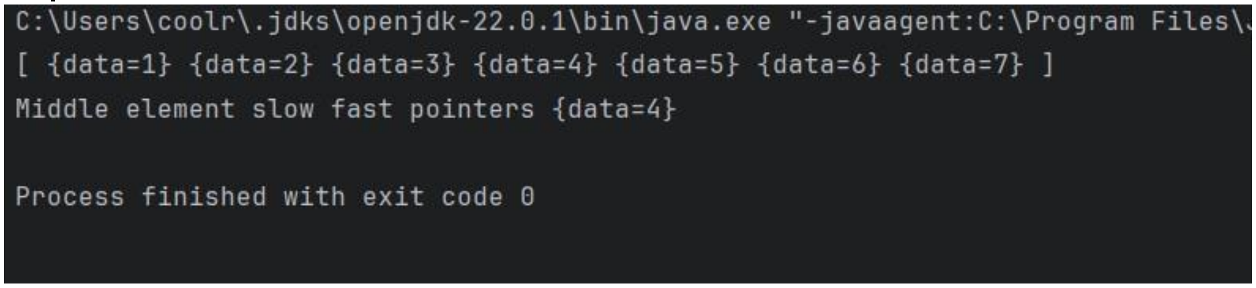
Day 14 Assignment

1. **You are given a singly linked list. Write a function to find the middle element without using any extra space and only one traversal through the linked list.**

```java
public Node<T> getMiddle() {
    Node<T> first = getFirst();
    Node<T> second = getFirst();

    // Check if the list is empty
    if (first == null) {
        return null;
    }


    // Traverse the list to find the middle
    while (second != null && second.getNext() != null) {
        second = second.getNext().getNext();
        first = first.getNext();
    }


    return first;
}
package m5_core_java_programming.day_14;



import m5_core_java_programming.mycollection.LinkedList;


/*
    You are given a singly linked list. Write a function to find the middle
element
    without using any extra space and only one traversal through the linked
list.
*/


public class Assignment_1 {
    public static void main(String[] args) {
        LinkedList<Integer> ls = new LinkedList<>();
        ls.add(1);
        ls.add(2);
        ls.add(3);
        ls.add(4);
        ls.add(5);
        ls.add(6);
        ls.add(7);
        System.out.println(ls);
      System.out.println("Middle element slow fast pointers "+ls.getMiddle());
    }
}
```

Output



```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\
[ {data=1} {data=2} {data=3} {data=4} {data=5} {data=6} {data=7} ]
Middle element slow fast pointers {data=4}

Process finished with exit code 0
```

---

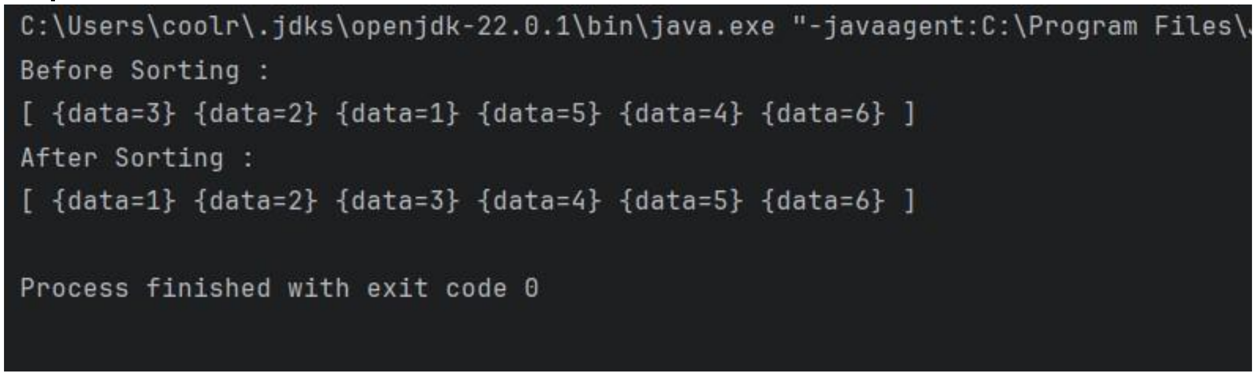
2. **You have a queue of integers that you need to sort. You can only use additional space equivalent to one stack. Describe the steps you would take to sort the elements in the queue.**

