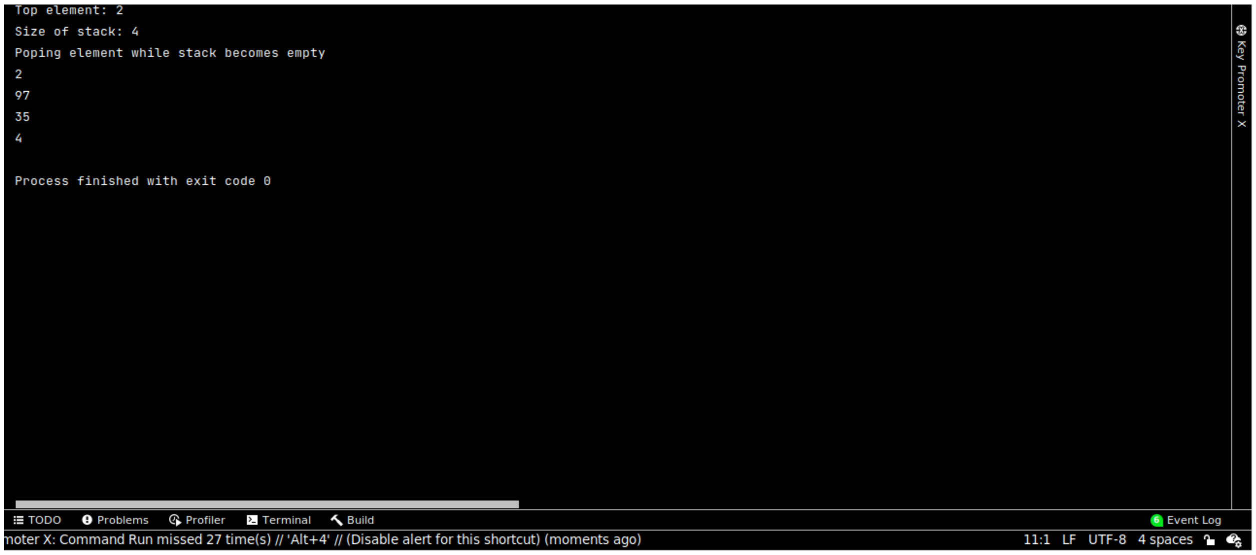
P5:- How do you implement a stack using queues?

I will use two queues. Whenever I have to push an element, I will enqueue it on the first queue. For pop operation, I will dequeue the element from the 2nd queue. If 2nd queue is empty then I will dequeue all elements from the 1st queue and enqueue them in the 2nd one in the same order as they have been removed from the 1st queue. This will ensure that the stack order(LIFO) is maintained.

Stack.java file :

```
import java.util.LinkedList;

public class Stack {
    private LinkedList<Integer> q1 = new LinkedList<>();
    private LinkedList<Integer> q2 = new LinkedList<>();
    private int size;

    private void fillSecondQueue() {
        if (q2.isEmpty()) {
            while (!q1.isEmpty()) {
                q2.add(q1.removeLast());
            }
        }
    }

    public void push(int value) {
        q1.add(value);
        size++;
    }

    public void pop() {
        if (isEmpty()) {
            throw new Error("Can't pop from an empty Stack");
        }
        this.fillSecondQueue();
        q2.removeFirst();
        size--;
    }

    public int peek() {
        if (isEmpty()) {
            throw new Error("Can't peek in an empty Stack");
        }
        this.fillSecondQueue();
        return q2.getFirst();
    }

    public boolean isEmpty() {
        return q1.isEmpty() && q2.isEmpty();
    }

    public int size() {
        return this.size;
    }

    public static void main(String[] args) {
        Stack st = new Stack();

        st.push(4);
        st.push(35);
        st.push(97);
        st.push(2);

        System.out.println("Top element: " + st.peek());
        System.out.println("Size of stack: " + st.size());

        System.out.println("Poping element while stack becomes empty");
        while (!st.isEmpty()) {
            System.out.println(st.peek());
            st.pop();
        }
    }


}
```

Output of the code is as shown below.



```
Top element: 2
Size of stack: 4
Poping element while stack becomes empty
2
97
35
4

Process finished with exit code 0
```

