Function Analyze

Berkan AKIN
171044073

MyLinkList Class

```
public class MyLinkList <E> {
	private static class Node<E>{
		private E data;
		private Node <E> next;

		private Node(E dataItem) {
			data =dataItem;
			next=null;
		}
		private Node(E dataItem,Node<E> nodeRef) {
			data =dataItem;
			next=nodeRef;
		}
	}


	private Node<E> head =null;
	private int size=0;
	public int getSize() {
		return size;
	}
	private void addFirst(E item) {
		head = new Node<> (item,head);                          1
		size++;                                                  1

	}
	O(1)
```

```
	private void addAfter(Node<E> node,E item) {
		node.next = new Node<> (item,node.next);    1
		size++;                                     1
	}
	T(n)=2      O(1)
```

```
	private E removeAfter(Node<E> node) {
		Node<E> temp = node.next;
		if(temp !=null) {
			node.next=temp.next;      1
			size--;                   1
			return temp.data;          1
		}
		else {
			return null;               1
		}
	}
	Tb(4) = O(1)
	Tw(2) = O(1)
```

```
	private E removeFirst() {
		Node<E> temp =head;
		if(head != null) {         1
			head = head.next;    1
		}
		if(temp != null) {          1
```

```
            size--;                1
            return temp.data;      1
        }
        else {
            return null;             1
        }
    }
        O(n)
```

```
        private Node<E> getNode(int index){
            Node<E> node = head;
            for(int i=0; i <index && node !=null;i++) {   n
                node=node.next;                           n
            }
            return node;                                  1
        }

            O(n)
```

```
        public E get(int index) {
            if(index < 0 || index >= size) {     1
                //throw
            }
            Node <E> node =getNode(index);    1
            return node.data;                 1
        } O(1)
```

```
        public E set(int index,E newValue) {
            if(index < 0 || index >=size) {    1
                //throw
            }
            Node<E> node = getNode(index);    1
            E result =node.data;              1
            node.data=newValue;               1
            return result;                   1
        }
            O(1)
```

```
        public void add(int index,E item) {
            if(index <0 || index > size) {   1
                //throw
            }
            if(index==0) {                      1
                addFirst(item);                 1
            }
            else {                              1
                Node<E> node =getNode(index-1);  1
                addAfter(node,item);              1
            }

        }
            O(1)
```

```
        public boolean add(E item) {
            add(size,item);
            return true;
        }
        O(n)
```

```
public boolean remove(int index) {

      // if the index is out of range, exit
      if (index < 1 || index > size)
            return false;

      Node temp = head;
      if (head != null) {
            for (int i = 0; i < index; i++) {      n
                  temp=temp.next;                   n


            }
            temp=temp.next.next;                     1

            // decrement the number of elements variable
            size--;
            return true;                             1

      }
      return false;                                    1
}
```

O(n)

MyArraylist Class

```
public MyArrayList() {
      capacity=INITIAL_CAPACITY;
      theData =(E[]) new Object[capacity];
}

public boolean add(E anEntry) {
      if(size==capacity) {              1
            //reallocate();
      }
      theData[size]= anEntry;             1
      size++;                             1
      return true;                         1
}
```

O(1)

```
public void add(int index,E anEntry) {
      if(index < 0  || index > size) {
            //throw new ArrayIndexOutBoundsExeception(index);
      }
      if(size==capacity) {     1
            //reallocate();
      }
      for(int i=size;i > index ;i--) { n
            theData[i] = theData[i-1];  n
      }

      theData[index] =anEntry;      1
      size++;                       1
}
```

O(n)

```
public E get(int index) {
            if(index < 0 || index >= size) {       1
                  //throw
            }

            return theData[index];                     1
      }
      O(1)
```

```
      public E set(int index, E newValue) {
            if(index < 0 || index >= size) {   1
                  //throw
            }
            E oldValue =theData[index];     1
            theData[index] = newValue;       1
            return oldValue;           1
      }

      O(1)
```

```
      public E remove(int index) {
            if(index < 0 || index >= size) {    1
                  //throw
            }
            E returnValue =theData[index];      1
            for(int i=index +1; i < size ; i++) {   n +1
                  theData[i-1] = theData[i];        n
            }
            size--;                                 1
            return returnValue;                     1
      }
      O(n)
```

```
      private void reallocate() {
            capacity=2*capacity;                                  1
            theData = Arrays.copyOf(theData, capacity);         1
      }
      public int getSize() {                                        1
            return size;                                            1
      }
               O(1)
```

HybrisdList Class

```
public void add(E item) {
            if(size==0) {
                  Node firstNode = new Node(item);     1
                  head = firstNode;                     1
                  iter = firstNode;                    1
                  size++;                             1
            }
            else if(size>=5 && size%CAPACITY ==0) {
```

```
            Node newNode = new Node(item);        1
            iter.next = newNode;                   1
            iter =iter.next;                        1
            size++;                                  1

      }
      else {
            iter.data.add(item);     n
            size++;           1
      }                       1

}

O(n)  Q(1)
```

```
public E get(int index) {
      Node temp =head;                                          1
      int nodeNum=index/CAPACITY;                               1
      int i=0;
      while(i<nodeNum) {                                        n
            temp=temp.next;                                      n
            i++;                                                 n
      }
      return (E) temp.data.get(index%CAPACITY);                    1
}

O(n)
```

```
public E set(int index,E item) {
      Node temp =head;     1
      int nodeNum=index/CAPACITY;             1
      int i=0;                                   1
      if(index<0 || index>=size) {             1
            //throw                              1
      }
      while(i<nodeNum) {                       n
            temp=temp.next;                    n
            i++;                               n
      }
      return (E) temp.data.set(index%CAPACITY,item); n
}
O(n)
```

```
public void add(int index, E item) {
      int i=0,nodeNum=index /CAPACITY;
      int arrIndex=index%CAPACITY;
      E tmpValue;
      Node temp =head;
      if(index==size) {
            this.add(item);
            return;
      }
      while(i<nodeNum) {                                  n
            temp=temp.next;                               n
            i++;                                         n

      }

      i=0;
      tmpValue = (E) this.set(index, item);
```

```
        index++;
        while(index<size) {                                    n
              if(index>=CAPACITY && index%CAPACITY==0) {        n
                    temp=temp.next;                              n
              }
              tmpValue =(E) this.set(index, tmpValue);
              index++;
        }
        this.add(tmpValue);

  }
        O(n)
```

```
  public E remove(int index) {
        E tmpValue=this.set(index, this.get(index+1));      1
        index++;                                             1
        while(index <size) {                                 n
              this.set(index, this.get(index+1));             n
              index++;                                        n
        }
        size--;                                               1
        return tmpValue;                                       1
  }
  O(n)
```

```
  public void pirntS() {                                          1
        Node <E> temp=head;                                       1
        int i=0;
        for(i=0;i<size;i++) {                                      n

              if(i>=CAPACITY && i%CAPACITY==0) {                   n
                    temp=temp.next;
              }
              System.out.println(temp.data.get(i%5));              1
        }

  }
 O(1)
```