- First I will check if the queue is empty or not.
- If its not empty then,
- I will remove the first element from the queue and put it inside the stack.
- Then I will run an infinite loop
- Inside the Infinite loop my first condition is to check whether stack and queue is not empty.
- If both of them are not empty then I will compare the head of the queue and stack.
- If the head of the stack is bigger than the queue's head then I will put it back in queue.
- Else I will put the head of queue and pushed it to the stack
- Else if check if the stack is empty
- If the stack is empty then remove the head of the queue and put it back inside the stack.
  - Else if q is empty then
- Pop out all the items from the stacks and then add it to the queue, break the infinite loop with a return statement.

```
package m5_core_java_programming.day_14;
```

```java
import m5_core_java_programming.mycollection.Queue;
import m5_core_java_programming.mycollection.Stack;


/*
   You have a queue of integers that you need to sort. You can only use
additional space equivalent
  to one stack. Describe the steps you would take to sort the elements in the
queue.
*/
public class Assignment_2 {


    public static void sortQueue(Queue<Integer> q, Stack<Integer> stk) {
        if (q.size() == 0) {
            return;
        }


        stk.push(q.poll());


        while (true) {
            if (!stk.empty() && q.size() != 0) {
                int a = stk.peek();
                int b = q.getFirst();
                if (a > b) {
                    stk.push(q.poll());
                } else {
                    q.add(stk.pop());
                }
            } else if (stk.empty()) {
                stk.push(q.poll());
            } else if (q.size() == 0) {
                while (!stk.empty()) {
                    q.add(stk.pop());
                }
                return;
            }
        }
    }


    public static void main(String[] args) {
        Queue<Integer> q = new Queue<>();
        Stack<Integer> stk = new Stack<>();


        q.add(3);
        q.add(2);
        q.add(1);
        q.add(5);
        q.add(4);
        q.add(6);
        System.out.println("Before Sorting : ");
        System.out.println(q);
        sortQueue(q, stk);
        System.out.println("After Sorting : ");
        System.out.println(q);


    }
}
```

Output



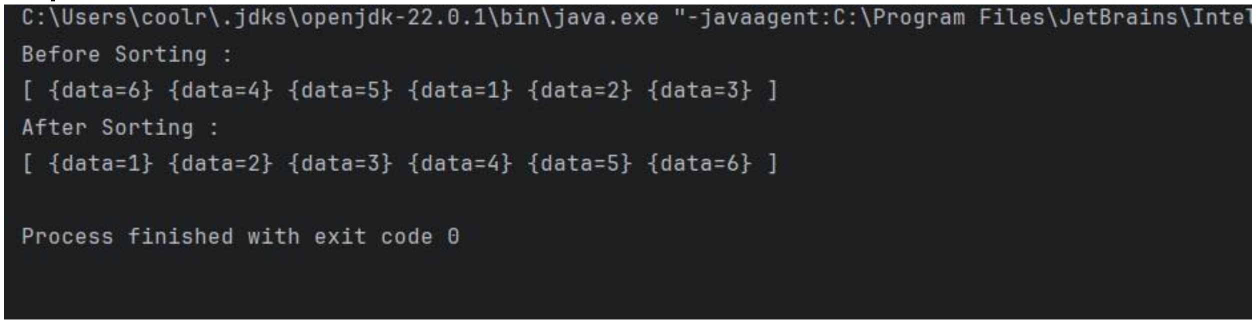
```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\
Before Sorting :
[ {data=3} {data=2} {data=1} {data=5} {data=4} {data=6} ]
After Sorting :
[ {data=1} {data=2} {data=3} {data=4} {data=5} {data=6} ]

Process finished with exit code 0
```