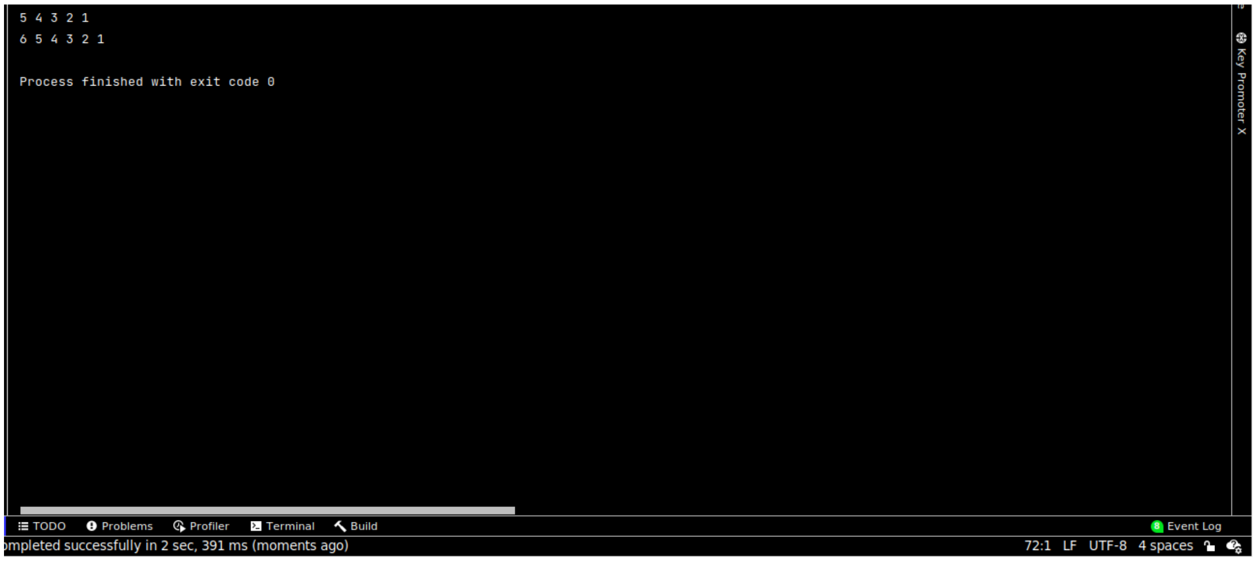
```
moter X: Command Run missed 27 time(s) // 'Alt+4' // (Disable alert for this shortcut) (moments ago)
```

P6- Write a program to reverse a linklist using both iterative and recursive methods.

LinkedList.java file :

```
import java.util.ArrayList;

public class LinkedList {
    public LinkedListNode head;
    public LinkedListNode tail;

    public LinkedList reverseIterative() {
        LinkedList list = new LinkedList();
        LinkedListNode current = head;
        ArrayList<Integer> arr = new ArrayList<>();
        while (current != null) {
            arr.add(current.value);
            current = current.next;
        }

        for (int i = arr.size() - 1; i >= 0; i--)
            list.insert(arr.get(i));

        return list;
    }

    private LinkedList reverseRecursive(LinkedListNode current, LinkedList list) {
        if (current == null)
            return list;
        list = reverseRecursive(current.next, list);
        list.insert(current.value);
        return list;
    }

    public LinkedList reverseRecursive() {
        LinkedList list = new LinkedList();
        return reverseRecursive(head, list);
    }

    public void insert(int value) {
        if (head == null) {
            head = new LinkedListNode(value, null);
            tail = head;
        } else {
            LinkedListNode node = new LinkedListNode(value, null);
            tail.next = node;
            tail = node;
        }
    }

    public void print() {
        LinkedListNode current = head;
        while (current != null) {
            System.out.print(current.value + " ");
            current = current.next;
        }
        System.out.println();
    }

    class LinkedListNode {
        public int value;
        public LinkedListNode next;

        LinkedListNode(int value, LinkedListNode next) {
            this.value = value;
            this.next = next;
        }
    }

    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.insert(1);
        list.insert(2);
        list.insert(3);
        list.insert(4);
        list.insert(5);

        LinkedList iterRev = list.reverseIterative();
        iterRev.print();

        list.insert(6);
        LinkedList recurRev = list.reverseRecursive();
        recurRev.print();
    }
}
```

Output of the code is as shown below.



```
5 4 3 2 1
6 5 4 3 2 1

Process finished with exit code 0
```