---

3. **You must write a function to sort a stack such that the smallest items are on the top. You can use an additional temporary stack, but you may not copy the elements into any other data structure such as an array. The stack supports the following operations: push, pop, peek, and isEmpty.**

```
package m5_core_java_programming.day_14;


/*
  You must write a function to sort a stack such that the smallest items are
on the top.
   You can use an additional temporary stack, but you may not copy the elements
into any
```

```
   other data structure such as an array. The stack supports the following
operations:
   push, pop, peek, and isEmpty.
*/


import m5_core_java_programming.mycollection.Stack;


public class Assignment_3 {


   public static void sortStack(Stack<Integer> stk) {
       Stack<Integer> tempStack = new Stack<>();


       if (stk.isEmpty()) {
           return;
       }


       tempStack.push(stk.pop());


       while (true) {
           int current = stk.pop();


          while (!tempStack.isEmpty() && tempStack.peek() > current) {
               stk.push(tempStack.pop());
           }


           tempStack.push(current);


           if (stk.isEmpty()) {
               while (!tempStack.isEmpty()) {
                   stk.push(tempStack.pop());
               }
               return;
           }
       }


   }
   public static void main(String[] args) {
       Stack<Integer> stk = new Stack<>();

       stk.push(3);
       stk.push(2);
       stk.push(1);
       stk.push(5);
       stk.push(4);
       stk.push(6);


       System.out.println("Before Sorting :");
       System.out.println(stk);
       sortStack(stk);
       System.out.println("After Sorting :");
       System.out.println(stk);
   }
}
```
Output



```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\Inte
Before Sorting :
[ {data=6} {data=4} {data=5} {data=1} {data=2} {data=3} ]
After Sorting :
[ {data=1} {data=2} {data=3} {data=4} {data=5} {data=6} ]

Process finished with exit code 0
```

---

4. **A sorted linked list has been constructed with repeated elements. Describe an algorithm to remove all duplicates from the linked list efficiently.**

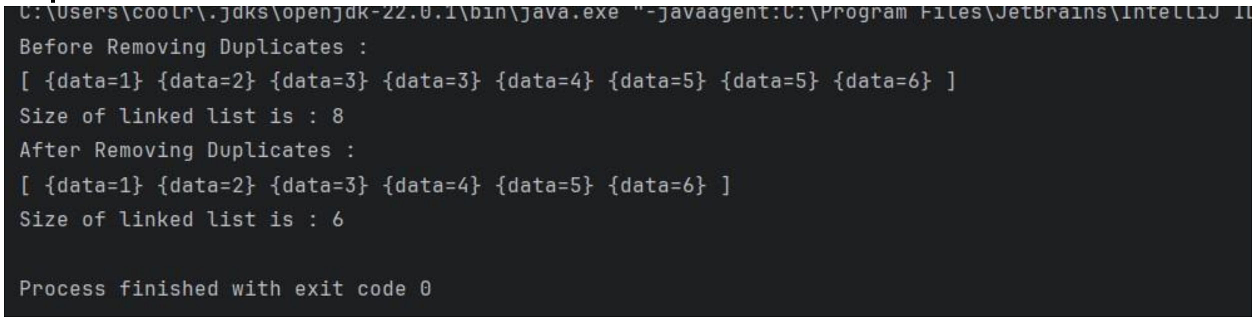
- Create a pointer curr which will point towards the head
- While curr is not null traverse the whole linkedlist
- Create one more pointer which point towards the current node ● Create one more element counter and set it at -1.
- While at is not null and current node data is equal to at node data
- At will go to next node
- And Counter will be increased by 1.
- After the end of the loop, the current node’s next pointer will point towards the at pointer and length will decrease by the counter of duplicates.
- If at is null then
- Set current node as the end of the linkedlist

```
public void removeDuplicates() {
   if (head == null) {
       return;
   }
   Node<T> curr = head;


   while (curr != null) {
       Node<T> at = curr;
       int i = -1;
      while (at != null && at.getData().compareTo(curr.getData()) == 0) {
           at = at.getNext();
           i++;
       }
       curr.setNext(at);
       length -= i;
       if (at == null) {
           end = curr;
           end.setNext(at);
       }
       curr = curr.getNext();


   }
}
package m5_core_java_programming.day_14;


import m5_core_java_programming.mycollection.LinkedList;


/*
   A sorted linked list has been constructed with repeated elements.
   Describe an algorithm to remove all duplicates from the linked list
efficiently.
*/
public class Assignment_4 {
   public static void main(String[] args) {
       LinkedList<Integer> ls = new LinkedList<>();


       ls.add(1);
```

[1] **. Given a stack and a smaller array representing a sequence, write a function that determines if the sequence is present in the stack. Consider the sequence present if, upon popping the elements, all elements of the array appear consecutively in the stack.**

```java
        ls.add(5);
        ls.add(6);
        ls.add(1);
        ls.add(3);
        ls.add(3);
        ls.add(4);
        ls.add(2);


        ls.bubbleSort();
        System.out.println("Before Removing Duplicates :");
        System.out.println(ls);
        System.out.println("Size of linked list is : " + ls.size());
        ls.removeDuplicates();
        System.out.println("After Removing Duplicates :");
        System.out.println(ls);
        System.out.println("Size of linked list is : " + ls.size());
    }
}
```

Output

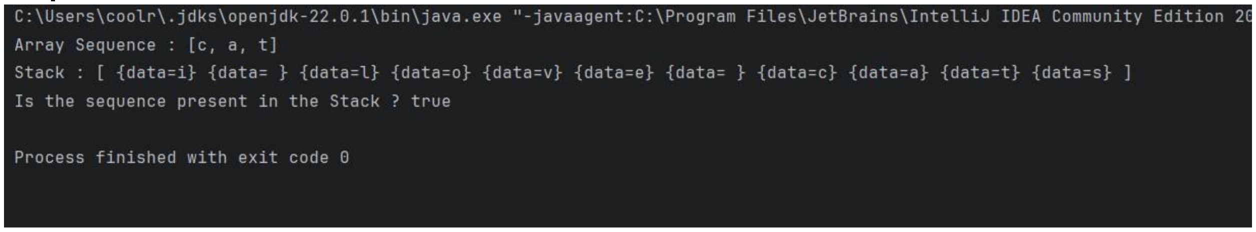


```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IL
Before Removing Duplicates :
[ {data=1} {data=2} {data=3} {data=3} {data=4} {data=5} {data=5} {data=6} ]
Size of linked list is : 8
After Removing Duplicates :
[ {data=1} {data=2} {data=3} {data=4} {data=5} {data=6} ]
Size of linked list is : 6

Process finished with exit code 0
```

```java
package m5_core_java_programming.day_14;


import m5_core_java_programming.mycollection.Stack;


import java.util.Arrays;
```

```java
/*
  Given a stack and a smaller array representing a sequence, write a function
that determines
  if the sequence is present in the stack. Consider the sequence present if,
 upon popping the elements, all elements of the array appear consecutively in
the stack.
*/
public class Assignment_5 {
   public static boolean checkSeq(Stack<Character> stk, char[] sequence) {
       if (stk.isEmpty()) {
           return false;
       }
       if (sequence.length == 0) {
           return true;
       }


       int[] table = new int[sequence.length];
       prefixTable(table, sequence);

       int j = 0;
       while (!stk.isEmpty()) {
           if (stk.peek() == sequence[j]) {
               j++;
               stk.pop();
           } else {
               while (j != 0 && stk.peek() != sequence[j]) {
                   j = table[j - 1];
               }
               if (stk.peek() == table[j]) {
                   j++;
               }
               stk.pop();
           }
           if (j == sequence.length) {
               return true;
           }
       }
       return false;
   }


   public static void prefixTable(int table[], char[] sequence) {
        int j = 0;
        for (int i = 1; i < sequence.length; i++) {
            if (sequence[j] == sequence[i]) {
                table[i] = j;
                j++;
            } else {
                while (j != 0 && sequence[j] != sequence[i]) {
                    j = table[j - 1];
                }
            }
        }
    }


    public static void main(String[] args) {
        Stack<Character> stk = new Stack<>();


        stk.push('s');
        stk.push('t');
        stk.push('a');
        stk.push('c');
        stk.push(' ');
        stk.push('e');
        stk.push('v');
        stk.push('o');
        stk.push('l');
        stk.push(' ');
        stk.push('i');


        char[] sequence = new char[3];


        sequence[0] = 'c';
        sequence[1] = 'a';
        sequence[2] = 't';


       System.out.println("Array Sequence : "+ Arrays.toString(sequence));
        System.out.println("Stack : "+stk);
        System.out.println("Is the sequence present in the Stack ? " +
checkSeq(stk, sequence));
    }
}
```

Output



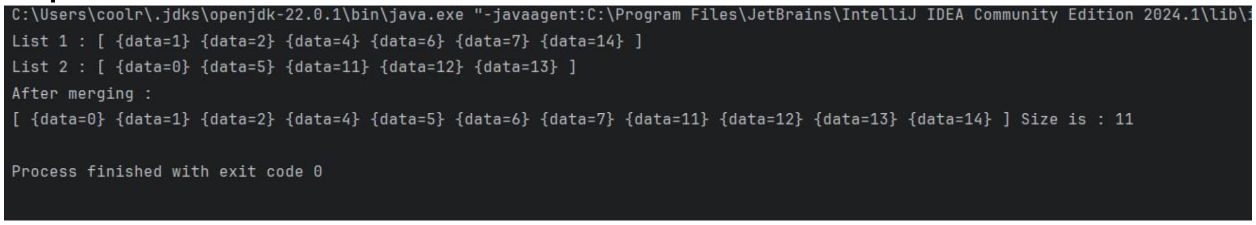
```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 20
Array Sequence : [c, a, t]
Stack : [ {data=i} {data= } {data=l} {data=o} {data=v} {data=e} {data= } {data=c} {data=a} {data=t} {data=s} ]
Is the sequence present in the Stack ? true

Process finished with exit code 0
```

---

6. **You are provided with the heads of two sorted linked lists. The lists are sorted in ascending order. Create a merged linked list in ascending order from the two input lists without using any extra space (i.e., do not create any new nodes).**

```
public void addAllSorted(LinkedList<T> second) {
   if (second.size() == 0) {
       return;
   }
   if (head == null && second.size() != 0) {
       head = second.getFirst();
       end = second.getLast();
       length = second.size();
       return;
   }
   if (head == null && second.size() == 0) {
       return;
   }
   if (end.getData().compareTo(second.head.getData()) < 0) {
       end.setNext(second.getFirst());
       end = second.getLast();
       end.setNext(null);
       length += second.length;
       return;
   }


   Node<T> point = null;
   Node<T> left = this.head;
   Node<T> right = second.getFirst();


   if (left != null && (right == null ||
left.getData().compareTo(right.getData()) <= 0)) {
       point = left;
       left = left.getNext();
   } else if (right != null) {
       point = right;
       right = right.getNext();
   }
   this.head = point;


   while (left != null && right != null) {
       if (left.getData().compareTo(right.getData()) <= 0) {
           point.setNext(left);
           left = left.getNext();
       } else {
           point.setNext(right);
           right = right.getNext();
       }
       point = point.getNext();
   }


   if (left != null) {
       point.setNext(left);
   } else if (right != null) {
       point.setNext(right);
       end = second.getLast();
   }


   this.length += second.size();
}
package m5_core_java_programming.day_14;


/*
   You are provided with the heads of two sorted linked lists.
   The lists are sorted in ascending order.
   Create a merged linked list in ascending order from the two input lists
   without using any extra space (i.e., do not create any new nodes).
*/


import m5_core_java_programming.mycollection.LinkedList;
```

```java
public class Assignment_6 {
   public static void main(String[] args) {
       LinkedList<Integer> ls1 = new LinkedList<>();
       LinkedList<Integer> ls2 = new LinkedList<>();
       ls1.add(1);
       ls1.add(2);
       ls1.add(4);
       ls1.add(6);
       ls1.add(7);
       ls1.add(14);


       ls2.add(0);
       ls2.add(5);
       ls2.add(11);
       ls2.add(12);
       ls2.add(13);

       System.out.println("List 1 : " + ls1);
       System.out.println("List 2 : " + ls2);
       ls1.addAllSorted(ls2);
       System.out.println("After merging : ");
       System.out.println(ls1 + " " + "Size is : " + ls1.size());


   }
}
```
Output:



```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1\lib\
List 1 : [ {data=1} {data=2} {data=4} {data=6} {data=7} {data=14} ]
List 2 : [ {data=0} {data=5} {data=11} {data=12} {data=13} ]
After merging :
[ {data=0} {data=1} {data=2} {data=4} {data=5} {data=6} {data=7} {data=11} {data=12} {data=13} {data=14} ] Size is : 11

Process finished with exit code 0
```

---

7. **Consider a circular queue (implemented using a fixed-size array) where the elements are sorted but have been rotated at an unknown index. Describe an approach to perform a binary search for a given element within this circular queue.**

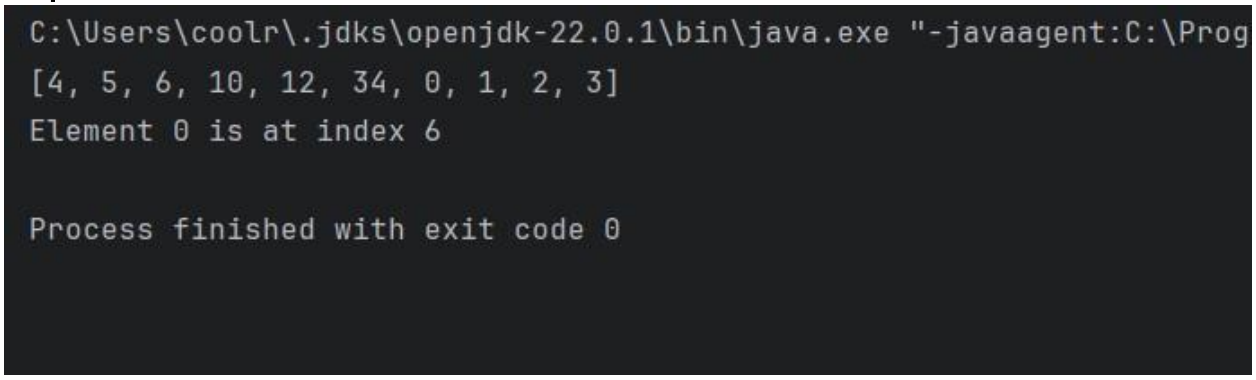
- First set two points, let's say left and right to the 0th index and last index of the array respectively.

- After that run a while loop until left is not greater than right.
- Calculate middle index using (left+right)/2 formula.
- If the element is found in the middle.
- Return index.
- Else if the element at middle is less than the right element then it may be in sorted order.
- If the middle element is smaller than the target and the right element is bigger than target.
- Then move towards the right side of the split.
- Else move towards the left side
- Else
- If the target is bigger than the left element and smaller than the middle element than the left most part if sorted.
- Move towards left side of the split
- Else move towards the right side
- If left goes beyond right then break the loop and return -1

```java
package m5_core_java_programming.day_14;


/*
   Consider a circular queue (implemented using a fixed-size array)
   where the elements are sorted but have been rotated at an unknown index.
  Describe an approach to perform a binary search for a given element within
this circular queue.
*/


import java.util.Arrays;


public class Assignment_7 {
   public static int indexOf(int[] arr, int element) {
       int left = 0;
       int right = arr.length - 1;


       while (left <= right) {
           int mid = (left + right) / 2;
           if (arr[mid] == element) {
               return mid;
           } else if (arr[mid] < arr[right]) {
               if (arr[mid] < element && element < arr[right]) {
                   left = mid + 1;
               } else {
                   right = mid - 1;
```

```java
                }
            } else {
                if (arr[mid] > element && element > arr[left]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            }
        }


        return -1;
    }


    public static void main(String[] args) {
        int[] arr = new int[10];


        arr[0] = 4;
        arr[1] = 5;
        arr[2] = 6;
        arr[3] = 10;
        arr[4] = 12;
        arr[5] = 34;
        arr[6] = 0;
        arr[7] = 1;
        arr[8] = 2;
        arr[9] = 3;
        System.out.println(Arrays.toString(arr));
        System.out.println("Element 0 is at index " + indexOf(arr, 0));
    }
}
```

Output



```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Prog
[4, 5, 6, 10, 12, 34, 0, 1, 2, 3]
Element 0 is at index 6

Process finished with exit code 0
```

: