public Node findMiddleNode

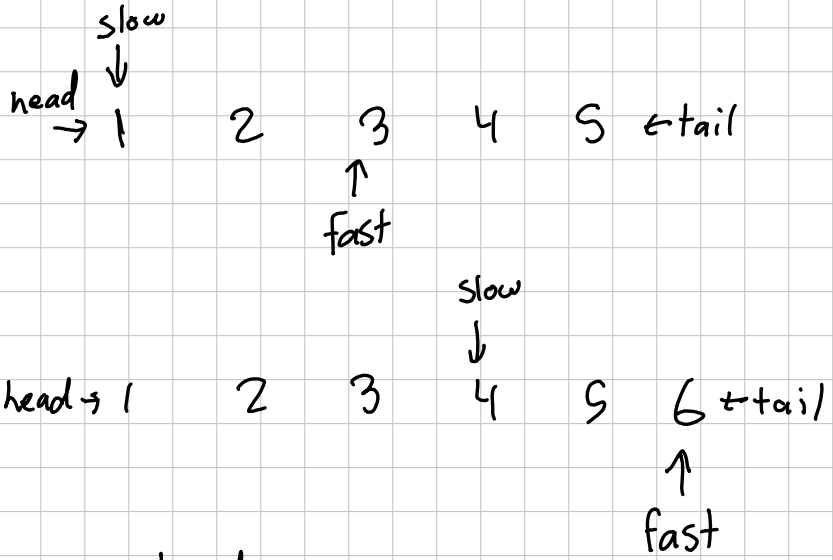
1 → 2 → 3 → (4) → 5 → 6

1 → 2 → (3) → 4 → 5

length ∅

Use Floyd's Tortoise and Hare algo

- 2 pointers
- one moves twice as fast as the other
- when fast reach end, then slow is at middle



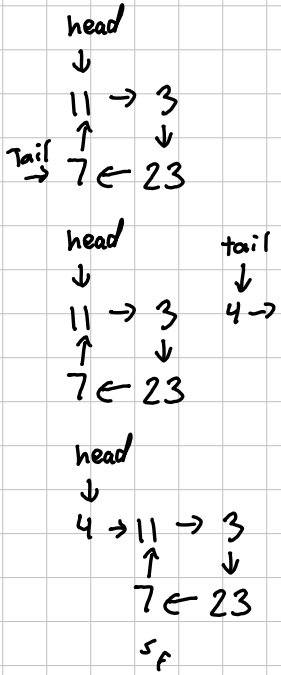

```
slow = head
fast = head
  if (head == null) return null
 while (fast.next != null)
          fast = fast.next
          slow = slow.next
          if (fast.next != null)
             fast = fast.next
return slow
```

public boolean hasLoop

Use Floyd's cycle-finding algo
- slow moves one step at a time
- fast moves two steps at a time
- if loop, then pointers will meet
- if no loop, fast reach end

Constraints:
- traverse list once
- no additional Data S. or modifying existing data struc.

```
slow = head
fast = head

if (head == null)
    return false

while (fast.next != null)
    slow = slow.next
    fast = fast.next
    if (fast.next != null)
        fast = fast.next
    if (fast == slow)
        return true

return false
```



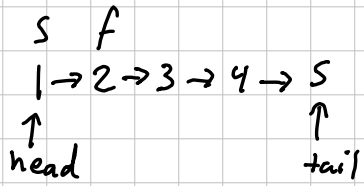
public Node findKthFromEnd(int k)

1 → 2 → 3 → 4 → 5

Ø length

k=2 → 4
k=5 → 1
k=6 → null

Use 2 pointers technique




```
if (head == null)
  return null

if (head.next == null)
     return head

slow = head
fast = head

for (int i = 1 , i < k, i++)
       fast = fast.next
       if (fast == null)
            return null

while (fast.next != null)
       fast = fast.next
       slow = slow.next

return slow
```

public void partitionList (int x)

3 → 8 → 5 → 10 → 2 → 1 x = 5

values < 5 : 3, 2, 1

values ≥ 5 : 8, 5, 10

OUT : 3 → 2 → 1 → 8 → 5 → 10

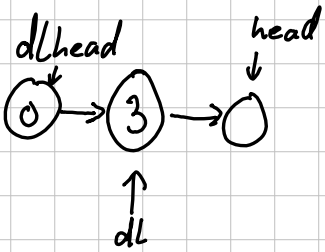
- ∅ Tail
- ∅ modify values

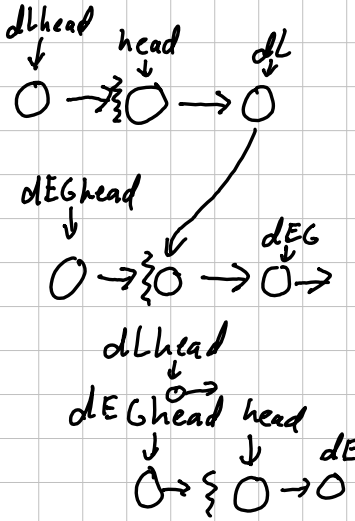
---

1 → 4 → 3 → 2 → 5 → 2 x = 3

OUT : 1 → 2 → 2 → 4 → 3 → 5

```
if (head == null || head.next == null) return
   dLNode = new Node(0)
   dEGNode = new Node(0)
     dLhead = dLNode
     dEGhead = dEGNode
for (int i=0 ; i<length ; i++)
      if (head.value< x)
            dLNode.next = head
            head = head.next
            dLNode = dLNode.next
            dLNode.next = null
      else
            dEGNode.next = head
            head = head.next
            dEGNode = dEGNode.next
            dEGNode.next = null
   if(dLNode != dLhead)
      head = dLhead.next
       dLhead.next = null
      dLNode.next= dEGhead.next
       dEGhead.next = null
   else
         head= dEGhead.next
          dEGhead.next = null
```





public void removeDuplicates() @Tail

Without set $O(n^2)$
Using HashSet $O(n)$

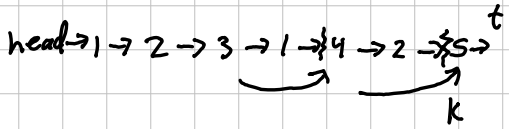
$1 \to 2 \to 3 \to 1 \to 4 \to 2 \to 5$
OUT: $1 \to 2 \to 3 \to 4 \to 5$

↳ stores collection of <u>unique</u> elements, can check for duplicates

```
if ( head ==null || head.next ==null)
    return
```

```
HashSet <Integer> set = new HashSet <>()
```

```
temp = head
keep = head
```




```
while ( temp != null)

    if ( set.add (temp.value))
            keep = temp
            temp = temp.next
    else
        keep.next = temp.next
        temp.next = null
        temp = keep.next
```

```
public int binaryToDecimal()
```

```
      ②   ③   ④
head→ 1 → 0 → 1 ←tail
```

① num = 0
② num = 0×2 + 1 = 1 ($2^0 = 1$)
③ num = 1×2 + 0 = 2
④ num = 2×2 + 1 = 5 ← 101 = 5

```
if (head == null)
    return -1

temp = head
num = 0

while (temp != null)
    if (temp.value > 1)
        return -1
    num = num * 2 + temp.value
    temp = temp.next

return num
```

1101 = 13
0×2 + 1 = 1
1×2 + 1 = 3
3×2 + 0 = 6
6×2 + 1 = 13

10 10 = 10
0×2 + 1 = 1
1×2 + 0 = 2
2×2 + 1 = 5
5×2 + 0 = 10

Return 0 instead of -1

```
temp = head
num = 0

while (temp != null)
    if (temp.value > 1)
        return 0
    num = num * 2 + temp.value
    temp = temp.next
```

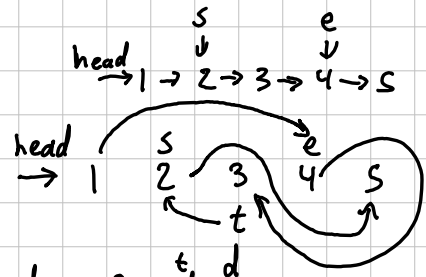
```
public void reverseTwoNodes(int m, int n)
```

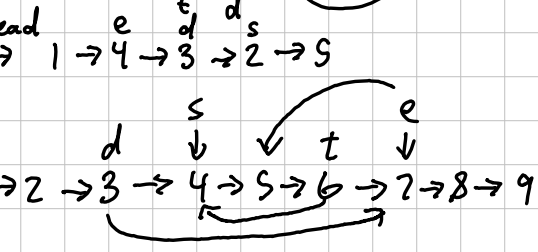
ØTail

1 → 2 → 3 → 4 → 5 rB(1,3)

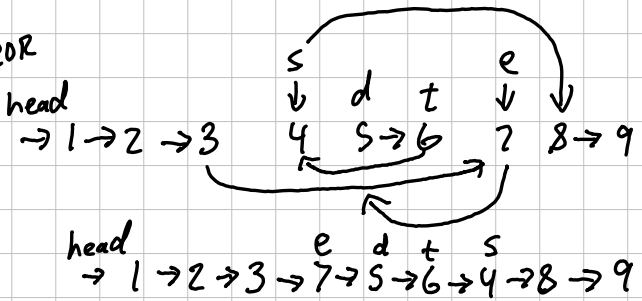
OUT: 1 → 4 → 3 → 2 → 5

```
if(head==null || m<0 || n<0 || m>= length || n >= length || m==n)
    return
 startIndex = head
 endIndex  = head
 if ( m<n)
for (int i=0; i<m ; i++)
        s = s.next
for (int i=0; i<n ; i++)
        e = e.next
else
 for (int i=0; i<n ; i++)
        s = s.next
 for (int i=0; i<m; i++)
        e = e.next
d=head
while(d.next!=s)
   d=d.next
t=s
while (t.next !=e)
    t =t.next
d.next=e
t.next=s
d=s.next
s.next=e.next
e.next=d
```





ERROR



ERROR (0, 4)

cannot read "next" because "d" is null

public void reverseBetween(int m, int n)

head → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 rB(4, 7)

OUT: 1 → 2 → 3 → 4 → 8 → 7 → 6 → 5 → 9

head → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 rB(3, 7)

OUT: 1 → 2 → 3 → 8 → 7 → 6 → 5 → 4 → 9

```
if(head==null||head.next==null)
   return
dummy = new Node(0)
dHead = dummy
 dummy.next = head
 temp = head
 end = dHead
for(int i=0; i<m ; i++)
   dummy = dummy.next
head = dummy.next
for(int i=0; i<n; i++)
    end = end.next
temp = end.next
 • • •
```

dH → 0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 (d, h, e, t)

dH → 0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 (d, h, e, t)

rB(3,7)

head → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 (start, mI, nI, end)

head → 1 → 2 → 3 → (start) (end) 9

```
if(head==null||head.next==null||n>m)
   return
dummy = new Node(0)
start = dummy → dummy.next = head
for(int i=0; i<m ; i++)
  start = start.next
mI = start.next
 nI = head
for(int i=0; i<n ; i++)
  nI = nI.next
 end = nI.next
```

```
start.next = null
nI.next = null
      p = null
      t = mI
      a = t.next
```

p t a

4 → 5 → 6 → 7 → 8 →

mI nI

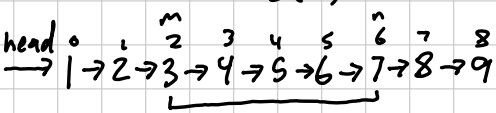
```
for(int i=0; i<n-m ; i++)
      a = t.next
      t.next = p
      p = t
      t = a
Start.next = nI
mI.next = end
dummy.next = null
```

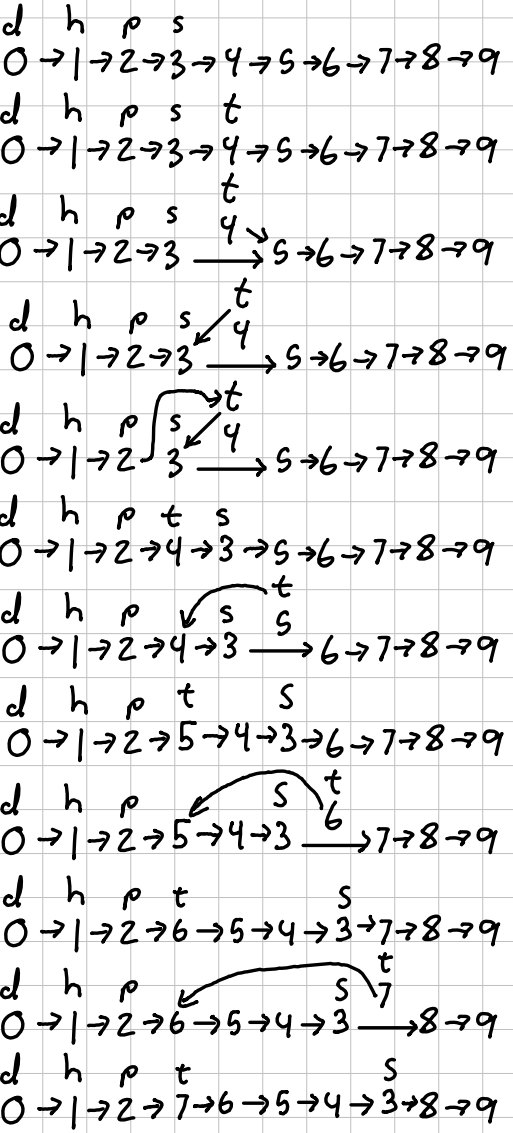
- ✗ Reverse all elements
- ✗ Reverse Partial List
- ✗ Reverse Single Swap

Understanding

```
public void reverseBetween(int m, int n)
if(head==null||m>n)
  return

dummy = new Node(0)
dummy.Next = head

Node p = dummy
for(int i=0; i<m; i++)
   p = p.next

Node start = p.next

for(int i=0; i<n-m; i++)
   Node temp = s.next
   s.next = temp.next
   temp.next = p.next
   p.next = temp

head = dummy.next
dummy.next = null
```

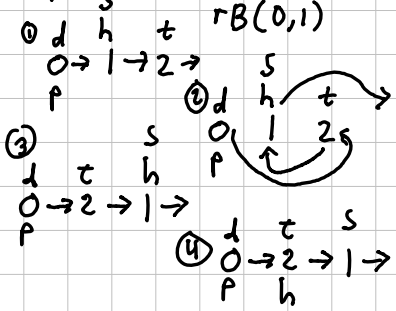
rB(2,6)





rB(0,0)

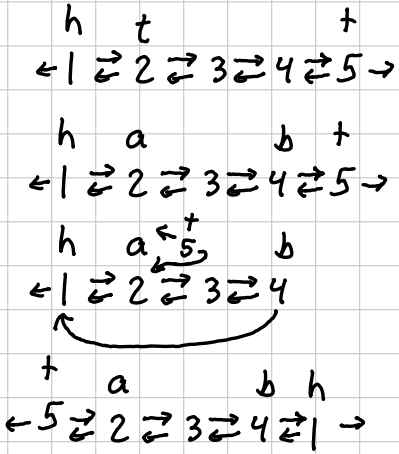
rB(0,1)



public void swapFirstLast()

head → 1 - 2 - 3 - 4 - 5 ← tail

Result: head → 5 - 2 - 3 - 4 - 1 ← tail




```
if (length == 0 || length == 1)
    return

a = head.next
b = tail.prev

tail.next = a
tail.prev = null
b.next = head
a.prev = tail
head.next = null
head.prev = b

a = tail
tail = head
head = a
```

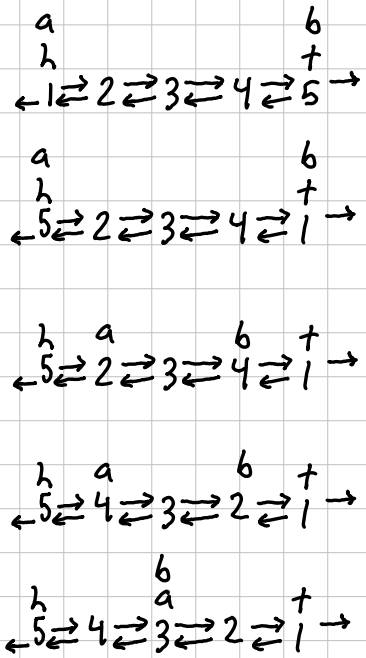
```
int temp = head.value

head.value = tail.value

tail.value = temp
```

public void reverse()

head → 1-2-3-4-5 ← tail

Result: head → 5-4-3-2-1 ← tail

```
if (length < 2)
    return

a = head
b = tail
for (int i=0; i<length/2; i++)

    int temp = a.value
    a.value = b.value
    b.value = temp

    a = a.next
    b = b.prev
```



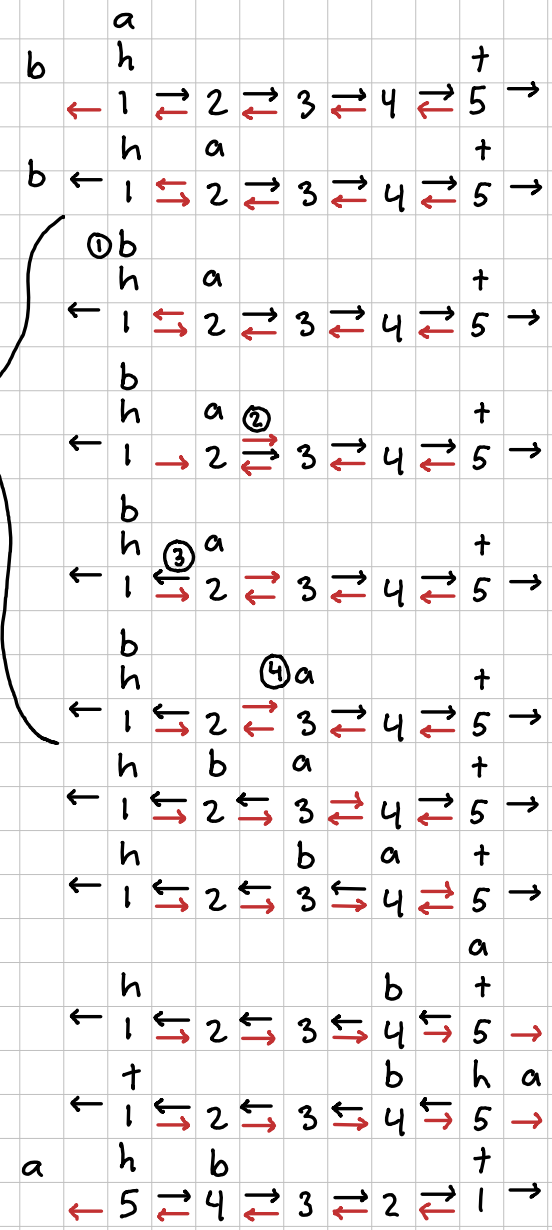
public void reverse()

Ø not swapping values of nodes

head → 1 - 2 - 3 - 4 - 5 ← tail

Result: head → 5 - 4 - 3 - 2 - 1 ← tail

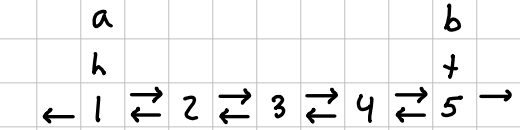
```
if (length < 2)
    return

a = head
b = null

while (a != null)
  ① b = a.prev
  ② a.prev = a.next
  ③ a.next = b
  ④ a = a.prev

temp = head
head = tail
tail = temp
```

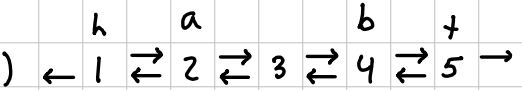


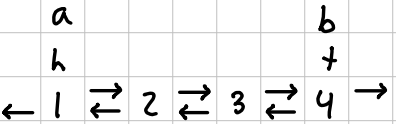
public boolean isPalindrome()

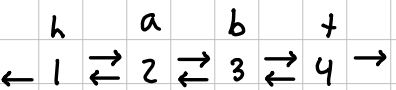
head → 1 - 2 - 3 - 4 - 5 ← tail

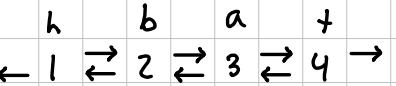
```
if (length <2)
   return true

Node a=head
Node b=tail

for(int i =0;i<(length/2);i++)

   if (a.value != b.value)
      return false

   a = a.next
   b = b.prev

return true
```











public void swapPairs()

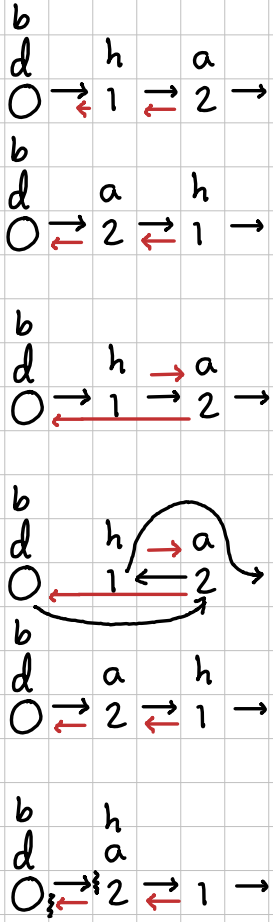
head → 1 - 2 - 3 - 4

Result: head → 2 - 1 - 4 - 3

Ø Tail

Ø change values

```
if (length < 2)
    return

dummy = new Node(0)
dummy.next = head

b = dummy

    a = head.next
  head.prev = a
   a.prev = b
  head.next = a.next
   a.next = head
   b.next = a

  head = dummy.next
 head.prev = null
 dummy.next = null
```



```
public void swapPairs()
```

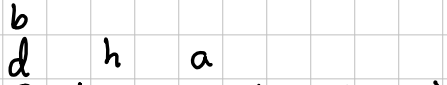
head → 1 - 2 - 3 - 4

Result: head → 2 - 1 - 4 - 3

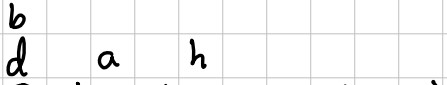
Ø Tail

Ø change values

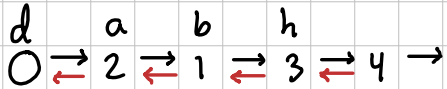
```
if (length <2)
    return
```



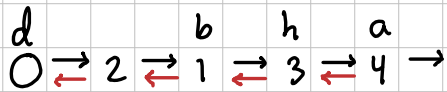

```
dummy =new Node(0)
dummy.next = head
```

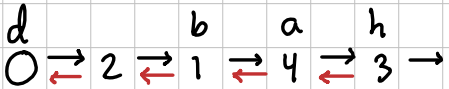



```
b =dummy
```




```
while (head !=null && head.next !=null)
    a =head.next
    head.prev = a
    a.prev = b
    head.next =a.next
*   if (head.next !=null)
        head.next.prev =head
    a.next =head
    b.next = a
    b =head
    head = b.next
```






```
head = dummy.next
head.prev = null
dummy.next =null
```

*Forgot to attach the prev of node after the current pair after pair swapped

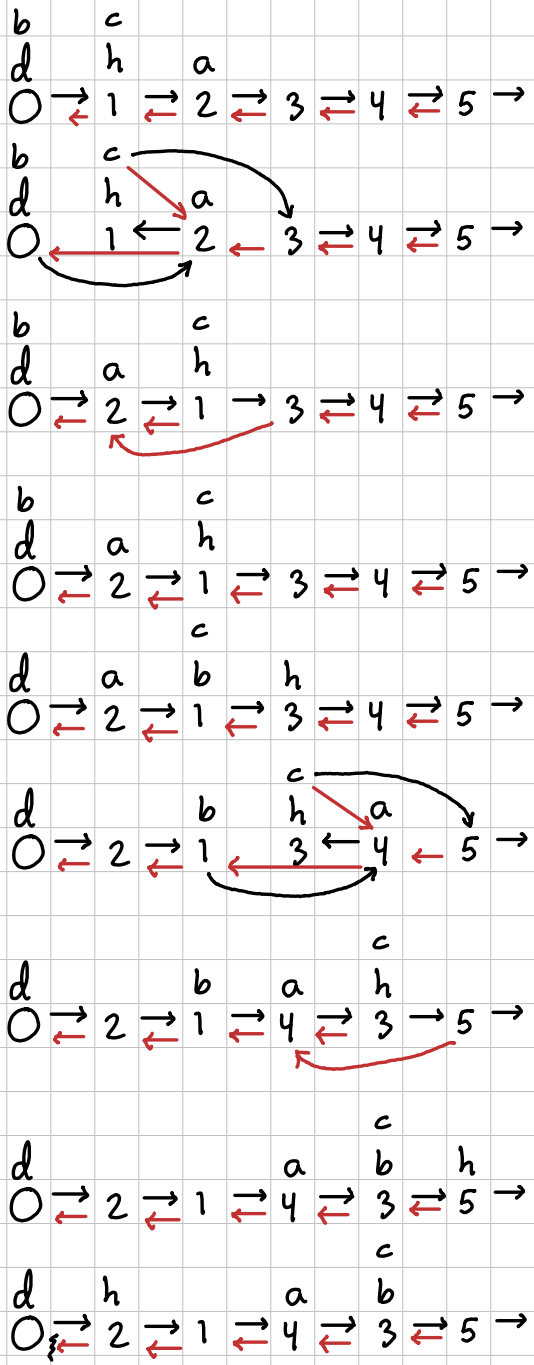
Not passing odd number of nodes tests despite working as seen in output

*Corrected

Understanding

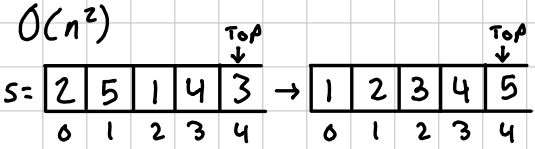
Public void swapPairs()

```
dummy = new Node(0)
dummy.next = head
b = dummy

while (head!=null && head.next!=null)
    c = head
    a = head.next

    b.next = a
    c.next = a.next
    a.next = c
    a.prev = b
    c.prev = a

    if (c.next!=null)
        c.next.prev = c

    head = c.next
    b = c

head = dummy.next
if (head!=null)
  head.prev = null
```

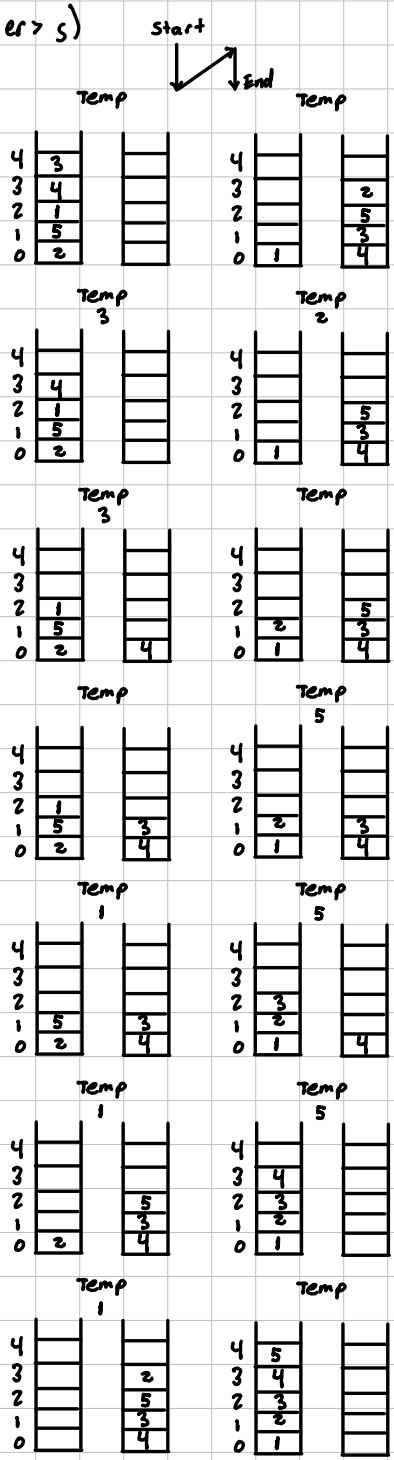


public static void sortStack(Stack<Integer> s)

$O(n^2)$

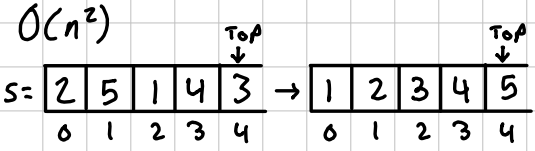



```
if (s.isEmpty)
    return

Stack<Integer> oS = new Stack<>()
temp = s.pop()

while(!s.isEmpty)
    if (s.peek() > temp)  X
        oS.push(s.pop())
    else
        oS.push (temp)
        temp = s.pop()

while(!oS.isEmpty)
    if(oS.peek() > temp)  X
        s.push(temp)
        temp = oS.pop()
    else
        s.push(oS.pop())

s.push (temp)
```



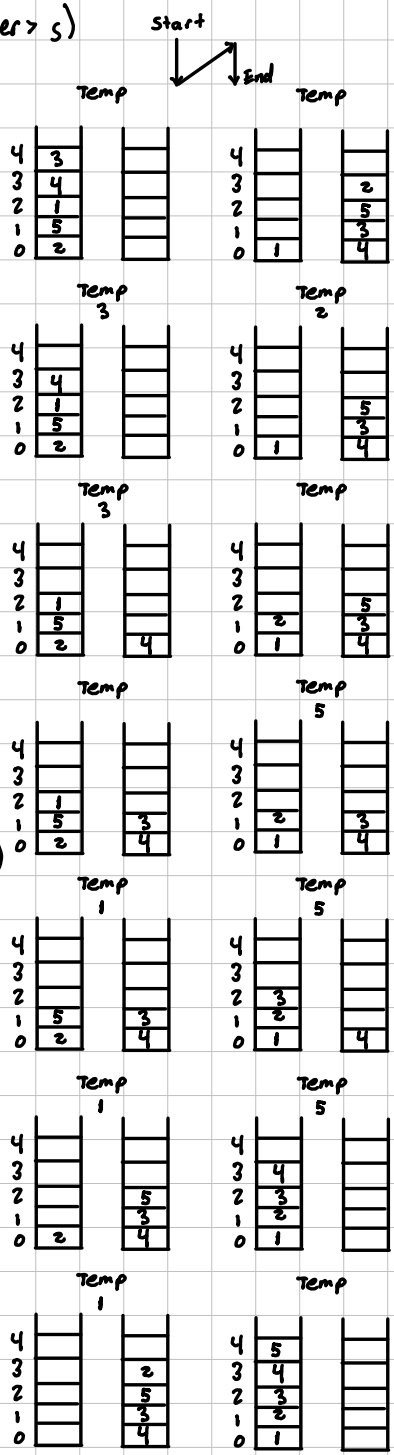
public static void sortStack(Stack<Integer> s)

$O(n^2)$



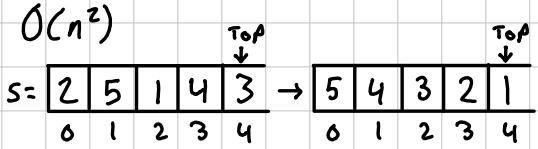

```
if(s.isEmpty())
    return

Stack<Integer> oS = new Stack<>()

while(!s.isEmpty())
    int temp = s.pop()
    while(!s.isEmpty() && s.peek()>temp)
        oS.push(s.pop())
    oS.push(temp)

while(!oS.isEmpty())
    int temp = s.pop()
    while(!oS.isEmpty() && oS.peek()<temp)
        s.push(oS.pop())
    s.push(temp)
```

Incomplete ✗

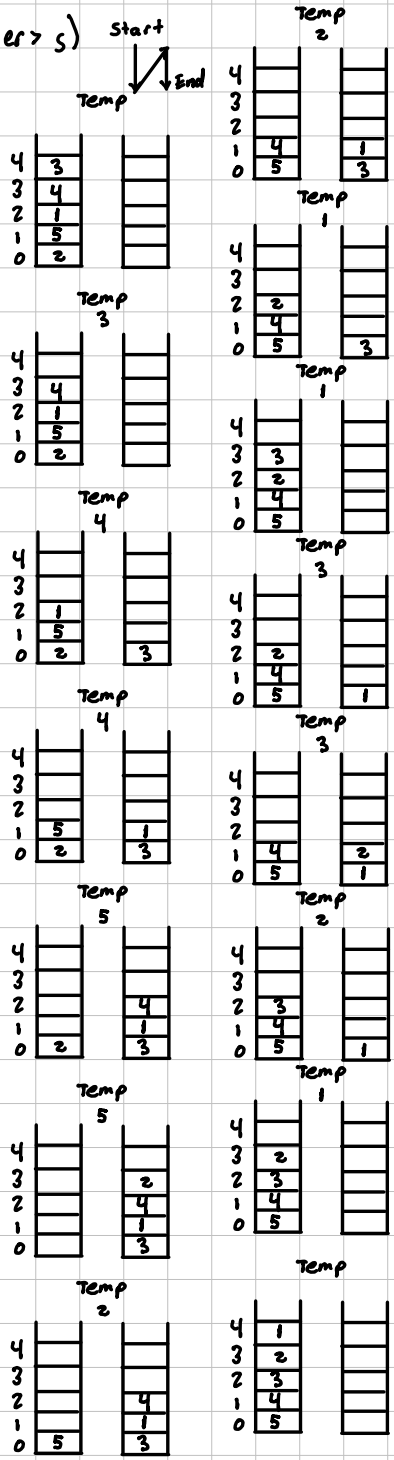


public static void sortStack(Stack<Integer> s)

$O(n^2)$

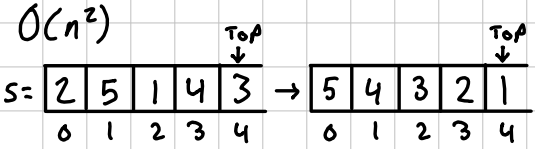



```
if( s.isEmpty)
    return

while(!s.isEmpty)
    int temp = s.pop()
    while(!s.isEmpty && s.peek<temp)
        os.push(s.pop)
    os.push(temp)

while(!os.isEmpty)
    int temp = os.pop()
    while(!os.isEmpty && os.peek>temp)
        s.push(os.pop)
    s.push(temp)

while(!s.isEmpty)
    int temp = s.pop()
    while(!s.isEmpty && s.peek<temp)
        os.push(s.pop)
    os.push(temp)

while(!os.isEmpty)
    int temp = os.pop()
    while(!os.isEmpty && os.peek>temp)
        s.push(os.pop)
    s.push(temp)
```



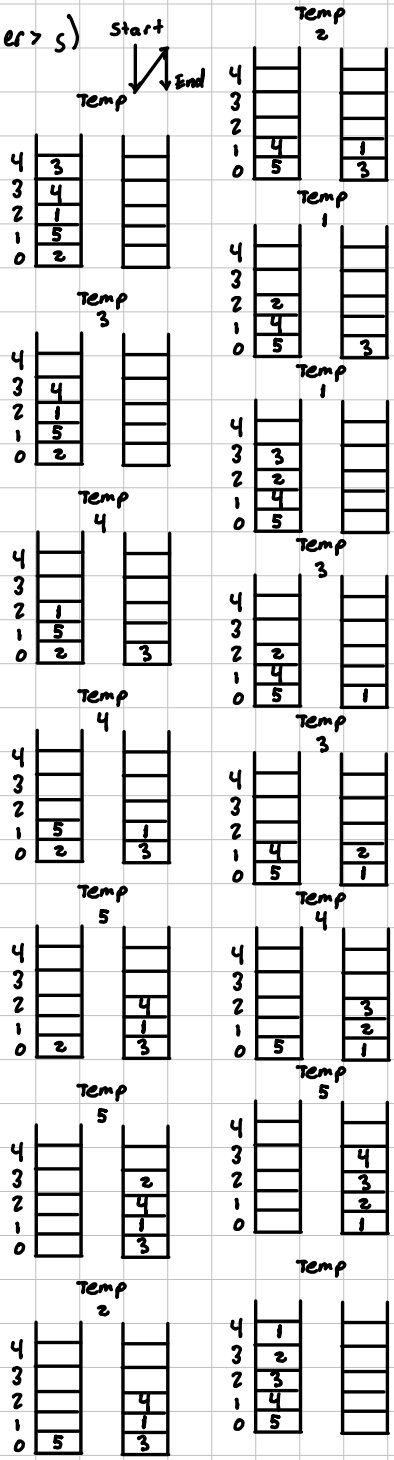
public static void sortStack(Stack<Integer> s)

$O(n^2)$




```
for(int i=0; i < s.size()/2; i++)

    int temp = s.pop()

    while(!s.isEmpty)

        if(!s.isEmpty && s.peek >temp)
            oS.push(temp)
            temp = s.pop()
        else
            oS.push(s.pop)

    while(!oS.isEmpty)

        if(!oS.isEmpty && oS.peek <temp)
            s.push(temp)
            temp = oS.pop()
        else
            s.push(oS.pop)

    s.push(temp)
```

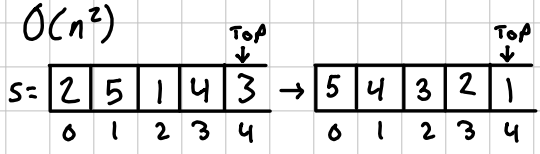
Seems to work



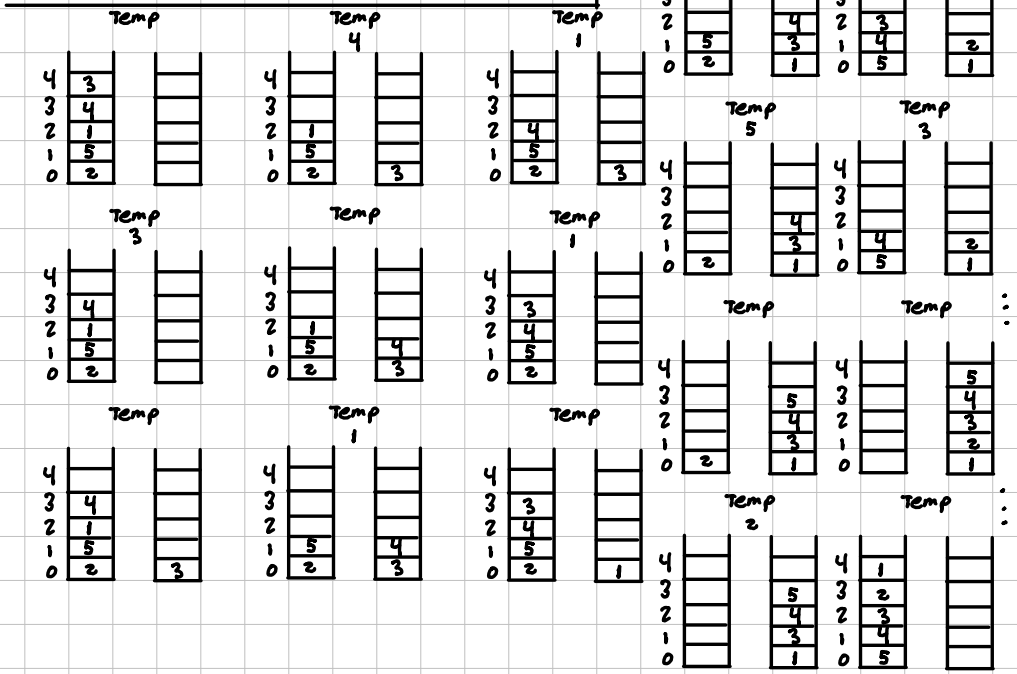
* Understanding

public static void sortStack(Stack<Integer> s)

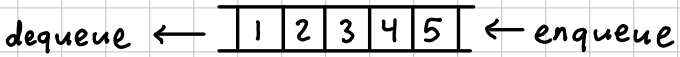
$O(n^2)$

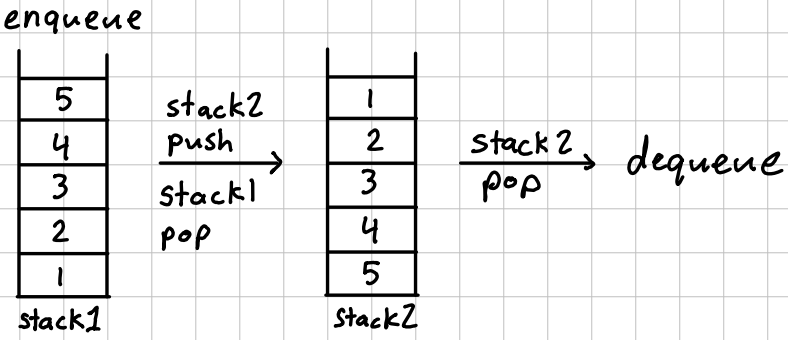



```
Stack<Integer> oS = new Stack<>()

while(!s.isEmpty)
    int temp = s.pop()
    while(!oS.isEmpty && oS.peek>temp)
        s.push(oS.pop)
    oS.push(temp)

while(!oS.isEmpty)
    s.push(oS.pop)
```



Queue using two stacks

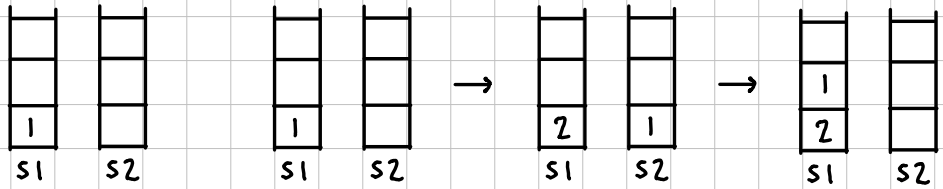




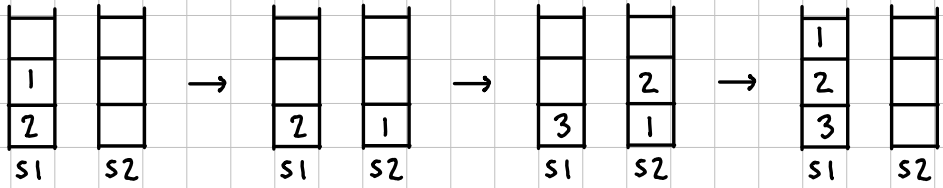
Enqueue : 1, 2, 3

Dequeue → s1.pop()

Enqueue(1) Enqueue(2)



Enqueue(3)



```
public static List<Integer> findDuplicates(int[] nums)
```

[1,2,3] → [] [3,2,1,3] → [3] [2,2,4,4,4] → [2,4]

```
HashMap<Integer,Integer> map = new HashMap<>()
List <Integer> list = new ArrayList<>()

 for (int i : nums)

    if (map.containsKey(i))
        map.put(i, map.get(i)+1)

    else
        map.put(i, 1)

for(Integer j : map.keySet())

    if(map.get(j) > 1)
        list.add(j)

return list
```

```
public static Character firstNonRepeatingChar (String s)

 leetcode → l     hello → h     aabbcc → null

HashMap<Character, Boolean> map = new HashMap<>()

for (int i = 0; i < s.length(); i++)

  char c = s.charAt(i)

     if (map.containsKey(c))
         map.put(c, true)

     else
         map.put(c, false)

  for(int i = 0; i < s.length(); i++)

     char ch = s.charAt(i)

     if (map.get(ch) == false)
          return ch

return null
```

```
public static List<List<String>> groupAnagrams (String[] strings)

[eat, tea, tan, ate, nat, bat] → [ [eat, tea, ate], [tan, nat], [bat] ]

HashMap<List<Character>, List<String>> mapLists = new HashMap<>()

for(String s : strings)

    List <Character> cList = new ArrayList<>()

    for(int i=0; i<s.length(); i++)
        char c = s.charAt(i)
        cList.add(c)

    Collections.sort (cList)

    List <String> sList = new ArrayList<>()

     if (mapLists.containsKey(cList))
          sList= mapLists .get(cList)
          sList. add (s)
          mapLists.put (cList, sList)
     else
          sList.add(s)
          mapList.put (cList, sList)

List<List<String>> list =new ArrayList<> (mapLists.values())

return list
```

public static int[] twoSum(int[] nums, int target)

[2,7,11,15] t=9 → [0,1] [3,3] t=6 → [0,1] [] t=0 → []
[3,2,4] t=6 → [1,2] [1,2,3,4,5] t=10 → []

```
HashMap<Integer, Integer> map = new HashMap<>()

for (int i=0; i < nums.length; i++)

    if(map.containsKey(target - nums[i]))

        return new int[] {map.get(target-nums[i]), i}

    map.put(nums[i], i)

return new int[] {}
```

public static int[] subarraySum(int[] nums, int target)

[1,2,3,4,5] t=9 → [1,3]
[2,3,4,5,6] t=3 → [1,1]
[-1,2,3,-4,5] t=0 → [0,3]
[ ] t=0 → [ ]

1 2 3 4 5 (9)

① sum = 1 < target, sum + next < target
② sum = 3 < target, sum + next < target
③ sum = 6 < target, sum + next > target
④ sum = 11 > target, sum ≠ target

① sum=2 < target, sum + next < target
② sum = 5 < target, sum + next < target
③ sum = 9 = target, return subarray

-1 2 3 -4 5 (0)

① sum = -1 < target, sum + next > target
② sum = 1 > target, sum + left < target
③ sum = 4 > target, 0 - 4 = -4 → complement exist after
④ sum=0, return subarray

✗ Incomplete and found no consistent logic

public static int[] subarraySum(int[] nums, int target)

[1,2,3,4,5] t=9 → [1,3]
[2,3,4,5,6] t=3 → [1,1]

[-1,2,3,-4,5] t=0 → [0,3]
[ ] t=0 → [ ]

1 2 3 4 5 (9)

```
HashMap<Integer, Integer> map = new HashMap<>()
for (int i=0; i<nums.length; i++)

    if (map.containsKey(target ))
        return new int[] {i,i}

  map.put (nums[i], i)


for(int i=0; i<nums.length; i++)
```

Incomplete

```
public static int[] subarraySum(int[] nums, int target)
```

[1,2,3,4,5] t=9 → [1,3]
[2,3,4,5,6] t=3 → [1,1]
[-1,2,3,-4,5] t=0 → [0,3]
[] t=0 → []

```
HashMap<Integer, Integer> map = new HashMap<>()
                sum      index
int sum = 0

map.put(0,-1)

for (int i=0; i<nums.length; i++)

    sum += nums[i]

    if (map.containsKey(sum-target)
        return new int[] { map.get(sum-target)+1 , i}

    map.put(sum, i)

return new int[] {}
```

1 2 3 4 5 (9)

① sum=1, 1-9=-8 DNE
② map(1,0)
③ sum=3, 3-9=-6 DNE
④ map(3,1)
⑤ sum=6, 6-9=-3 DNE
⑥ map(6,2)
⑦ sum=10, 10-9=1
⑧ map.get(1)=0+1=1, i=3 => {1,3}

-1 2 3 -4 5 (0)

① sum=-1, -1-0=-1 DNE
② map(-1,0)
③ sum=1, 1-0=1 DNE
④ map(1,1)
⑤ sum=4, 4-0=4 DNE
⑥ map(4,2)
⑦ sum=0, 0-0=0
⑧ map.get(0)=-1+1=0, i=3 => {0,3}

```
public static List<Integer> removeDuplicates(List<Integer> myList)
```

[1,2,3,4,1,2,5,6,7,3,4,8,9,5] → [1,2,3,4,5,6,7,8,9]

```
HashSet<Integer> set = new HashSet<>()
for (Integer i : myList)
    set.add(i)
List<Integer> list = new ArrayList<>(set)
return list
```

```
HashSet<Integer> set = new HashSet<>(myList)
return new ArrayList<>(set)
```

```
public static boolean hasUniqueChars (String s)

 HashSet<Character> set = new HashSet<>()

for (int i=0; i<s.length(); i++)

    char c = s.charAt(i)

    if (!set.add(c))
        return false
    else
        set.add(c)

return true
```

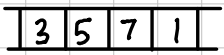
```
public static List<int[]> findPairs (int[] arr1, int[] arr2, int target)
```

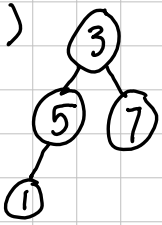
arr1: [1,2,3,4,5] ⟶ [5,2] [3,4] [1,6]
arr2: [2,4,6,8,10] t = 7

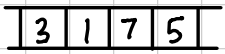
```
HashSet < Integer> set  = new HashSet<>()
List <int []> pairs  = new ArrayList<>()

for(int i:arr1)

    set.add(i)

for(int j : arr2)

   if(set.contains(target-j)

       pairs.add(new int[]{ target-j, j })

return pairs
```

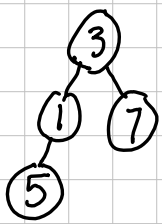
```
public static int longestConsecutiveSequence(int[] nums)

[100,4,200,1,3,2] => 4 ([4,3,2,1])

HashSet<Integer> set = new HashSet<>()

for(int i : nums)

    set.add(i)

int max = 0
int count = 0


for (Integer j : set)

    if(!set.contains(j-1))
        int current = j
        count = 1
        while(set.contains(current+1))
            current++
            count++

        if(count > max)
            max = count

return max
```

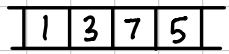
```
public void insertMin(int value)
    heap.add(value)
    int current = heap.size() - 1
while(current > 0 &&
        heap.get(current) < heap.get(parent(current)))
    swap(current, parent(current))
    current = parent(current)
```

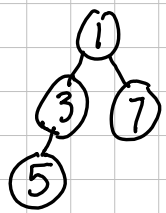






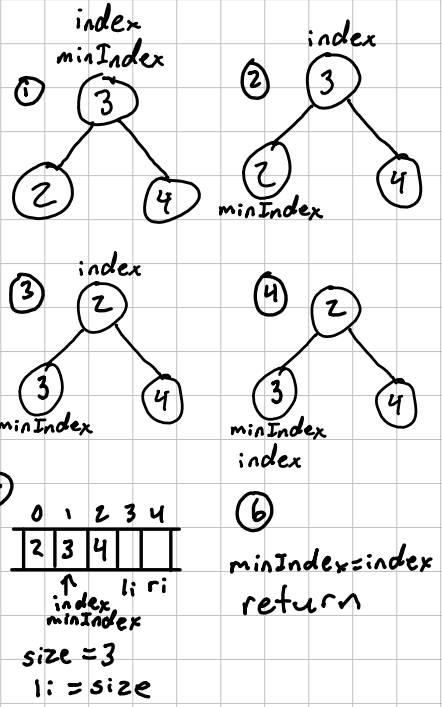






```
public Integer removeMin()
 if (heap.size() == 0)
     return null
 if (heap.size == 1)
     return heap.remove(0)

 int minValue = heap.get(0)
 heap.set(0, heap.remove(heap.size()-1))
   sinkDown(0)
  return   minValue
```

```
public void sinkDownMin(int index)

  int minIndex = index

while (true)

    int leftIndex = leftChild(index)
    int rightIndex = rightChild(index)

    if( leftIndex <heap.size() &&
       heap.get(leftIndex) < heap.get(minIndex))

        minIndex = leftIndex

    if( rightIndex <heap.size() &&
       heap.get(rightIndex) < heap.get(minIndex))

        minIndex = rightIndex

    if(minIndex != index)

        swap(index, minIndex)

        index = minIndex

    else

        return
```



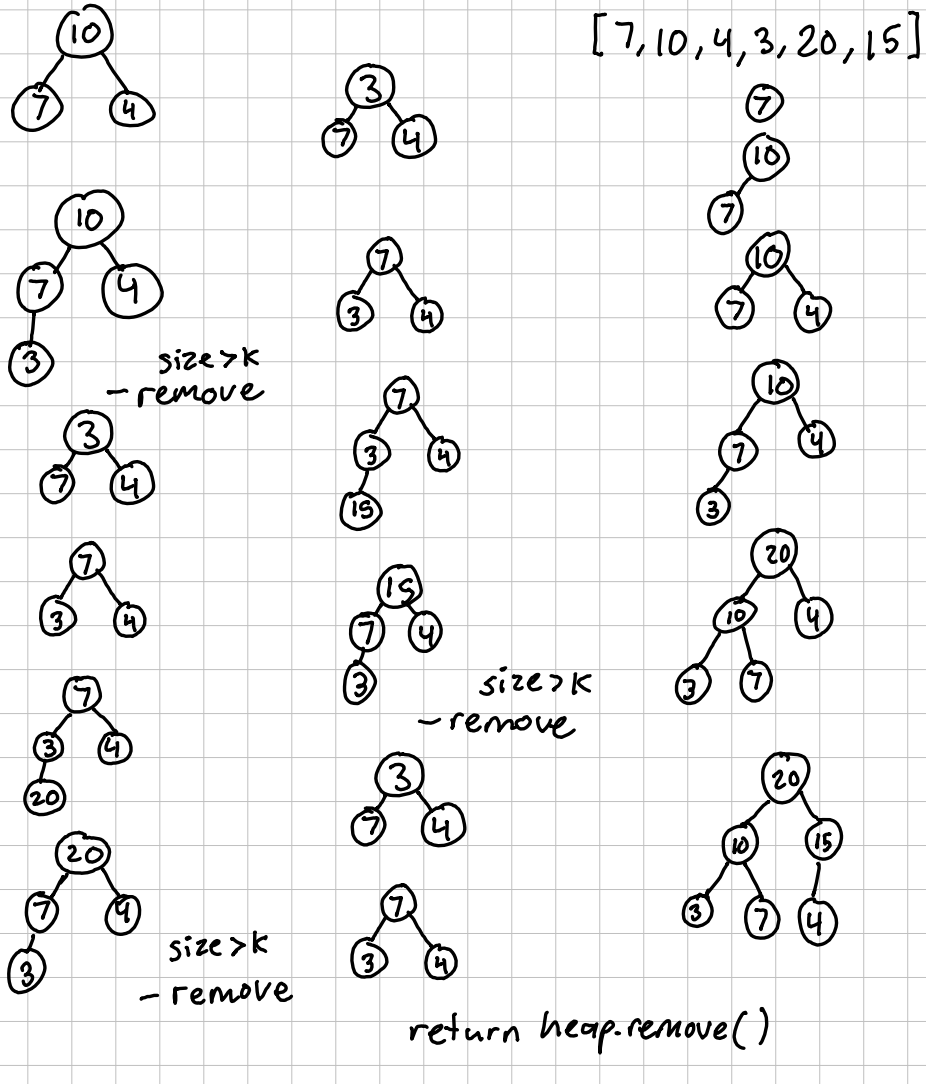
public static int findKthSmallest(int[] nums, int k)

nums → +, -, any size, can contain duplicates

k → $1 \le k \le$ nums.length, position to return

use maxHeap, O(n log n)

[7, 10, 4, 3, 20, 15] , k=3 → 7 ([3, 4, 7, 10, 15, 20]) — 3rd smallest number



public static int findKthSmallest(int[] nums, int k)

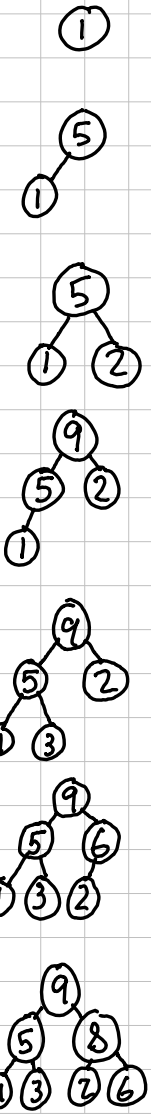
nums → +, -, any size, can contain duplicates
k → 1 ≤ k ≤ nums.length, position to return
use maxHeap, O(n log n)
[7, 10, 4, 3, 20, 15], k=3 → 7 (3rd smallest number)
([3, 4, 7, 10, 15, 20] — 1, 2, 3, 4, 5, 6)

```
Heap maxHeap = new Heap()

for (int i : nums)

    maxHeap.insert(i)

    if (maxHeap.getHeap().size() > k)

        maxHeap.remove()

return maxHeap.remove()
```

public static List<Integer> streamMax (int[] nums)

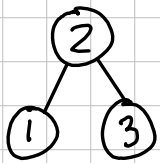
[1,5,2,9,3,6,8] → [1,5,5,9,9,9,9]
[10,2,5,1,0,11,6] → [10,10,10,10,10,11,11]
[3,3,3,3,3] → [3,3,3,3,3]

```
Heap maxHeap = new Heap()
List <Integer> list = new ArrayList<>()

for(int i :nums)

    maxHeap.insert(i)

    list.add (maxHeap.getHeap().get(0))

return list
```

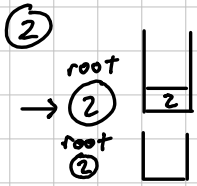


* Understanding

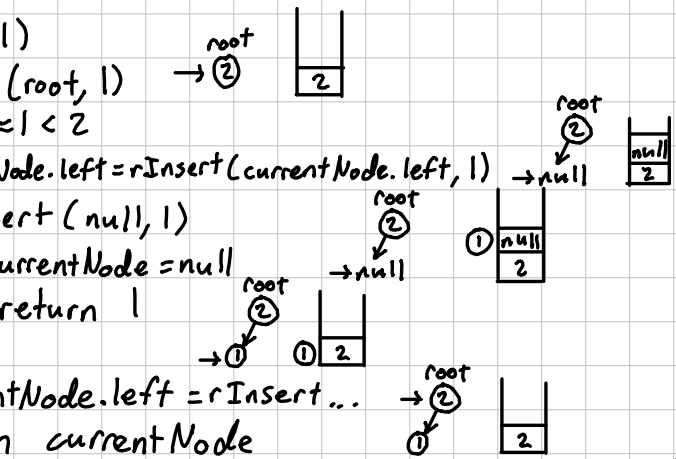
```
private Node rInsert(Node currentNode, int value)

  if (currentNode == null)
     return new Node(value)

  if (value < currentNode.value)
    currentNode.left = rInsert(currentNode.left, value)
  else if (value > currentNode.value)
    currentNode.right = rInsert(currentNode.right, value)

  return currentNode

public void rInsert(int value)

  if(root == null)
     root = new Node(value)

  rInsert(root, value)
```



rInsert(2)
root=null
root=2
rInsert(root,2)
return 2

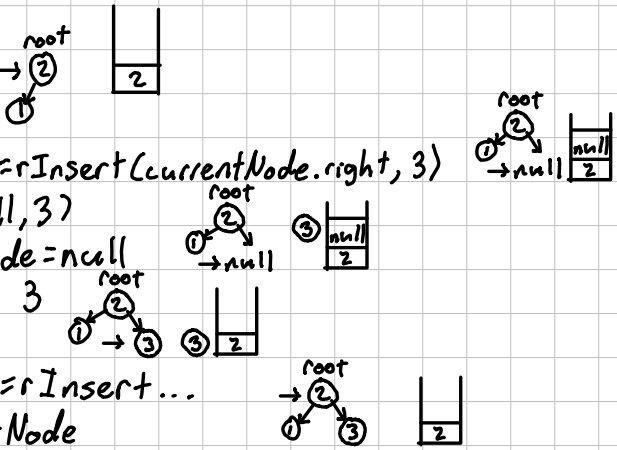


rInsert(1)
rInsert(root,1)
b value=1<2
currentNode.left=rInsert(currentNode.left,1)
rInsert(null,1)
a currentNode=null
return 1
a currentNode.left=rInsert...
return currentNode

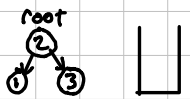


b since it's a void method, it does nothing

rInsert(3)
rInsert(root,3)
d value=3>2
currentNode.right=rInsert(currentNode.right,3)
rInsert(null,3)
c currentNode=null
return 3
c currentNode.right=rInsert...
return currentNode



d since it's a void method, nothing to return



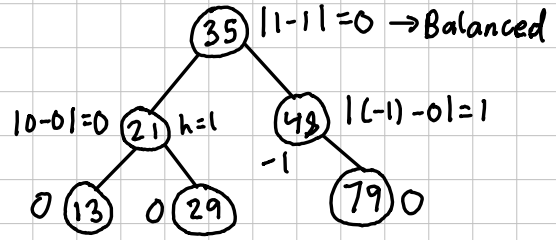
private Node sortedArrayToBST(int[] nums, int left, int right)

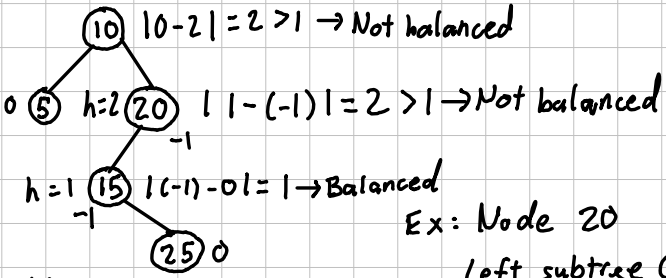
nums : sorted array
left : index to start the process
right : index to end the process

} use in public method (int[] nums)
this.root = method(nums, 0, nums.length-1)

- create BST → depth of two subtrees of any node does not differ by more than one
- return root node of BST created
- middle element of current segment is root
- recursive method builds left and right subtrees by calling itself with adjusted left and right indices to work on sub-segments of array

Difference of height = | height left - height right |





Ex: Node 15
Left subtree (none): h=-1
Right subtree (29): h=0
Difference: |(-1)-0|=1

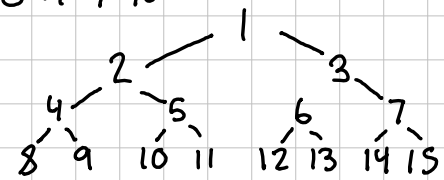
Ex: Node 20
Left subtree (15): h=1
Right subtree (null): h=-1
Difference: |1-(-1)|=2

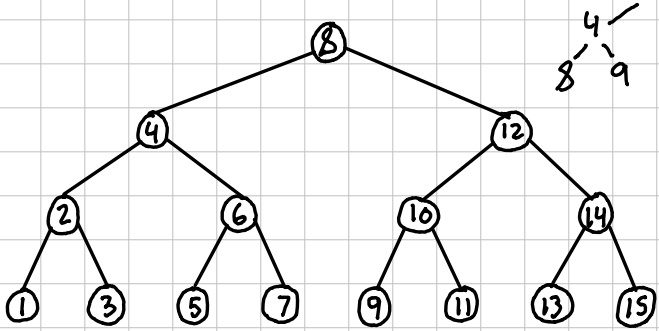
private Node sortedArrayToBST(int[] nums, int left, int right)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ODD

R

8 4 9 2 10 5 11 1 12 6 13 3 14 7 15

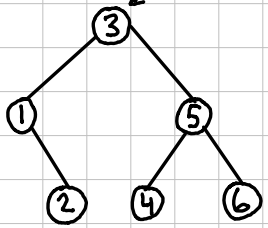




1 2 3 4 5 6 EVEN

R

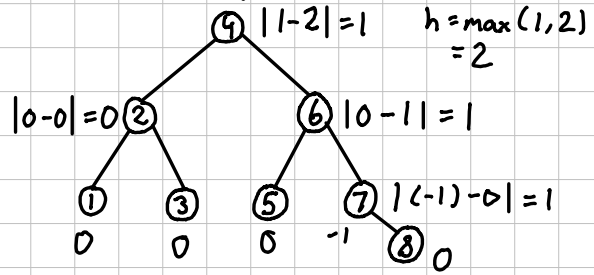
Node 1 & 5 : h = 1



1 2 3 4 5 6 7 8 EVEN

R

Node 2 & 6 : h = 1 &



```
private Node sortedArrayToBST(int[] nums, int left, int right)

    if (left > right)
      return null

    int mid = left + (right - left) / 2

    Node node = new Node(nums[mid])

    node.left = sortedArrayToBST(nums, left, mid-1)
    node.right = sortedArrayToBST(nums, mid+1, right)

    return node
```

* Understanding

private Node sortedArrayToBST(int[] nums, int left, int right)

1	2	3	4	5	6
0	1	2	3	4	5

BST(nums, 0, 5)

mid = 0 + (5-0)/2 = 2.5 = 2

node = 3

node.left = BST(nums, 0, 1)

mid = 0 + (1-0)/2 = 0.5 = 0

node = 1

node.left = BST(nums, 0, -1)

= null

node.right = BST(nums, 1, 1)

mid = 1 + (1-1)/2 = 1

node = 2

node.left = BST(nums, 1, 0)

= null

node.right = BST(nums, 2, 1)

= null

= 2

= 1

node.right = BST(nums, 3, 5)

mid = 3 + (5-3)/2 = 4

node = 5

node.left = BST(nums, 3, 3)

mid = 3 + (3-3)/2 = 3

node = 4

node.left = BST(nums, 3, 2)

= null

node.right = BST(nums, 4, 3)

= null

= 4

node.right = BST(nums, 5, 5)

mid = 5 + (5-5)/2 = 5

node = 6

node.left = BST(nums, 5, 4)

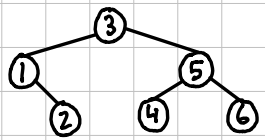
= null

node.right = BST(nums, 6, 5)

= null

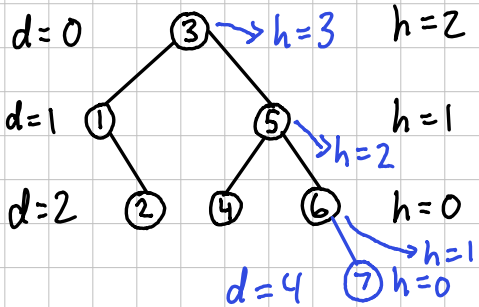
= 6

= 3



*Understanding

```
private int height (Node node)

  if (node == null)
      return 0
  int leftH = height(node.left)
  if (leftH == -1)
     return -1
  int rightH = height (node.right)
  if (rightH == -1)
     return -1
  if (Math.abs(leftH - rightH) > 1)
     return -1
  return 1 + Math.max(leftH, rightH)
```




```
public boolean isBalanced ()
   return height (this.root) != -1
```

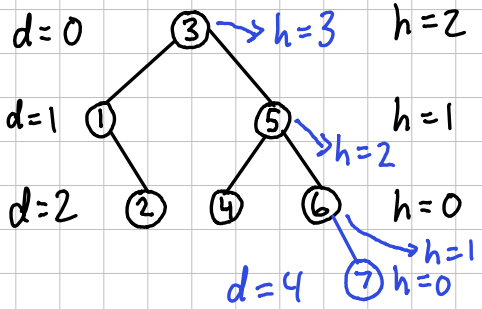
height(3)
- leftH = height(1)
  - leftH = height(null) = 0
  - rightH = height(2)
    - leftH = height(null) = 0
    - rightH = height(null) = 0
    - → = 1 + max(0,0) = 1
  - → = 1 + max(0,1) = 2
- rightH = height(5)
  - leftH = height(4)
    - leftH = height(null) = 0
    - rightH = height(null) = 0
    - → = 1 + max(0,0) = 1
  - rightH = height(6)
    - leftH = height(null) = 0
    - rightH = height(null) = 0
    - → = 1 + max(0,0) = 1
  - → = 1 + max(1,1) = 2
- → = 1 + max(2,2) = 3

*Check if balanced but the given height is false

Here it gives 3 but it should be 2 for root
Height of root node = 2
Depth of root node = 0 (depth of node = rank/level)

*Alternative of height and check height-balanced

```
private int height (Node node)

if (node == null)
    return -1

int leftH = height (node.left)

if

int rightH = height (node.right)
```

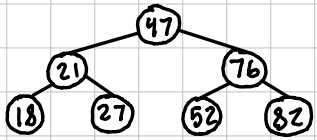


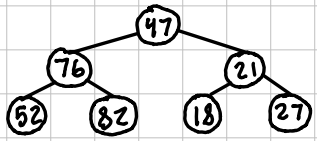
private Node invertTree(Node node)

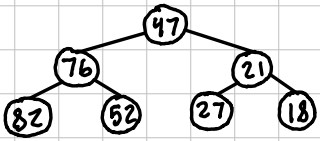
- inverts binary tree by swapping left and right children of all nodes recursively
- node: current node to proceed
- return: node after its subtree has been inverted
- base case: encounter null node
- temp node is used to facilitate swap of left and right children
- modify tree, not creating new tree
- method handles binary trees of any size and structure

```
public void invert()
    root = invertTree(root)
```

```
private Node invertTree(Node node)

if (node == null)
  return null

if (node.left != null || node.right != null)
   Node temp = node.left
    node.left = node.right
    node.right = temp
    node.left = iT(node.left)
    node.right = iT(node.right)

return node
```







---

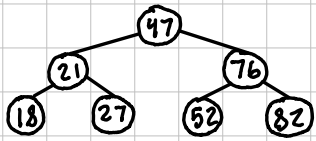
*Testing

iT(47)
- temp = 21
- node.left = 76
- node.right = 21
- node.left = iT(76)
  - temp = 52
  - node.left = 82
  - node.right = 52
  - node.left = iT(82) = 82
  - node.right = iT(52) = 52
  - = 76
- node.right = iT(21)
  - temp = 18
  - node.left = 27
  - node.right = 18
  - node.left = iT(27) = 27
  - node.right = iT(18) = 18
  - = 21
- = 47

Seems To Work

*Understanding

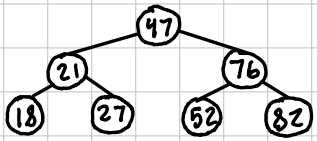
```
private Node invertTree(Node node)
  if(node==null)
     return null
  Node temp=node.left
  node.left =iT(node.right)
  node.right=iT(temp)
  return node
```

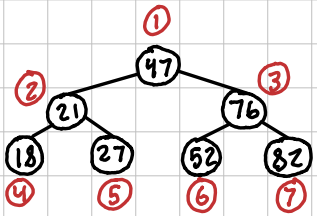



```
invertTree(47)
  temp=21
  node.left =iT(76)
    temp=52
    node.left=iT(82)
      temp=null
      node.left=iT(null)
                =null
      node.right=iT(null)
                 =null
               =82
    node.right=iT(52)
      temp=null
      node.left=iT(null)
                =null
      node.right=iT(null)
                 =null
               =52
  =76
node.right=iT(21)
  temp=18
  node.left=iT(27)
    temp=null
    node.left=iT(null)
              =null
    node.right=iT(null)
               =null
             =27
  node.right=iT(18)
    temp=null
    node.left=iT(null)
              =null
    node.right=iT(null)
               =null
             =18
=21
=47
```

*Understanding Breadth First Search (Level Order Traversal)

```
public ArrayList<Integer> BFS()
  current = root
  Queue<Node> queue = new LL<>()
  ArrayList<Integer> arr = new AL<>()
  queue.add(current)
  while (queue.size() > 0)
    current = queue.remove()
    arr.add(current.value)
    if(current.left != null)
       queue.add(current.left)
    if(current.right != null)
       queue.add(current.right)
  return arr
```





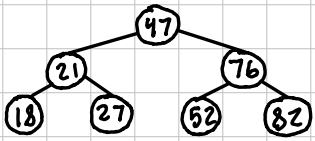
47	21	76	18	27	52	82

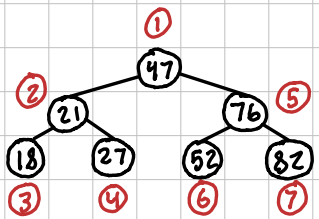
```
BFS()
 current = 47
 new queue, new arr
 queue.add(47)          q=[47]
while (queue.size()>0)
    current = 47        q=[]
    arr.add(47)         a=[47]
    queue.add(21)       q=[21]
    queue.add(76)       q=[21,76]
     current = 21       q=[76]
     arr.add(21)        a=[47,21]
     queue.add(18)      q=[76,18]
     queue.add(27)      q=[76,18,27]
    current = 76        q=[18,27]
    arr.add(76)         a=[47,21,76]
    queue.add(52)       q=[18,27,52]
    queue.add(82)       q=[18,27,52,82]
```

```
current = 18    q=[27,52,82]
arr.add(18)     a=[47,21,76,18]
current = 27    q=[52,82]
arr.add(27)     a=[47,21,76,18,27]
current=52      q=[82]
arr.add(52)     a=[47,21,76,18,27,52]
current=82      q=[]
arr.add(82)     a=[47,21,76,18,27,52,82]

=[47,21,76,18,27,52,82]
```

* Understanding Depth First Search - Pre Order

```
private void rDFSPreOrder(Node node, AL<Integer> r)
    if (node==null)
        return
    r.add(node.value)
    rDFSPrO(node.left, r)
    rDFSPrO(node.right, r)

public AL<Integer> rDFSPrO()
    AL<Integer> results=new AL<>()
    rDFSPrO(root, results)
    return results
```





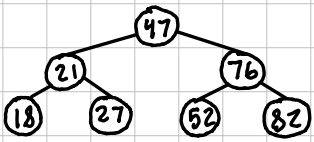
47	21	18	27	76	52	82

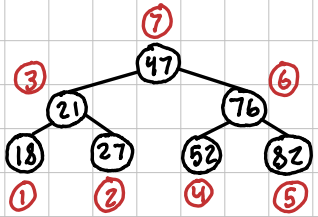
```
rDFSPrO(47, results)
↳ r.add(47)
  rDFSPrO(21, r)
  ↳ r.add(21)
    rDFSPrO(18, r)
    ↳ r.add(18)
      rDFSPrO(null, r)
      rDFSPrO(null, r)
    rDFSPrO(27, r)
    ↳ r.add(27)
      rDFSPrO(null, r)
      rDFSPrO(null, r)
  rDFSPrO(76, r)
  ↳ r.add(76)
    rDFSPrO(52, r)
    ↳ r.add(52)
      rDFSPrO(null, r)
      rDFSPrO(null, r)
    rDFSPrO(82, r)
    ↳ r.add(82)
      rDFSPrO(null, r)
      rDFSPrO(null, r)
```

= [47, 21, 18, 27, 76, 52, 82]

* Understanding Depth First Search - Post Order

```
private void rDFSPO(Node node, AL<Integer> r)
    if(node==null)
        return
    rDFSPO(node.left, r)
    rDFSPO(node.right, r)
    r.add(node.value)

public AL<Integer> rDFSPO()
    AL<Integer> results = new AL<>()
    rDFSPO(root, results)
    return results
```





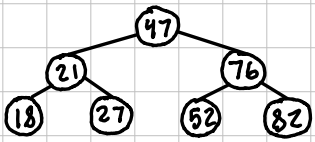
18	27	21	52	82	76	47

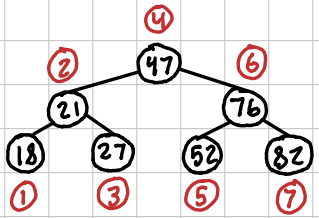
```
rDFSPO(47, results)
    rDFSPO(21)
        rDFSPO(18)
            rDFSPO(null)
            rDFSPO(null)
            r.add(18)
        rDFSPO(27)
            rDFSPO(null)
            rDFSPO(null)
            r.add(27)
        r.add(21)
    rDFSPO(76)
        rDFSPO(52)
            rDFSPO(null)
            rDFSPO(null)
            r.add(52)
        rDFSPO(82)
            rDFSPO(null)
            rDFSPO(null)
            r.add(82)
        r.add(76)
    r.add(47)
```

= [18, 27, 21, 52, 82, 76, 47]

Understanding Depth First Search - In Order

```
private void rDFSInO (Node node, AL<Integer> r)
    if(node==null)
        return
    rDFSInO (node.left, r)
    r.add(node.value)
    rDFSInO (node.right, r)

public AL<Integer> rDFSInO()
    AL<Integer> results = new AL<>()
    rDFSInO(root, results)
    return results
```



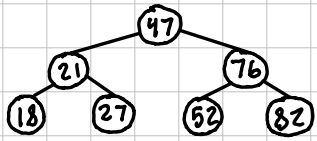


18	21	27	47	52	76	82

```
rDFSInO(47, results)
    rDFSInO(21, r)
        rDFSInO(18, r)
            rDFSInO(null, r)
            r.add(18)
            rDFSInO(null, r)
        r.add(21)
        rDFSInO(27, r)
            rDFSInO(null, r)
            r.add(27)
            rDFSInO(null, r)
    r.add(47)
    rDFSInO(76, r)
        rDFSInO(52, r)
            rDFSInO(null, r)
            r.add(52)
            rDFSInO(null, r)
        r.add(76)
        rDFSInO(82, r)
            rDFSInO(null, r)
            r.add(82)
            rDFSInO(null, r)

=[18, 21, 27, 47, 52, 76, 82]
```

public boolean isValidBST()

- checks if binary tree is a valid binary search tree
- valid if left child < parent < right child
- use in-order traversal to collect node values and then checks if list of values is in ascending order without duplicates
- assume DFSInO() to perform in-order traversal of the tree and returns an ArrayList of node values



In-Order →

18	21	27	47	52	76	82

18	21	27	47	52	76	82

↑
21 > 18

18	21	27	47	52	76	82

↑
76 > 52

18	21	27	47	52	76	82

↑
27 > 21

18	21	27	47	52	76	82

↑
82 > 76

18	21	27	47	52	76	82

↑
47 > 27

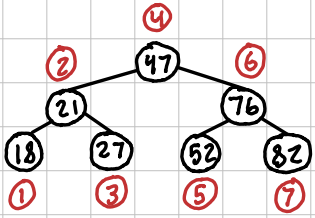
18	21	27	47	52	76	82

↑
52 > 47

```
ArrayList<Integer> results = DFSInO()

for(int i=1; i<results.size(); i++)

    if(results.get(i-1) >= results.get(i)
        return false

return true
```

public Integer kthSmallest(int k)

- use in-order traversal
- use stack<Node>, continue traversal until empty stack or null node
- while current node is not null, push it onto stack and move pointer to its left child.
- once current node is null, pop a node from stack, decrement k by one and if k=0, return value of popped node
- Move pointer to right child of popped node
- $0 < k \leq$ number of elements in the tree
- insert(int value) is called at least once before kthSmallest(int k)



18	21	27	47	52	76	82

```
Stack<Node> s=new Stack
Node c=root
while(c!=null || !s.isEmpty())
  while(c!=null)
     s.push(c)
     c=c.left
  c=s.pop()
  k--
  if(k==0)
     return c.value
  c=c.right
return null
```

in/out

k=5, s=[]↵
c=root
  s=[47,21,18]
  c=null(L)
c=18, k=4
c=null(R)
s=[47,21]
c=21, k=3
c=27(R)
  s=[47,27]
  c=null(L)
c=27, k=2
c=null(R)
s=[47]
c=47, k=1
c=76
  s=[76,52]
  c=null(L)
c=52, k=0
return 52

*Understanding

```
public static void bubbleSort(int[] a)
    for(int i=a.length-1; i>0; i--)
        for(int j=0; j<i; j++)
            if(a[j]>a[j+1])
                int temp=a[j]
                a[j]=a[j+1]
                a[j+1]=temp
```

	0	1	2	3	4	5
a =	4	2	6	5	1	3

Time Complexity = $O(n^2)$

Space Complexity = $O(1)$

i=5

j=0, a[0]>a[1]

4	2	6	5	1	3
0	1	2	3	4	5
2	4	6	5	1	3

j=1, a[1]<a[2]

2	4	6	5	1	3
0	1	2	3	4	5
2	4	6	5	1	3

j=2, a[2]>a[3]

2	4	6	5	1	3
0	1	2	3	4	5
2	4	5	6	1	3

j=3, a[3]>a[4]

2	4	5	6	1	3
0	1	2	3	4	5
2	4	5	1	6	3

j=4, a[4]>a[5]

2	4	5	1	6	3
0	1	2	3	4	5
2	4	5	1	3	6

i=4

j=0, a[0]<a[1]

2	4	5	1	3	6
0	1	2	3	4	5
2	4	5	1	3	6

j=1, a[1]<a[2]

2	4	5	1	3	6
0	1	2	3	4	5
2	4	5	1	3	6

j=2, a[2]>a[3]

2	4	5	1	3	6
0	1	2	3	4	5
2	4	1	5	3	6

j=3, a[3]>a[4]

2	4	1	5	3	6
0	1	2	3	4	5
2	4	1	3	5	6

i=3

j=0, a[0]<a[1]

2	4	1	3	5	6
0	1	2	3	4	5
2	4	1	3	5	6

j=1, a[1]>a[2]

2	4	1	3	5	6
0	1	2	3	4	5
2	1	4	3	5	6

j=2, a[2]>a[3]

2	1	4	3	5	6
0	1	2	3	4	5
2	1	3	4	5	6

i=2

j=0, a[0]>a[1]

2	1	3	4	5	6
0	1	2	3	4	5
1	2	3	4	5	6

i=1

j=0, a[0]<a[1]

1	2	3	4	5	6
0	1	2	3	4	5
1	2	3	4	5	6

j=1, a[1]<a[2]

1	2	3	4	5	6
0	1	2	3	4	5
1	2	3	4	5	6

* Understanding

```
public static void selectionSort(int[] a)
    for(int i=0; i<a.length; i++)
        int minIndex = i
        for(int j = i+1; j<a.length; j++)
            if (a[j] < a[minIndex])
                minIndex = j
        if(i != minIndex)
            int temp = a[i]
                a[i] = a[minIndex]
            a[minIndex] = temp
```

	0	1	2	3	4	5
a =	4	2	6	5	1	3

Time Complexity = $O(n^2)$

Space Complexity = $O(1)$

minIndex = 0 → 4

4	2	6	5	1	3
0	1	2	3	4	5
1	2	6	5	4	3

minIndex = 4 → 4

1	2	3	4	5	6
0	1	2	3	4	5
1	2	3	4	5	6

minIndex = 1 → 1

1	2	6	5	4	3
0	1	2	3	4	5
1	2	6	5	4	3

minIndex = 5 → 5

1	2	3	4	5	6
0	1	2	3	4	5
1	2	3	4	5	6

minIndex = 2 → 5

1	2	6	5	4	3
0	1	2	3	4	5
1	2	3	5	4	6

minIndex = 3 → 4

1	2	3	5	4	6
0	1	2	3	4	5
1	2	3	4	5	6

*Understanding

```
public static void insertionSort(int[] a)
    for(int i=1, i<a.length; i++)
        int temp = a[i]
        int j = i-1
        while(j > -1 && temp < a[j])
            a[j+1] = a[j]
            a[j]   = temp
             j--
```

a =

0	1	2	3	4	5
4	2	6	5	1	3

Time Complexity = $O(n^2)$
if already or almost sorted:
Time Complexity = $O(n)$
Space Complexity = $O(1)$

i=1, temp=2

j=0, 2<4

4	2	6	5	1	3
0	1	2	3	4	5
2	4	6	5	1	3

i=2, temp=6

j=1, 6>4

2	4	6	5	1	3
0	1	2	3	4	5
2	4	6	5	1	3

i=3, temp=5

j=2, 5<6

j=1, 5>4

2	4	6	5	1	3
0	1	2	3	4	5
2	4	5	6	1	3

temp=

0	1	2	3	4	5
1	2	4	5	6	3
				j	i

i=4, temp=1

j=3, 1<6

2	4	5	6	1	3
0	1	2	3	4	5
2	4	5	1	6	3

j=2, 1<5

2	4	5	1	6	3
0	1	2	3	4	5
2	4	1	5	6	3

j=1, 1<4

2	4	1	5	6	3
0	1	2	3	4	5
2	1	4	5	6	3

j=0, 1<2

2	1	4	5	6	3
0	1	2	3	4	5
1	2	4	5	6	3

i=5, temp=3

j=4, 3<6

1	2	4	5	6	3
0	1	2	3	4	5
1	2	4	5	3	6

j=3, 3<5

1	2	4	5	3	6
0	1	2	3	4	5
1	2	4	3	5	6

j=2, 3<4

1	2	4	3	5	6
0	1	2	3	4	5
1	2	3	4	5	6

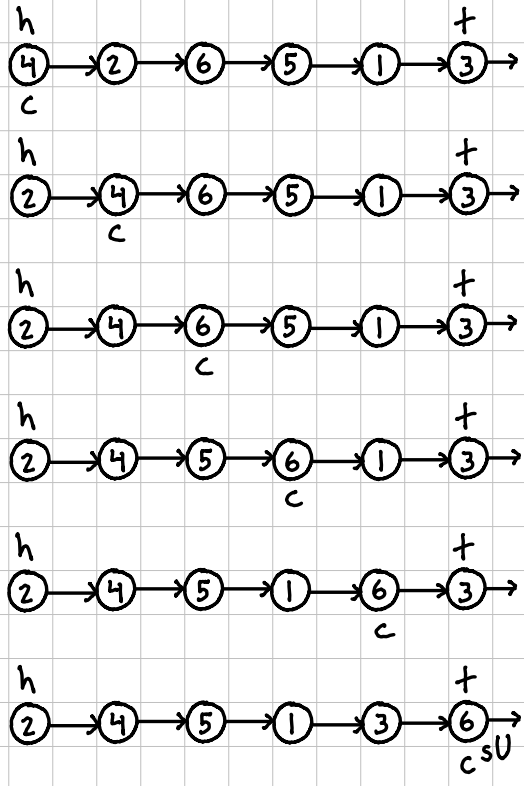
j=1, 3>2

1	2	3	4	5	6
0	1	2	3	4	5
1	2	3	4	5	6

public void bubbleSort() (Linked List)

- check if length < 2 → sorted → return
- Node sortedUntil = null → marker to end of sorted portion of list
- while (sortedUntil != this.head.next)
  - start from head → compare with next
  - current node > next node → swap values
  - compare and swap until reach node before sortedUntil
  - set sortedUntil to last node processed → marks end of sorted portion of list and start of unsorted portion for next iteration
- continue algo until sorted, no new LL

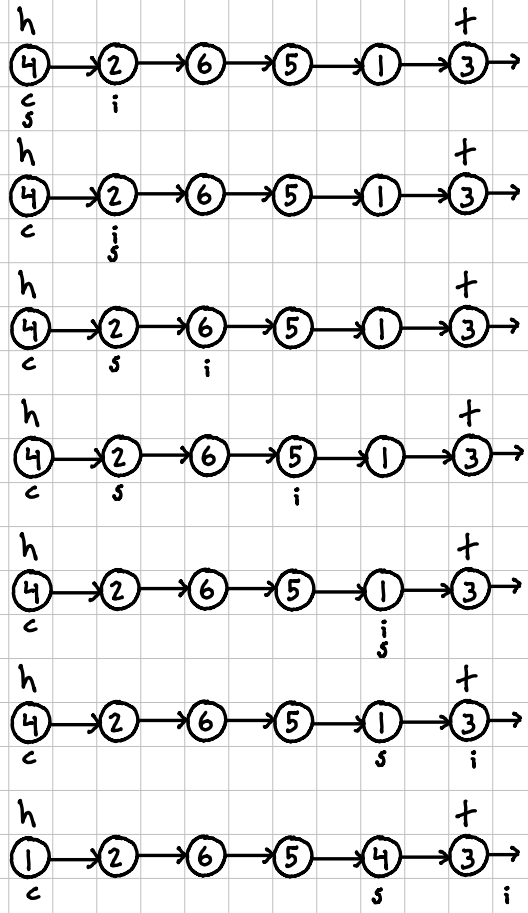
```
if(length<2)
   return
Node sU=null
while(sU!=this.head.next)
   Node c=this.head
   while (c.next!=sU)
      if(c.value>c.next.value)
         int temp=c.value
         c.value=c.next.value
         c.next.value=temp
      c=c.next
   sU=c
```



public void selectionSort()

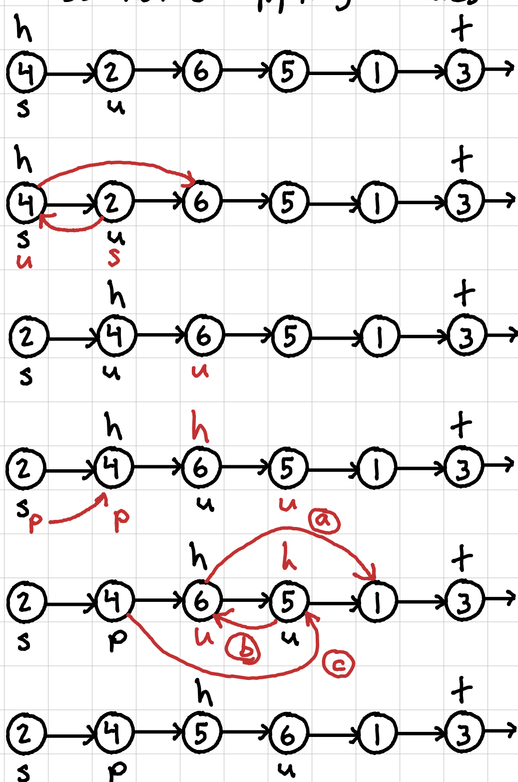
- check if length < 2 → sorted → return
- Node current = head
- while(current.next != null)
  - Node smallest = current , Node innerCurrent = current.next
  - while (innerCurrent != null)
    - if(innerCurrent.value < smallest.value) → smallest = innerCurrent
    - innerCurrent = innerCurrent.next
  - if (smallest != current) → swap values
  - current = current.next

```
if(length <2)
   return
Node c =head
while (c.next!=null)
  Node s=c
  Node i=c.next
  while (i !=null)
     if(i.value< s.value)
        s=i
     i = i.next
  if( s !=c)
     int temp = c.value
     c.value =s.value
     s. value = temp
  c =c.next
```



public void insertionSort()

- if length < 2 → sorted → return
- after sort update head and tail
- Node sortedListHead = head, Node unsortedListHead = head.next
- iterate through unsorted part and insert each node into correct position in sorted part
- if node in unsorted part < head of sorted part then node should be new head of sorted part
- else, iterate through sorted part using Node searchPointer until it finds correct position to insert node
- node should be inserted into sorted list by updating next reference of previous node to point new node and next reference of new node point to next node in sorted part so not swapping values

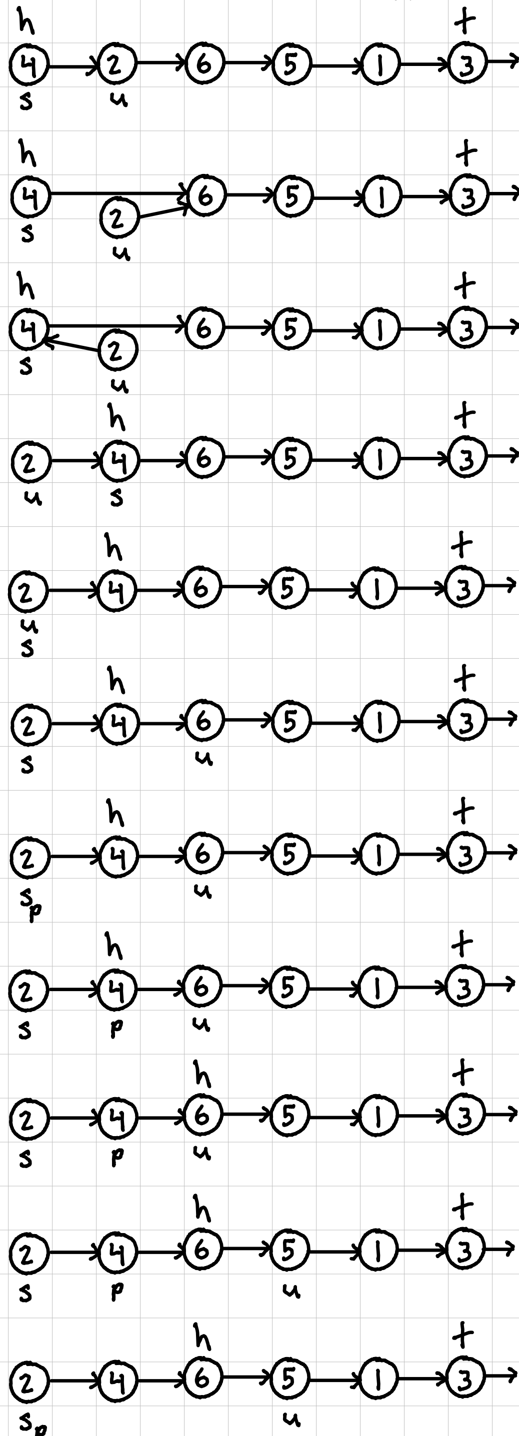


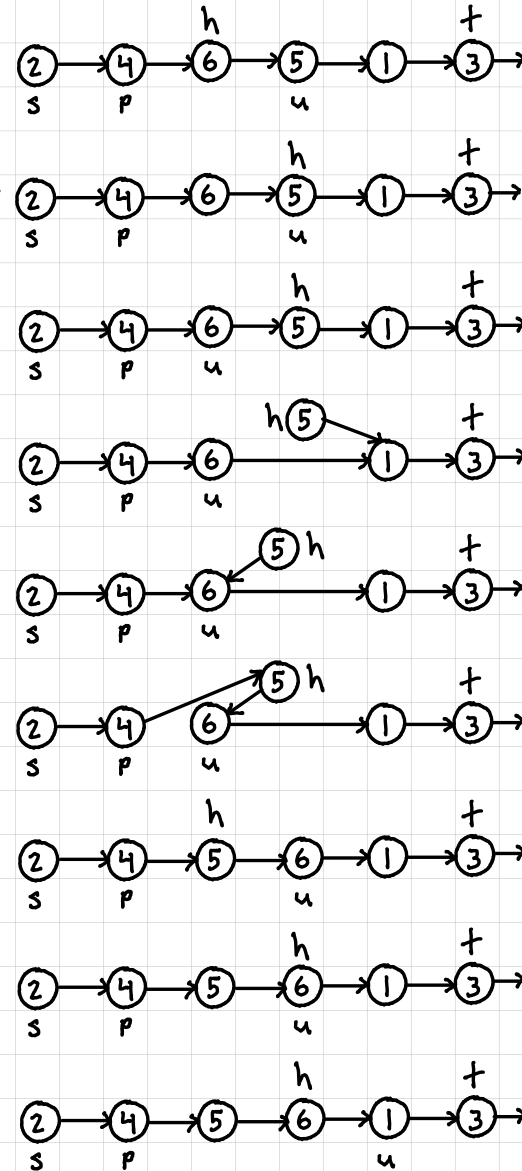

```
if (u.value < s.value)
  ① s.next = u.next        (if (h.next != u))
     u.next = s          *      h = h.next
     s = u                      h.next = u.next
     u = s.next                 u.next = s
                                s = u
else
   p = s
   while (p.next.value < u.value)
            && p.next != u   )
      p = p.next
   head = u
   u = p.next
   if (head != u)
      u.next = head.next
      head.next = u
      p.next = head   → h = u
u = u.next
*u = h.next
```

Doesn't work for ③

Doesn't work for ①

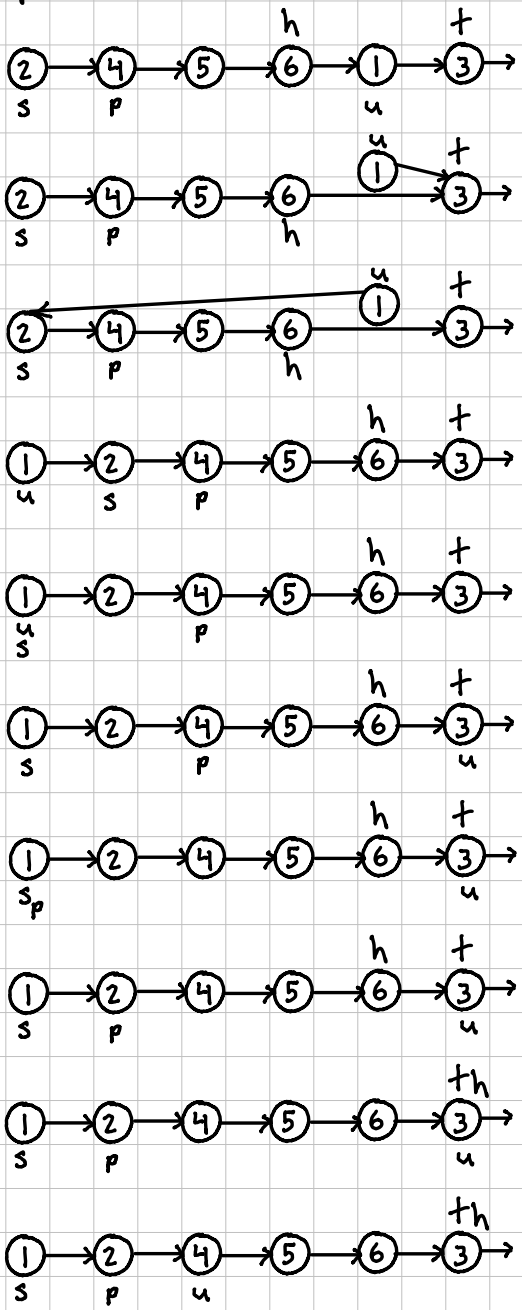
public void insertionSort()





Continued next page...

public void insertionSort()



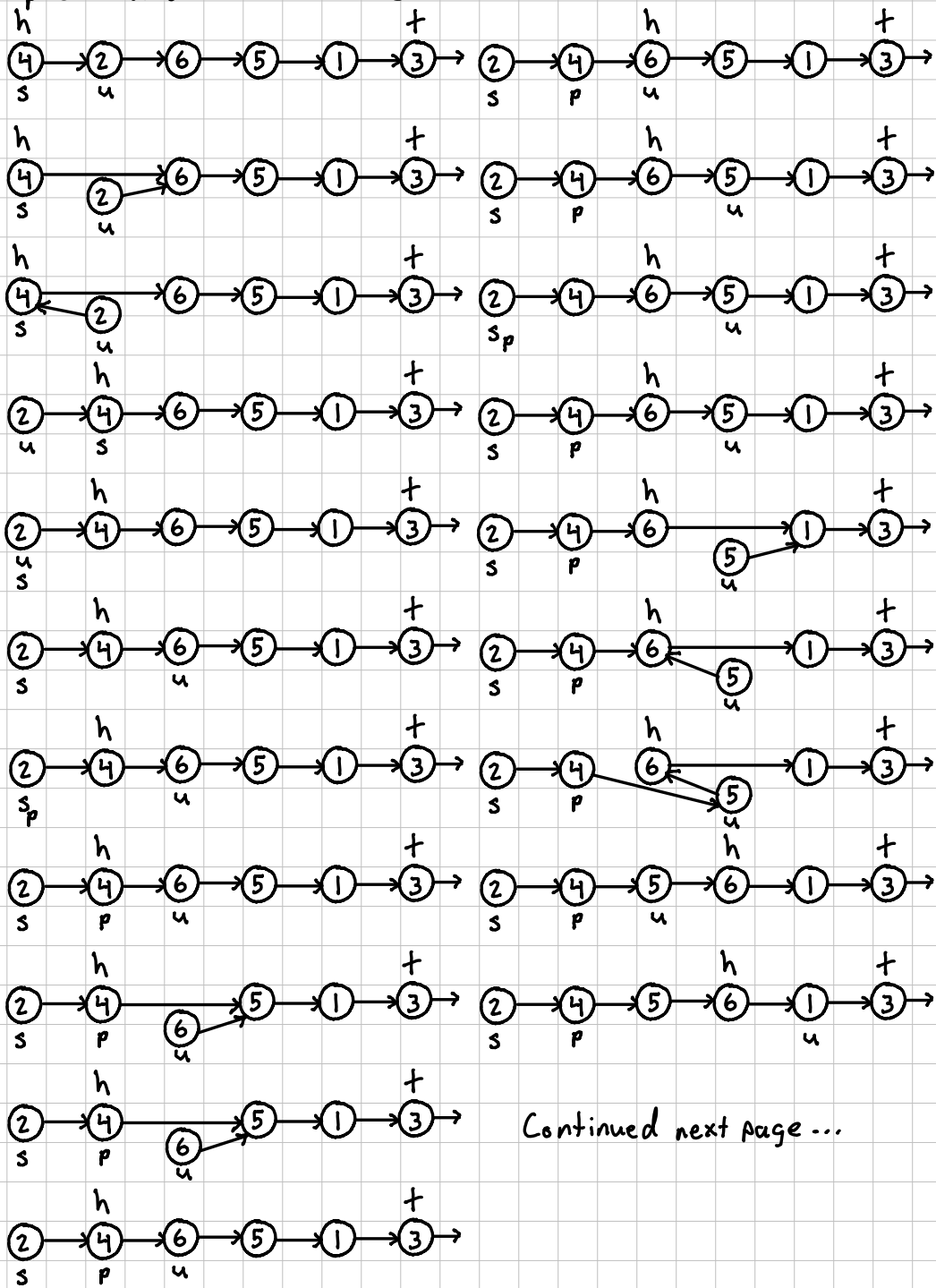
New code

```
if(u.value < s.value)
    h.next = u.next
    u.next = s
    s = u
else
    p = s
    while (p.next.value < u.value
            && p.next != u)
        p = p.next
    h.next = u.next
    u.next = p.next
    p.next = u
    if (h.next == u)
        h = u
u = h.next
```

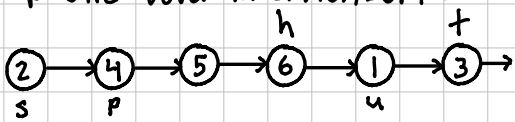
OK

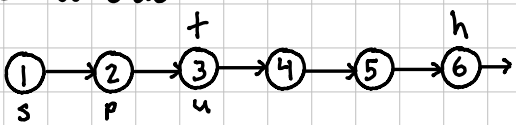
Doesn't work for last node

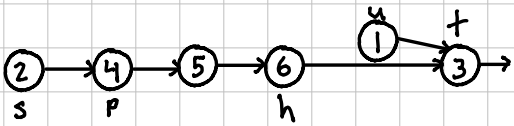
public void insertionSort() (New code)

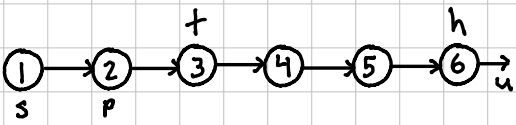


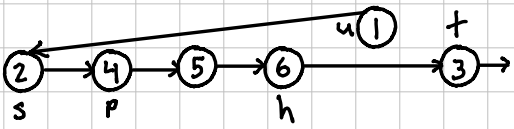
public void insertionSort() (New code)

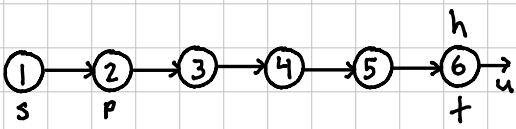


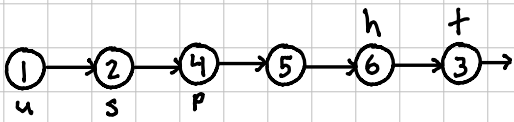


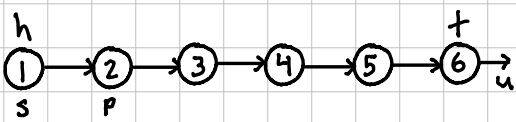


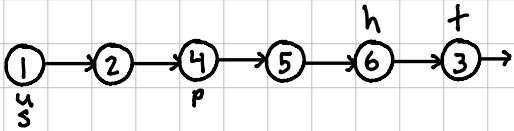


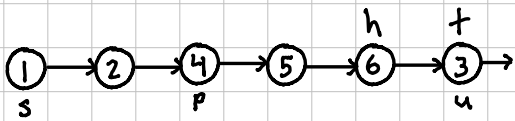


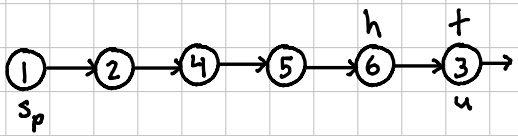


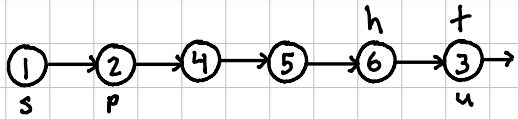


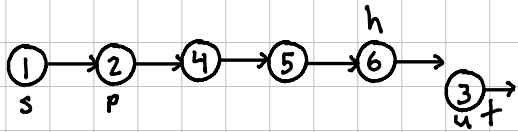


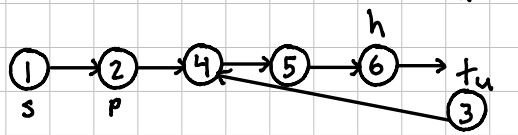


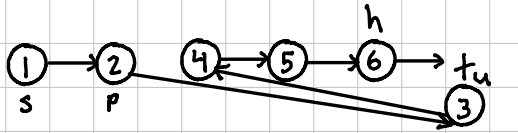








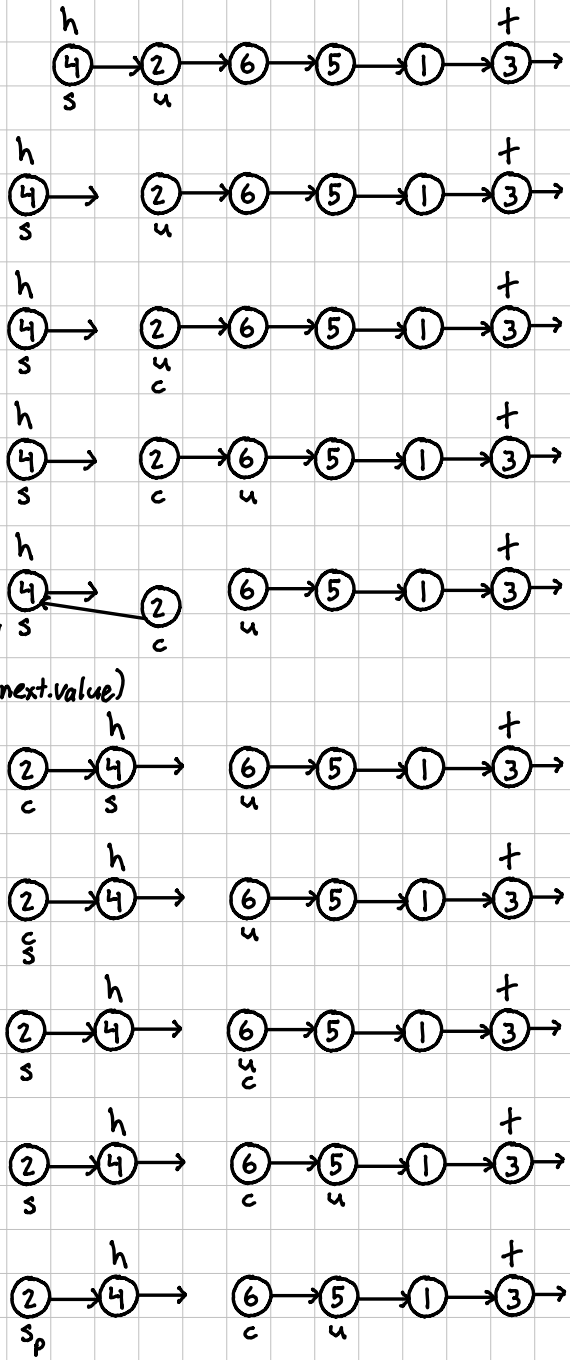





```
if(length<2)→return
Node s=h; Node u=h.next
while(u!=null)
    if(u.value<s.value)
        h.next=u.next
        u.next=s
        s=u
    else
        Node p=s
        while(p.next.value<u.value
                && p.next!=u)
            p=p.next
        h.next=u.next
        u.next=p.next
        p.next=u
        if(h.next==u)
            h=u
    u=h.next
t=h
h=s
```

*Understanding

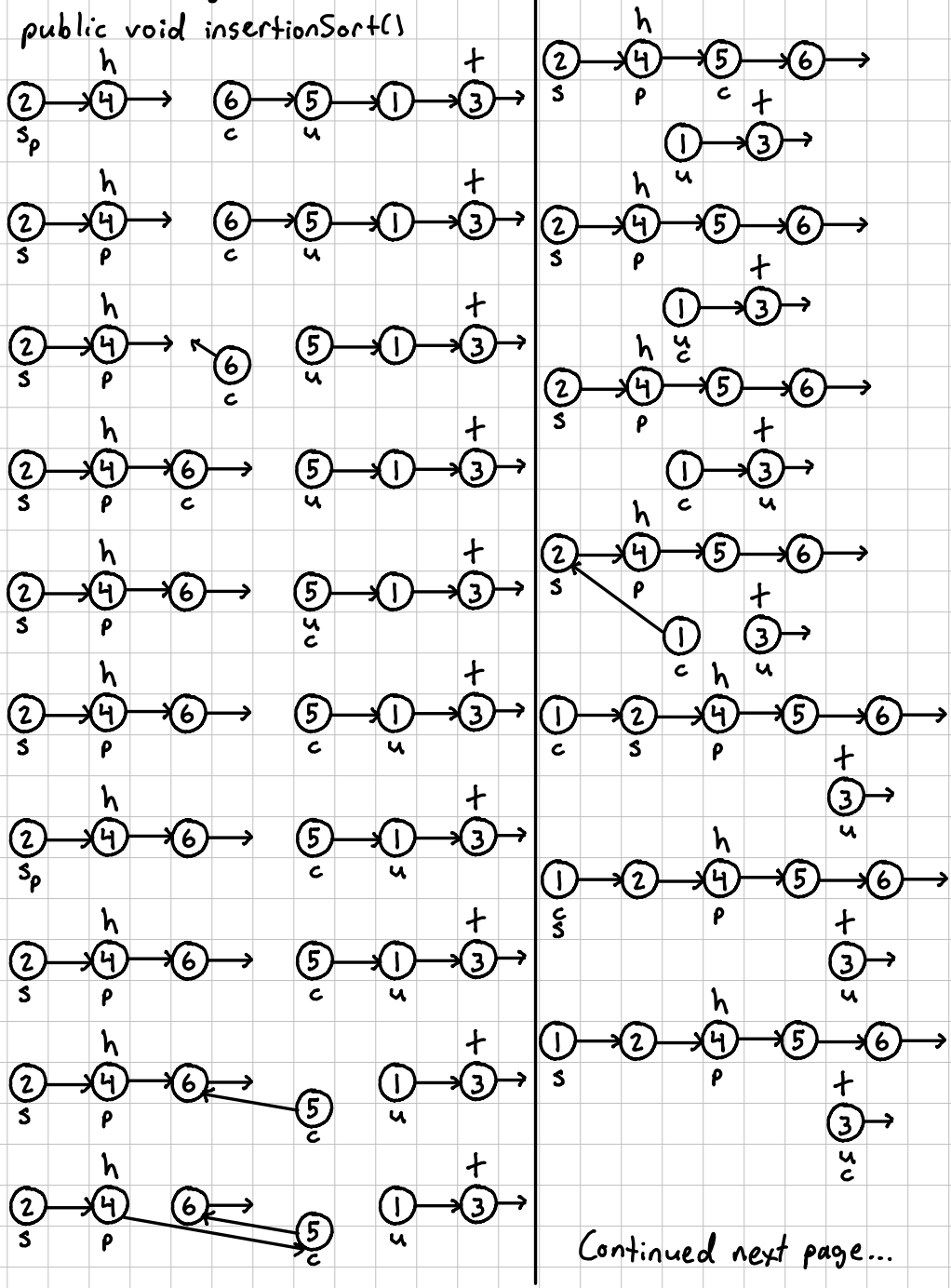
```
public void insertionSort()
  if(length<2)
     return
  Node s=head
  Node u=head.next
  s.next=null
  while(u!=null)
     Node c=u
     u=u.next
     if(c.value<s.value)
        c.next=s
        s=c
     else
        Node p=s
        while(p.next!=null
              && c.value>p.next.value)
           p=p.next
        c.next=p.next
        p.next=c
  head=s
  Node t=head
  while(t.next!=null)
     t=t.next
  tail=t
```



Continued next page...

*Understanding

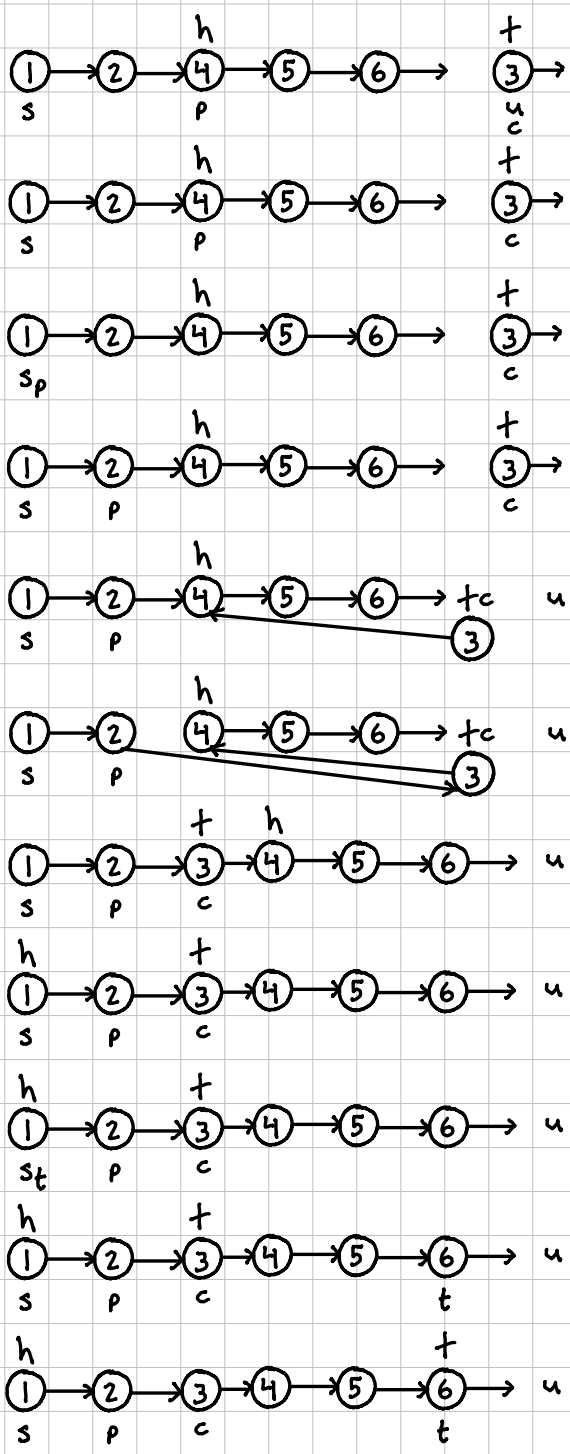
public void insertionSort()



Continued next page...

*Understanding

public void insertionSort()



*Understanding

$T = O(n \log n)$

$S = O(n)$ ↳ length of a

```
public static int[] mergeSort(int[] a)
    if(a.length ==1)
        return a
    int mI = a.length/2
    int[] l = mergeSort(Arrays.copyOfRange(a,0,mI))
    int[] r = mergeSort(Arrays.copyOfRange(a,mI,a.length))
    return merge(l,r) → T=O(n)

public static int[] merge(int[] a1, int[] a2) → T=O(n)
    int[] a = new int[a1.length+a2.length]
    int index =0
    int i =0
    int j =0
    while(i<a1.length && j<a2.length)
        if(a1[i] < a2[j])
            a[index] = a1[i]
            index++
            i++
        else
            a[index] = a2[j]
            index++
            j++
    while(i<a1.length)
        a[index] = a1[i]
        index++
        i++
    while(j<a2.length)
        a[index] = a2[j]
        index++
        j++
    return a
```

T=O(log n)

```
 i     j
[1,3] [2,4,5]  I=0
  [1]

   i   j
[1,3] [2,4,5] I=1
  [1,2]

    i    j
[1,3] [2,4,5] I=2
   [1,2,3]

     i  j
[1,3] [2,4,5] I=3
   [1,2,3,4]

      i   j
[1,3] [2,4,5] I=4
   [1,2,3,4,5]
```

*Understanding

public static int[] mergeSort(int[] a)

mS([3,1,4,2,5])
- mI = 5/2 = 2.5 = 2
- l = mS([3,1])
  - mI = 2/2 = 1
  - l = mS([3])
  - r = mS([1])
  - → = merge([3],[1]) = [1,3]
- r = mS([4,2,5])
  - mI = 3/2 = 1.5 = 1
  - l = mS([4])
  - r = mS([2,5])
    - mI = 2/2 = 1
    - l = mS([2])
    - r = mS([5])
    - → = merge([2],[5]) = [2,5]
  - → = merge([4],[2,5]) = [2,4,5]
- → = merge([1,3],[2,4,5]) = [1,2,3,4,5]

mS([3,5,2,6,1,4])
- mI = 6/2 = 3
- l = mS([3,5,2])
  - mI = 3/2 = 1.5 = 1
  - l = mS([3])
  - r = mS([5,2])
    - mI = 2/2 = 1
    - l = mS([5])
    - r = mS([2])
    - → = merge([5],[2]) = [2,5]
  - → = merge([3],[2,5]) = [2,3,5]
- r = mS([6,1,4])
  - mI = 3/2 = 1.5 = 1
  - l = mS([6])
  - r = mS([1,4])
    - mI = 2/2 = 1
    - l = mS([1])
    - r = mS([4])
    - → = merge([1],[4]) = [1,4]
  - → = merge([6],[1,4]) = [1,4,6]
- → = merge([2,3,5],[1,4,6]) = [1,2,3,4,5,6]

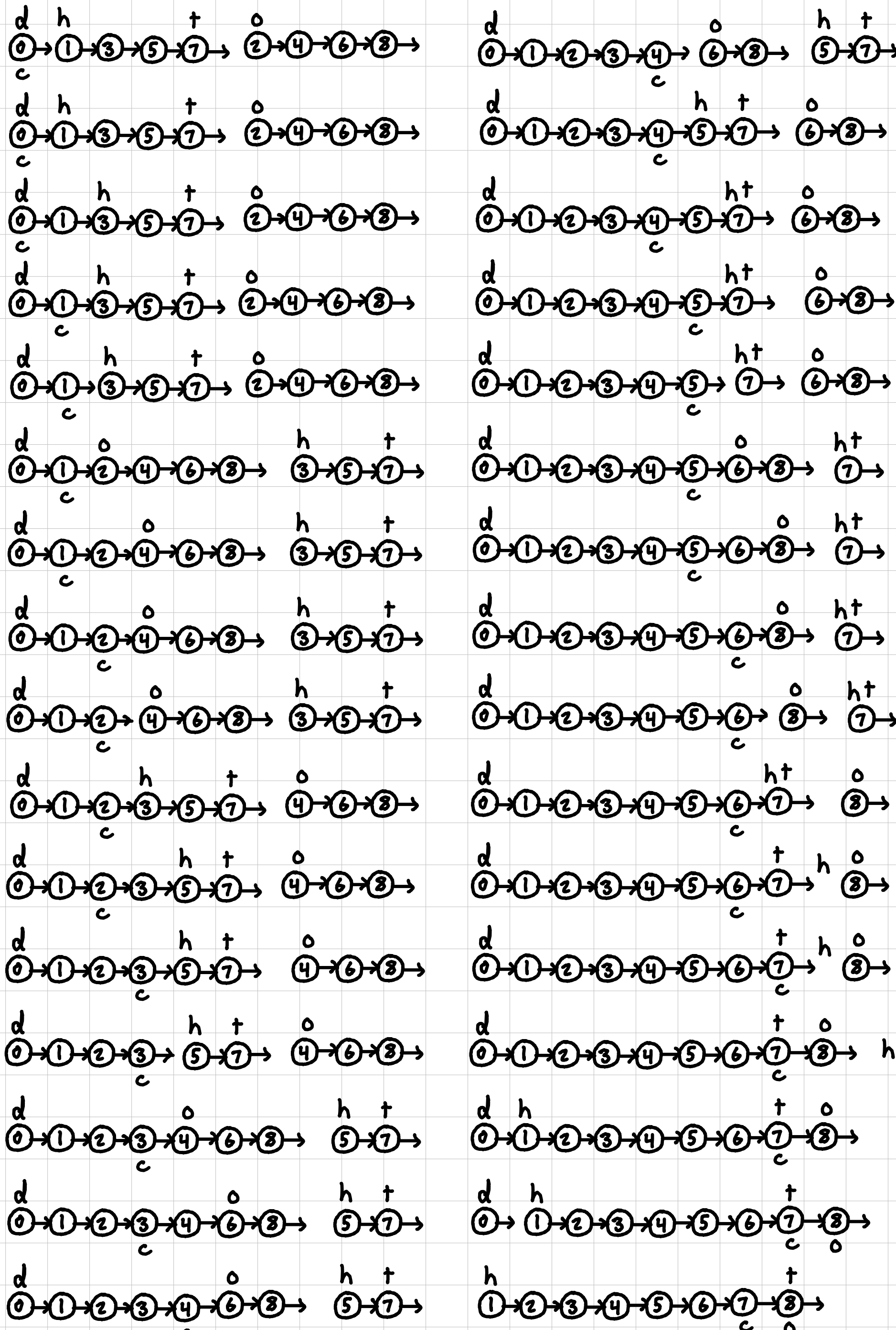
public void merge (LinkedList otherList)

list1 = [1,3,5,7], list2 = [2,4,6,8] → list1.merge(list2) = [1,2,3,4,5,6,7,8]

- assuming each list is in ascending order, if not, then list not in ascending order
- one list gets merged into the other
- gets head node of other list (otherList.getHead()) and sets it to otherHead
- dummy node = 0, head of merged list
- current = dummy which will be used to traverse merged linked list
- iterates through both input linked lists until null, append smaller node to merged list and updating head of that list to next node
- updates current to last node of merged list
- if either list still has nodes, appends them to the end of merged list
- updates head of current list to second node since first node is dummy
- updates length of current list to reflect merged list

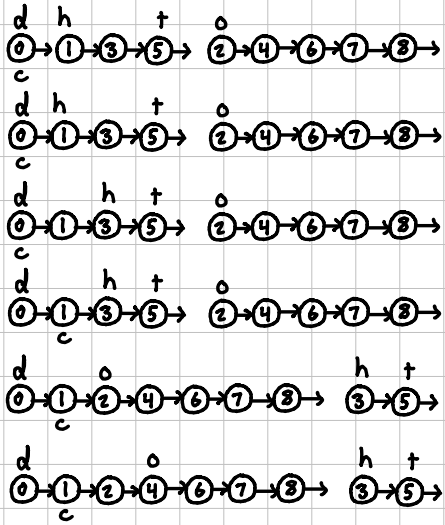
```
Node o = oL.getHead()
Node d = new Node(0)
Node c = d
length = 0
while (h != null && o != null)
    if (h.value < o.value)
        c.next = h
        h = h.next
        c = c.next
        length++
    else
        c.next = o
        o = o.next
        c = c.next
        length++
while (h != null)
    c.next = h
    h = h.next
    c = c.next
    length++
while (o != null)
    c.next = o
    o = o.next
    c = c.next
    length++
h = d.next
d.next = null
tail = c
```

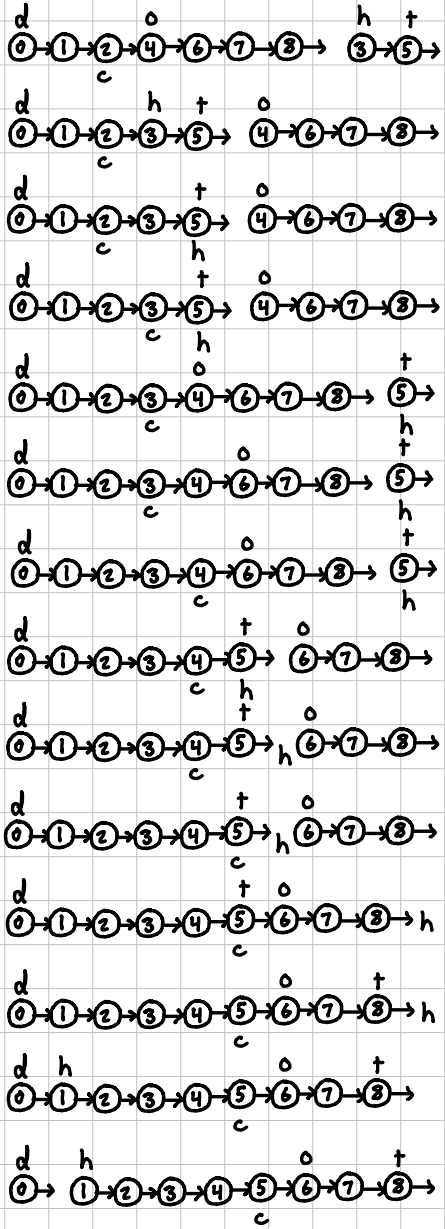
public void merge(LinkedList otherList)



*Understanding

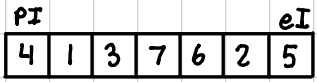
```
public void merge(LinkedList oL)
    Node o = oL.getHead()
    Node d = new Node(0)
    Node c = d
    while(h != null && o != null)
        if(h.value < o.value)
            c.next = h
            h = h.next
        else
            c.next = o
            o = o.next
        c = c.next
    if(h != null)
        c.next = h
    else
        c.next = o
        tail = oL.getTail()
    h = d.next
    d.next = null
    length += oL.getLength()
```

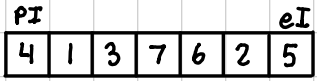


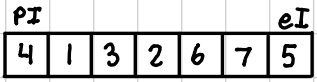


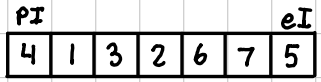
*Understanding

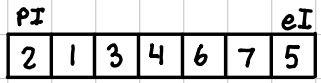
```
public static int pivot(int[] a, int pI, int endI)
    int swapI = pI
    for(int i = pI+1; i<=endIndex; i++)
        if(a[i] < a[pI])
            swapI++
            swap(a, swapI, i)
    swap(a, pI, swapI)
    return swapI
```

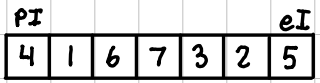


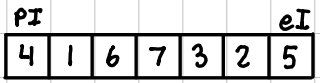


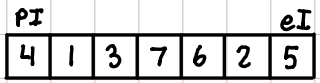






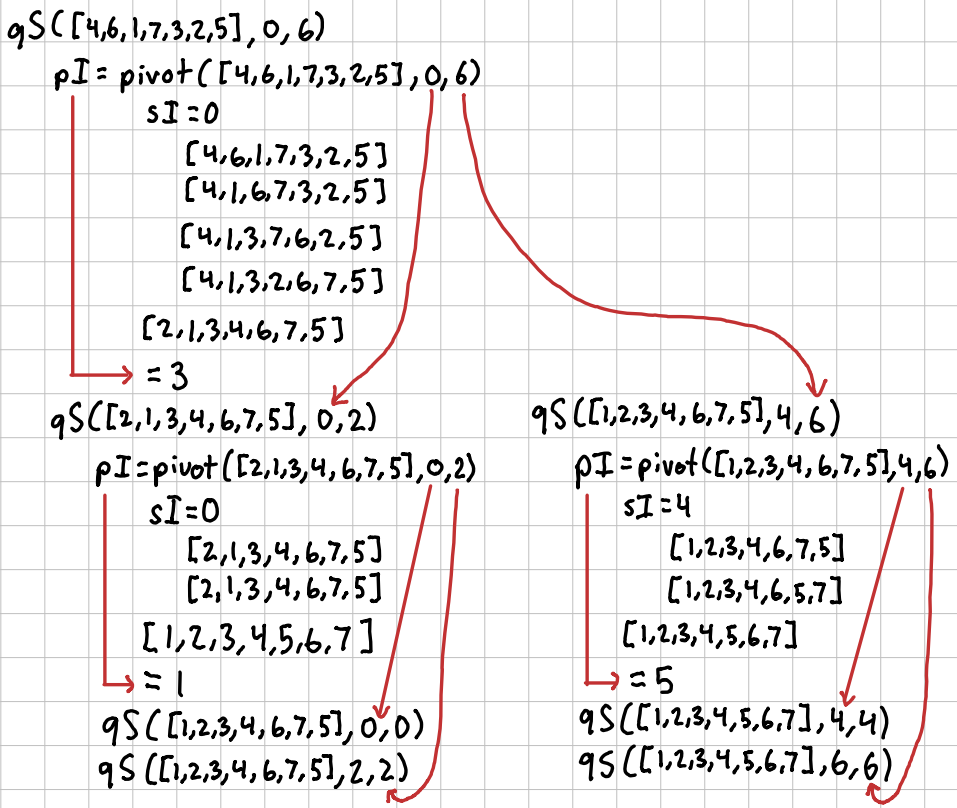


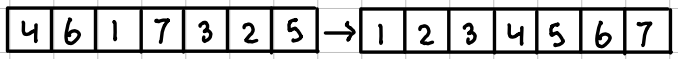




*Understanding

```
public static void quickSort(int[] a, int left, int right)
    if (left < right)
        int pI = pivot(a, l, r)
        quickSort (a, left, pivot-1)
        quickSort (a, pivot+1, right)
```

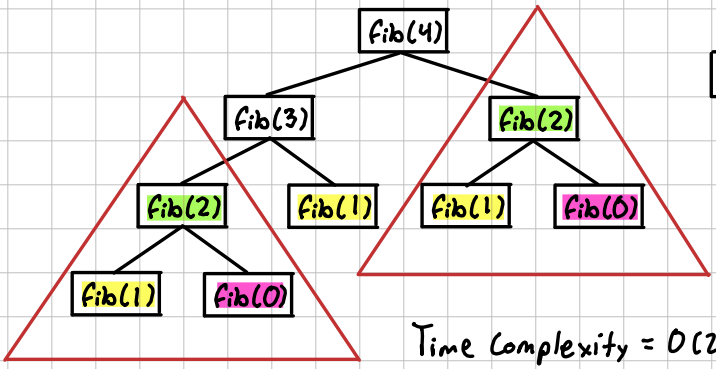




$T = O(n \log n)$ unless sorted then $T = O(n^2)$

*Understanding

```
public static int fib(int n)
    if(n ==0 || n==1)
        return n
    return fib(n-1) + fib(n-2)
```



Time Complexity = $O(2^n)$

0	1	2	3	4
0	1	1	2	3

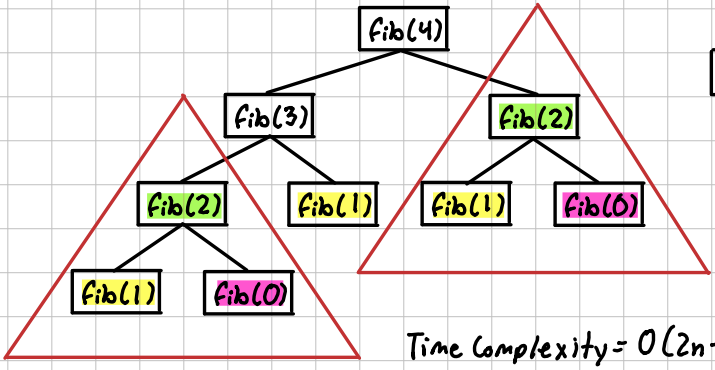
↓

just for visualization purpose

fib(4)
- fib(4-1) = fib(3)
  - fib(3-1) = fib(2)
    - fib(2-1) = fib(1)
      - = 1
    - + fib(2-2) = fib(0)
      - = 0
    - = 1
  - + fib(3-2) = fib(1)
    - = 1
  - = 2
- + fib(4-2) = fib(2)
  - fib(2-1) = fib(1)
    - = 1
  - + fib(2-2) = fib(0)
    - = 0
  - = 1
- = 3

*Understanding

```
static Integer[] memo = new Integer[100]
public static int fib(int n)
    if(memo[n] !=null)
        return memo[n]
    if (n==0 || n==1)
        return n
    memo[n] =fib(n-1)+fib(n-2)
    return memo[n]
```



0	1	2	3	4
0	1	1	2	3

↓

just for visualization purpose

Time Complexity = O(2n-1) = O(n)

fib(4)
- fib(4-1) = fib(3)
  - fib(3-1) = fib(2)
    - fib(2-1) = fib(1)
      - ↳ = 1
    - + fib(2-2) = fib(0)
      - ↳ = 0
    - ↳ = memo[2] = 1
  - + fib(3-2) = fib(1)
    - ↳ = 1
  - ↳ = memo[3] = 2
- + fib(4-2) = fib(2)
  - ↳ = memo[2] = 1
- ↳ = memo[4] = 3

*Understanding

```
public static int fib (int n)
    int[] fibList = new int[n+1]
    fibList[0] = 0
    fibList[1] = 1
    for(int i=2, i<=n, i++)
        fibList[i] = fibList[i-1] + fibList[i-2]
    return fibList[n]
```

fib (4)

fibList =

0	1	2	3	4

0	1	2	3	4
0				

0	1	2	3	4
0	1			

for(int i=2; i<=n; i++)

fibList[2] = fibList[2-1] + fibList[2-2]
= fibList[1] + fibList[0] = 0+1 = 1

0	1	2	3	4
0	1	1		

fibList[3] = fibList[3-1] + fibList[3-2]
= fibList[2] + fibList[1] = 1+1 = 2

0	1	2	3	4
0	1	1	2	

fibList[4] = fibList[4-1] + fibList[4-2]
= fibList[3] + fibList[2] = 2+1 = 3

0	1	2	3	4
0	1	1	2	3

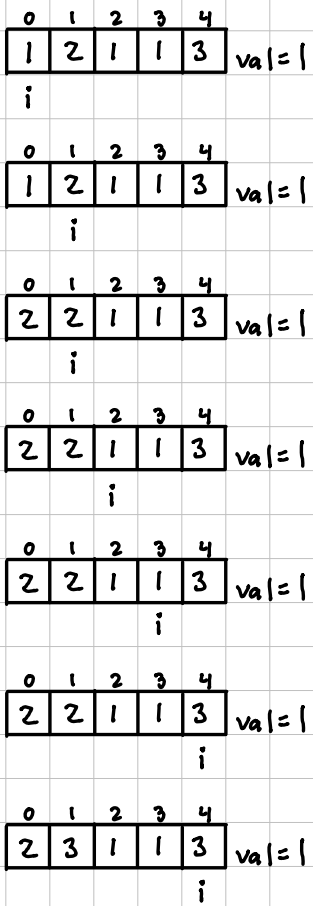
→ = fibList[4] = 3

public static int removeElement(int[] nums, int val)

- remove all val numbers in the array
- return new length of nums after removing val
- order of elements can be changed
- elements after new array length aren't considered
- iterate over nums, when number != val, then shift to front array

[3,2,2,3] v=3 → L=2 [2,2,int,int]

[-2,1,-3,4,-1,2,1,-5,4] v=1 → L=7 [-2,-3,4,-1,2,-5,4,-5,4]




```
int count = 0

for(int i=0; i<nums.length; i++)
    if (nums[i] != val)
        nums[i-count] = nums[i]
    else
        count++

return nums.length - count
```

Seems to work

public static int removeElement(int[] nums, int val)

0	1	2	3	4	5	6	7	8
-2	1	-3	4	-1	2	1	-5	-4
i								

0	1	2	3	4	5	6	7	8
-2	1	-3	4	-1	2	1	-5	-4
	i							

0	1	2	3	4	5	6	7	8
-2	1	-3	4	-1	2	1	-5	-4
		i						

0	1	2	3	4	5	6	7	8
-2	-3	-3	4	-1	2	1	-5	-4
		i						

0	1	2	3	4	5	6	7	8
-2	-3	-3	4	-1	2	1	-5	-4
			i					

0	1	2	3	4	5	6	7	8
-2	-3	4	4	-1	2	1	-5	-4
			i					

0	1	2	3	4	5	6	7	8
-2	-3	4	4	-1	2	1	-5	-4
				i				

0	1	2	3	4	5	6	7	8
-2	-3	4	-1	-1	2	1	-5	-4
				i				

0	1	2	3	4	5	6	7	8
-2	-3	4	-1	-1	2	1	-5	-4
					i			

0	1	2	3	4	5	6	7	8
-2	-3	4	-1	2	2	1	-5	-4
					i			

0	1	2	3	4	5	6	7	8
-2	-3	4	-1	2	2	1	-5	-4
						i		

0	1	2	3	4	5	6	7	8
-2	-3	4	-1	2	2	1	-5	-4
							i	

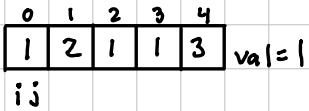
0	1	2	3	4	5	6	7	8
-2	-3	4	-1	2	-5	1	-5	-4
							i	

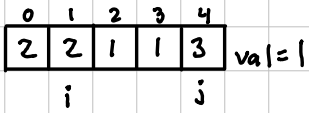
0	1	2	3	4	5	6	7	8
-2	-3	4	-1	2	-5	1	-5	-4
								i

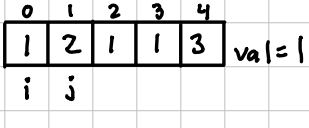
0	1	2	3	4	5	6	7	8
-2	-3	4	-1	2	-5	-4	-5	-4
								i

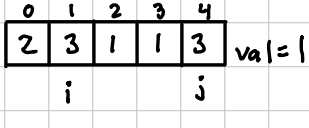
*Understanding

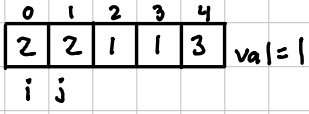
```
public static int removeElement(int[] nums, int val)
    int i=0
    for(int j=0; j<nums.length; j++)
        if(nums[j] != val)
            nums[i] = nums[j]
            i++
    return i
```

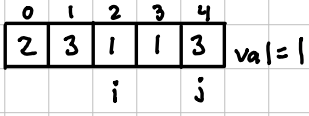


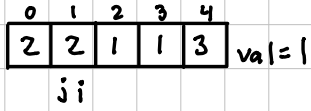


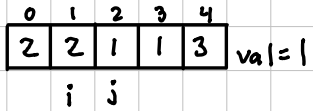


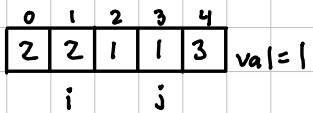




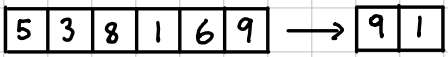




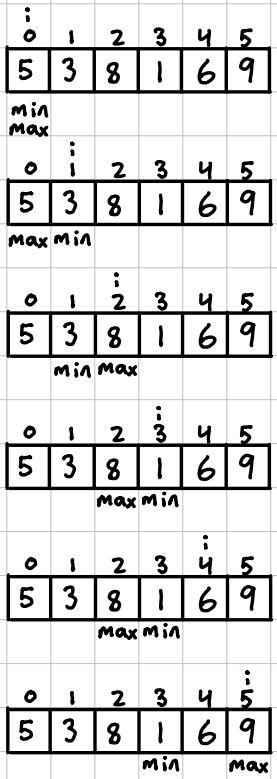




public static int[] findMaxMin(int[] myList)



- iterate through array to find max and min
- should work with zero and negative numbers
- should work even with duplicates




```
int min = myList[0]
int max = myList[0]

for (int i=0; i<myList.length; i++)

    if (myList[i] < min)
        min = myList[i]

    if(myList[i] > max)
        max = myList[i]

return new int[] {max, min}
```

Seems to work

Solution:

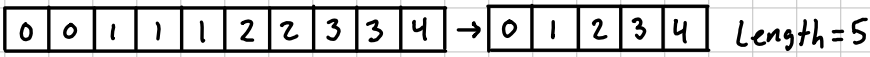
```
int min = myList[0]
int max = myList[0]
for (int num : myList)
    if(num > max)
        max = num
    else if(num < min)
        min = num
return new int[] {max, min}
```

```
public static String findLongestString(String[] stringList)
```

["apple", "banana", "kiwi", "pear"] → "banana"

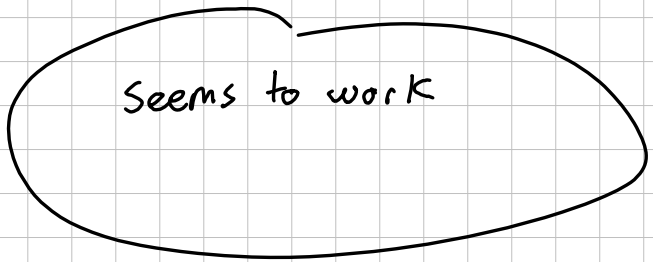
```
String longest = ""

for(String s : stringList)

    if(s.length() > longest.length())
        longest = s

return longest
```

public static int removeDuplicates(int[] nums)



- remove duplicates from sorted array in-place
- move unique elements to the start of the array
- return length of unique portion of array
- return 0 if empty

```
if(nums.length ==0)
    return 0

int current =0

for(int i=0; i<nums.length; i++)
   if(nums[i] != nums[current])
      current ++
      nums[current] = nums[i]

return current + 1
```



public static int removeDuplicates(int[] nums)

0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	2	2	3	3	4
i c									

0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	2	2	3	3	4
c	i								

0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	2	2	3	3	4
c		i							

0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	2	2	3	3	4
	c	i							

0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	2	2	3	3	4
	c	i							

0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	2	2	3	3	4
	c		i						

0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	2	2	3	3	4
	c			i					

0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	2	2	3	3	4
	c				i				

0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	2	2	3	3	4
		c			i				

0	1	2	3	4	5	6	7	8	9
0	1	2	1	1	2	2	3	3	4
		c			i				

0	1	2	3	4	5	6	7	8	9
0	1	2	1	1	2	2	3	3	4
		c				i			

0	1	2	3	4	5	6	7	8	9
0	1	2	1	1	2	2	3	3	4
		c					i		

0	1	2	3	4	5	6	7	8	9
0	1	2	1	1	2	2	3	3	4
			c				i		

0	1	2	3	4	5	6	7	8	9
0	1	2	3	1	2	2	3	3	4
			c				i		

0	1	2	3	4	5	6	7	8	9
0	1	2	3	1	2	2	3	3	4
			c					i	

0	1	2	3	4	5	6	7	8	9
0	1	2	3	1	2	2	3	3	4
			c						i

0	1	2	3	4	5	6	7	8	9
0	1	2	3	1	2	2	3	3	4
				c					i

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	2	2	3	3	4
				c					i

* Understanding

```
public static int removeDuplicates (int[] nums)
    if(nums.length ==0)
        return 0
    int write=1
    for(int read =1;read<nums.length; read++)
        if(nums[read] != nums[read-1])
            nums[write]=nums[read]
            write++
    return write
```

0	1	2	3	4	5	6	7
0	0	1	1	1	2	2	3
	w r						

0	1	2	3	4	5	6	7
0	0	1	1	1	2	2	3
	w	r					

0	1	2	3	4	5	6	7
0	1	1	1	1	2	2	3
	w	r					

0	1	2	3	4	5	6	7
0	1	1	1	1	2	2	3
		r w					

0	1	2	3	4	5	6	7
0	1	1	1	1	2	2	3
		w	r				

0	1	2	3	4	5	6	7
0	1	1	1	1	2	2	3
		w		r			

0	1	2	3	4	5	6	7
0	1	1	1	1	2	2	3
		w			r		

0	1	2	3	4	5	6	7
0	1	2	1	1	2	2	3
		w			r		

0	1	2	3	4	5	6	7
0	1	2	1	1	2	2	3
			w		r		

0	1	2	3	4	5	6	7
0	1	2	1	1	2	2	3
			w			r	

0	1	2	3	4	5	6	7
0	1	2	1	1	2	2	3
			w				r

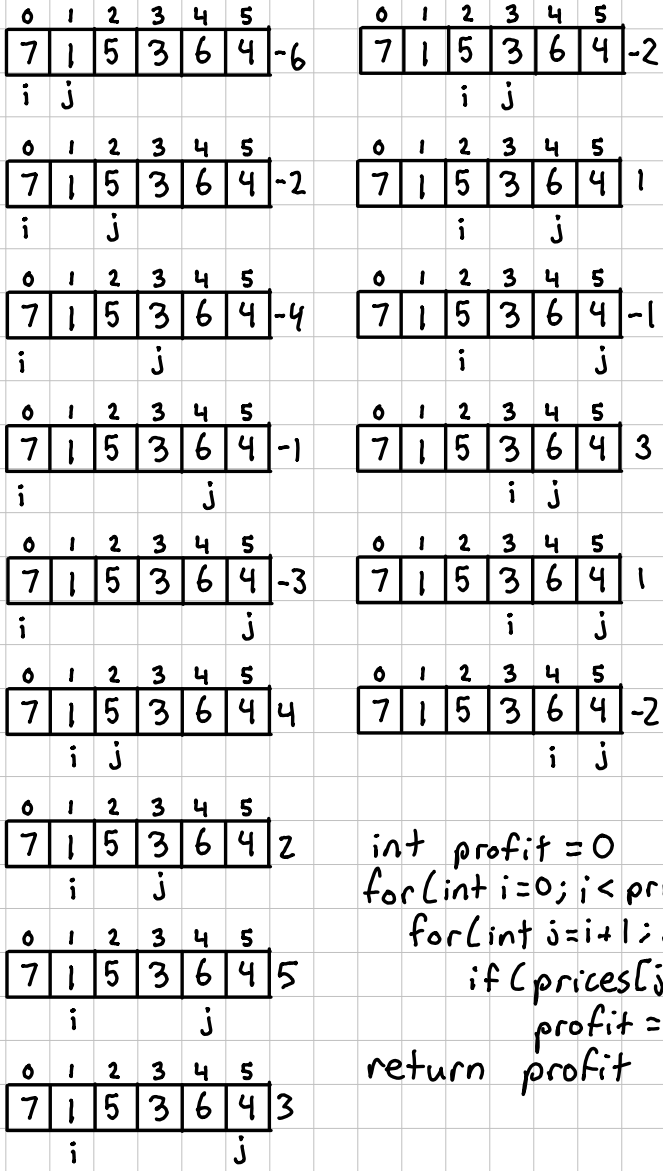
0	1	2	3	4	5	6	7
0	1	2	3	1	2	2	3
			w				r

public static int maxProfit (int[] prices)

- takes array representing price of stock on different days
- determine max profits made by buy and sell (must buy first then sell)

7	1	5	3	6	4

→ buy 1, sell 6 → 6-1 = 5 (max profit)

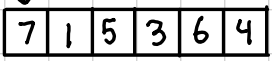


Seems to work

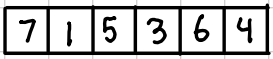
```
int profit = 0
for(int i=0; i< prices.length-1; i++)
   for(int j=i+1; j<price.length; j++)
      if(prices[j]-prices[i]>profit)
         profit = prices[j]-prices[i]
return profit
```

*Understand

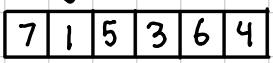
```
public static int maxProfit (int[] prices]
    int minPrice = Integer.MAX_VALUE → (largest possible integer)
    int maxProfit = 0
    for(int price: prices)
        minPrice = Math.min(minPrice, price) → (check which smaller)
        int profit = price - minPrice
        maxProfit = Math.max(maxProfit, profit) → (check which larger)
  return maxProfit
```



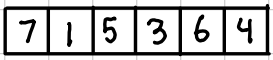
min(int max, 7) = 7
profit = 7 - 7 = 0
max(0,0) = 0



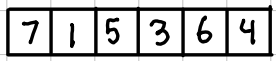
min(1,6) = 1
profit = 6 - 1 = 5
max(4,5) = 5



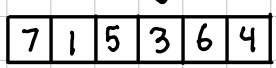
min(7,1) = 1
profit = 1 - 1 = 0
max(0,0) = 0



min(1,4) = 1
profit = 4 - 1 = 3
max(5,3) = 5



min(1,5) = 1
profit = 5 - 1 = 4
max(0,4) = 4



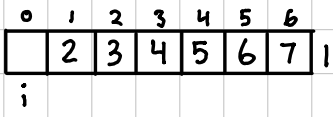
min(1,3) = 1
profit = 3 - 1 = 2
max(4,2) = 4

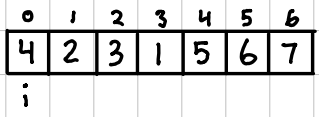
* Code rewritten

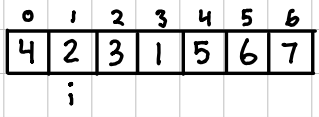
```
if(prices.length==0)
    return 0
int min = prices[0]
int max = 0
for(int price:prices)
    if( price < min)
        min = price
    int profit = price - min
    if(profit > max)
        max = profit
return max
```

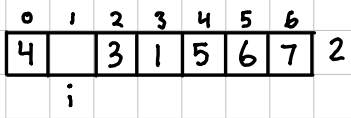
public static void rotate(int[] nums, int k) T = O(n)
S = O(1)

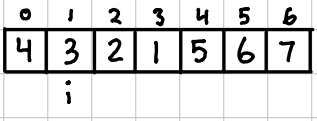
- rotate array to the right by k steps (k>=0) in-place
- each step needs to move last element to the front and shift the rest to the right
- number of rotations = k mod n (n = length nums) because after n rotations, array return to original state
- rotate without creating copy of array (except for temp variables)

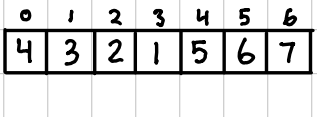












int pos = k % nums.length

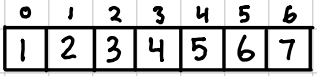
```
for (int i = 0; i <= pos/2; i++)
    int temp = nums[i]
    nums[i] = nums[pos-i]
    nums[pos-i] = temp
```

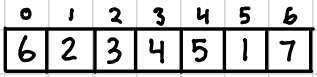
Incomplete

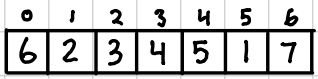
r = k mod n
r = 0 mod 7 = 0
r = 1 mod 7 = 1
r = 2 mod 7 = 2
r = 3 mod 7 = 3
r = 4 mod 7 = 4
r = 5 mod 7 = 5
r = 6 mod 7 = 6
r = 7 mod 7 = 0
⋮
r = 33 mod 7 = 5
⋮

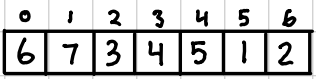
i = (i+k) mod n

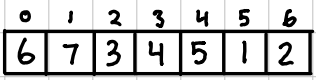
public static void rotate (int[] nums, int k)

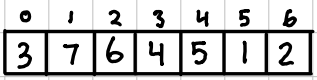


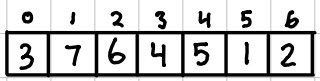


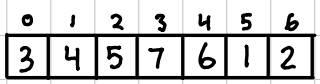


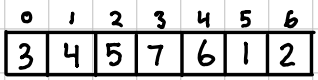


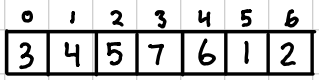


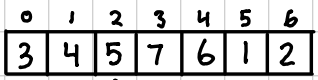


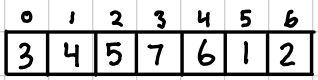


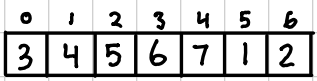




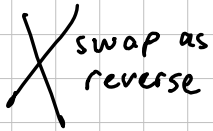




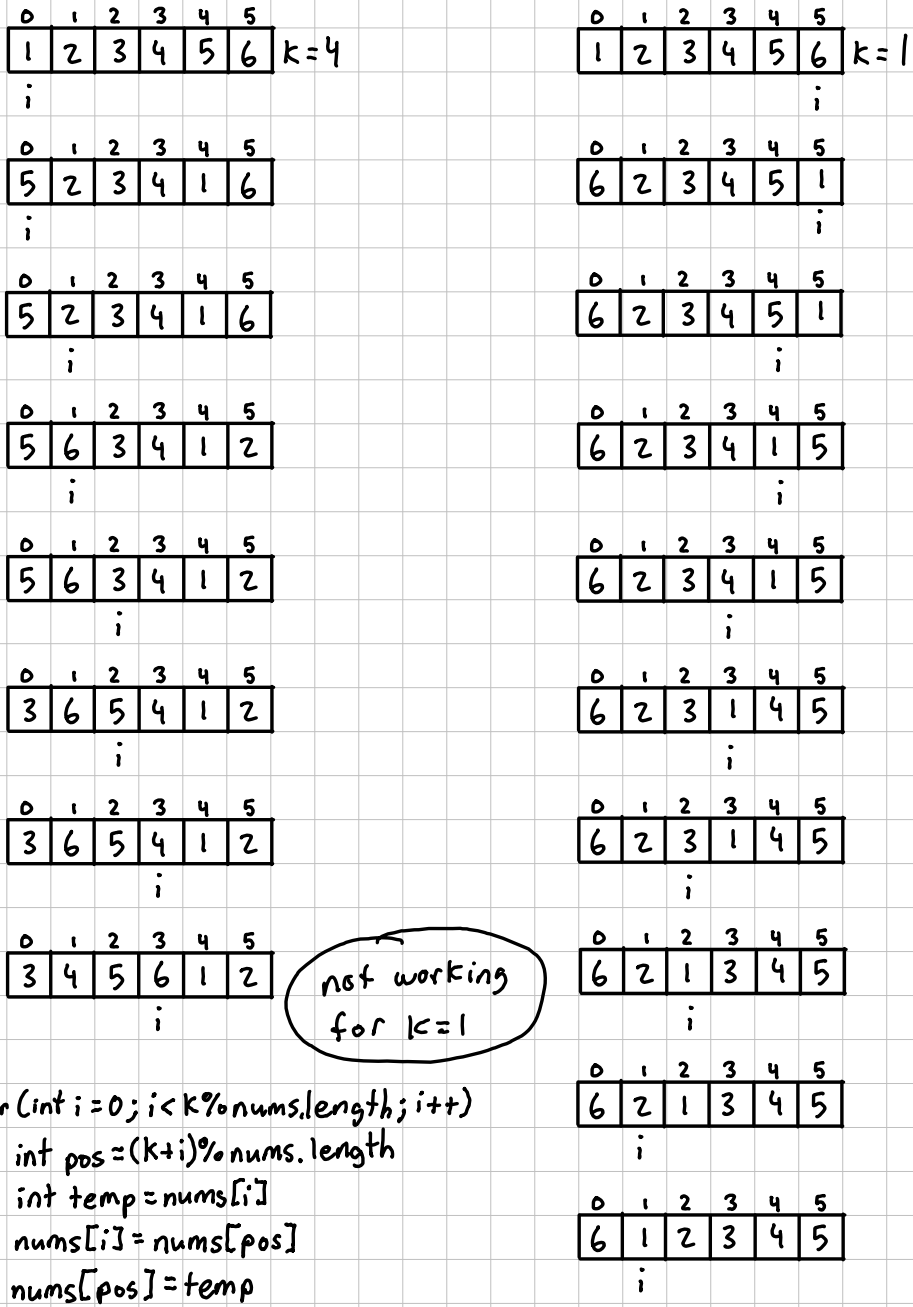





```
int pos = k % nums.length
for(int i = 0; i <= pos/2; i++)
    int temp = nums[i]
    nums[i] = nums[pos-i]
    nums[pos-i] = temp
for(int j = 0; j < pos-1; j++)
    if(nums[j] > nums[j+1])
        int temp = nums[j]
        nums[j] = nums[j+1]
        nums[j+1] = temp
```

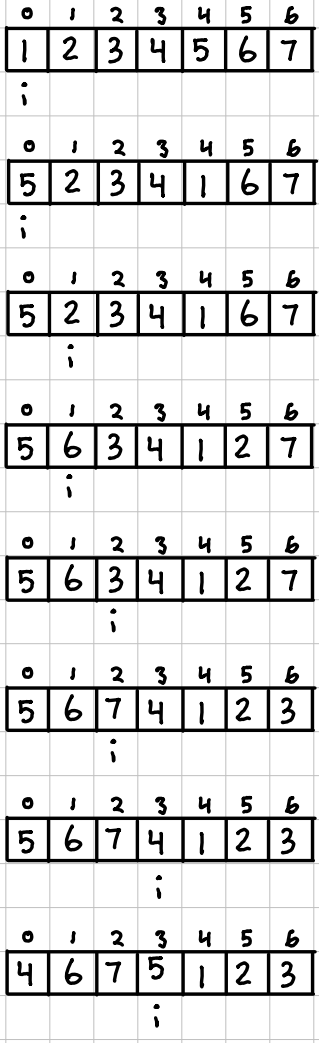


public static void rotate (int[] nums, int k)




```
for (int i=0; i<k%nums.length; i++)
    int pos=(k+i)%nums.length
    int temp=nums[i]
    nums[i]=nums[pos]
    nums[pos]=temp
```

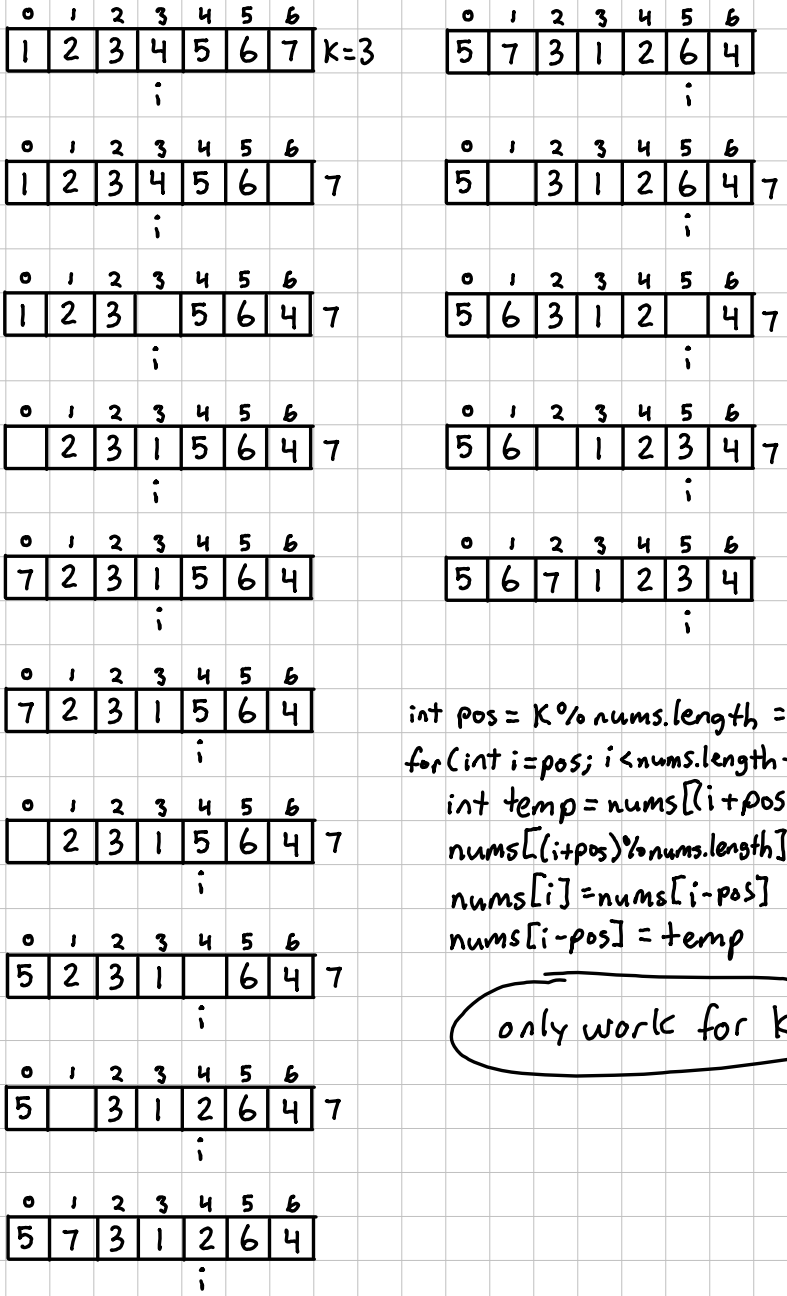
public static void rotate (int[] nums, int k)




```
if( K%nums.length<nums.length/2)
    for(int i=nums.length-1;
        i>=K%nums.length; i--)
        int pos=(k+i)%nums.length
        int temp=nums[i]
        nums[i]=nums[pos]
        nums[pos]=temp
else
    for(int i=0; i<K%nums.length; i++)
        int pos=(k+i)%nums.length
        int temp=nums[i]
        nums[i]=nums[pos]
        nums[pos]=temp

if (K%nums.length==nums.length/2)
```

not working for odd length array

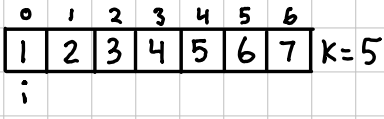
public static void rotate (int[] nums, int k)

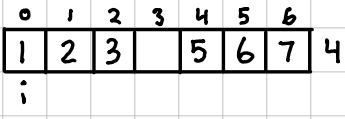


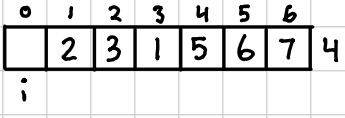

```
int pos = K % nums.length = 3
for(int i=pos; i<nums.length-1; i++)
    int temp = nums[(i+pos)%nums.length]
    nums[(i+pos)%nums.length] = nums[i]
    nums[i] = nums[i-pos]
    nums[i-pos] = temp
```

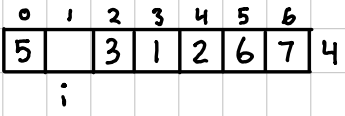
only work for k=3

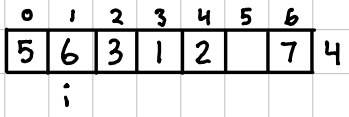
public static void rotate (int[] nums, int k)

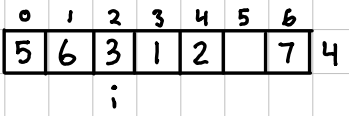


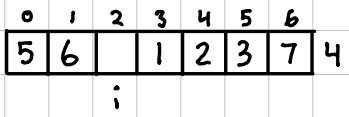


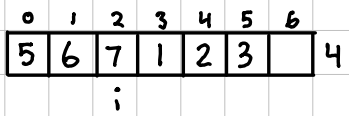


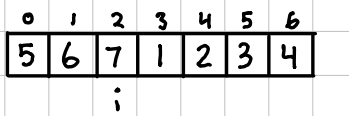


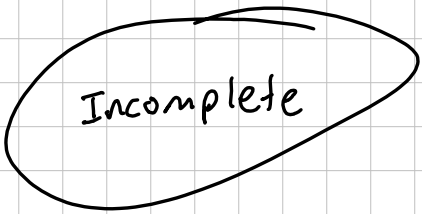




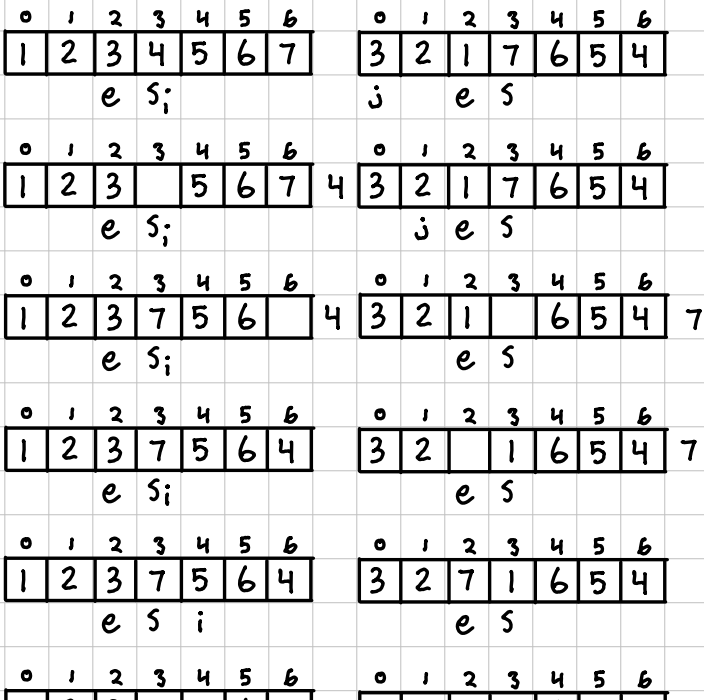




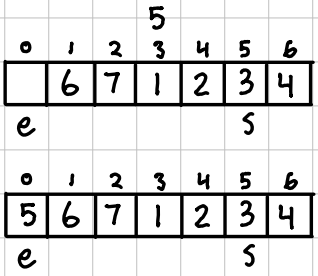




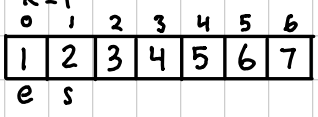
public static void rotate(int[] nums, int k)



K=3

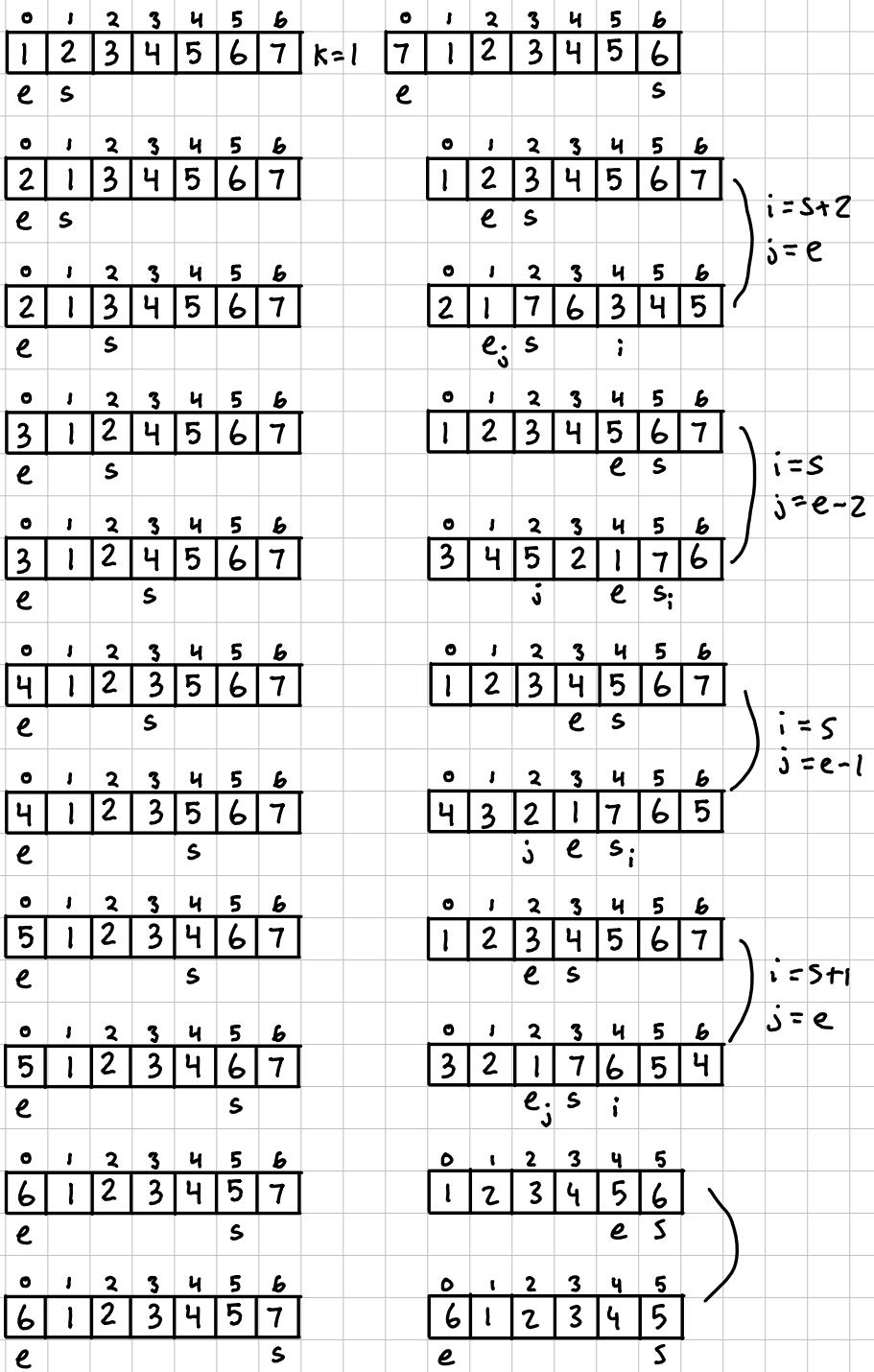


K=1

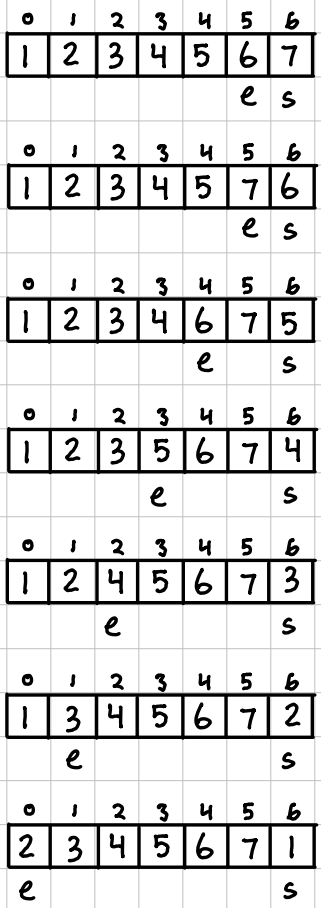


Doesn't work with K=1

public static void rotate (int[] nums, int k)



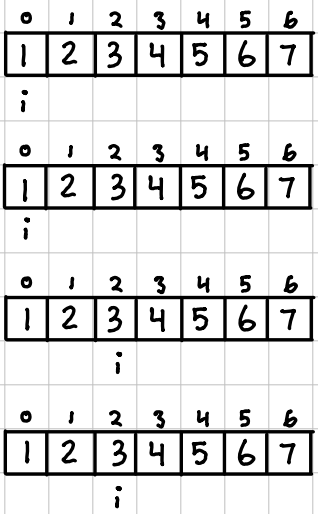
public static void rotate (int[] nums, int k)




```
if (k % nums.length == 0)
    return
int s = k % nums.length
int e = k % nums.length - 1

if (s == nums.length - 1 || e = 0)
    while(s < nums.length)
        int temp = nums[s]
        nums[s] = nums[e]
        nums[e] = temp
        s++
    while( e >= 0)
        int temp = nums[s]
        nums[s] = nums[e]
        nums[e] = temp
        e--
else
```

Incomplete

public static void rotate (int[] nums, int k)



k=2 cycle

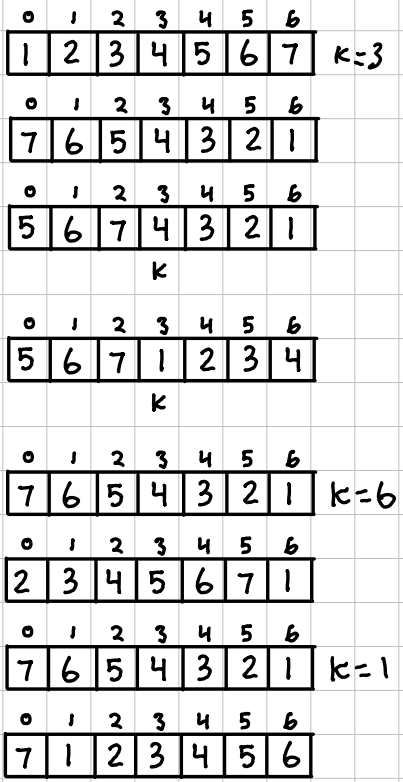
from	to
Left X	
0	temp
2	0
4	2
6	4
1	6
3	1
5	3
temp	5

k=3

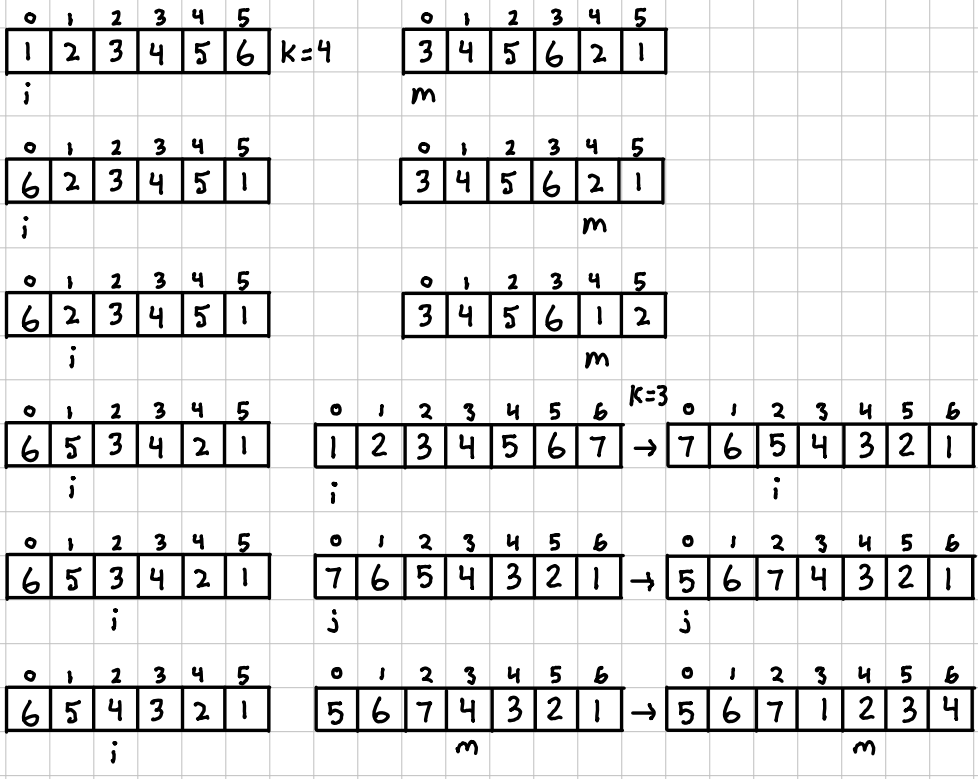
from	to
0	3
3	6
6	2
2	5
5	1
1	4
4	0

Doesn't work for even length array and even k?

reverse



public static void rotate(int[] nums, int k)



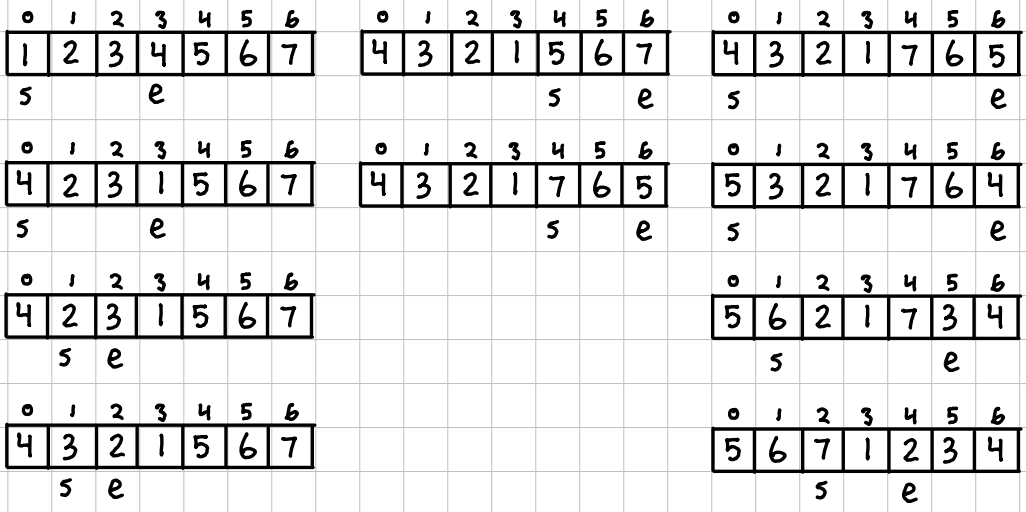

```
int pos = k%nums.length → if(pos==0) return
int lastIndex=nums.length-1
for(int i=0, i<nums.length/2; i++)
    int temp = nums[i]
    nums[i] = nums[lastIndex-i]
    nums[lastIndex-i]=temp
for(int j=0; j<pos/2; j++)
    int temp = nums[j]
    nums[j] = nums[pos-1-j]
    nums[pos-1-j] = temp
for(int m=0; m<(nums.length-pos)/2; m++)
    int temp = nums[m+pos]
    nums[m+pos] = nums[lastIndex-m]
    nums[lastIndex-m]=temp
```

seems to work

*Understanding

```
public static void rotate(int[] nums, int k)
    k = k % nums.length
    for(int s = 0, e = nums.length - k - 1; s < e ; s++, e--)
        int temp = nums[s]
        nums[s] = nums[e]
        nums[e] = temp
    for(int s = nums.length - k, e = nums.length - 1; s < e; s++, e--)
        int temp = nums[s]
        nums[s] = nums[e]
        nums[e] = temp
    for(int s = 0, e = nums.length - 1; s < e; s++, e--)
        int temp = nums[s]
        nums[s] = nums[e]
        nums[e] = temp
```

K=3

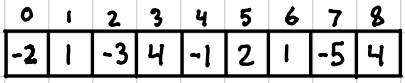


public static int maxSubarray(int[] nums)

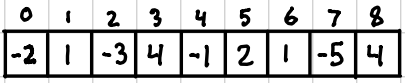
- find contiguous subarray (with at least one value) which has the largest sum and return sum
- subarray can be entire array, or smaller part, can have negative values

-2	1	-3	4	-1	2	1	-5	4

→ | 4 | -1 | 2 | 1 | = 6



sum = -2, max = -2 [0,0]
[-2]



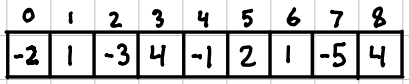
sum = -1, -1 < 1, max = 1 [1,1]
[1]

sum = -2, max = 1
[1, -3]

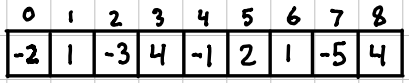
sum = 2, 2 < 4, max = 4 [3,3]
[4]

sum = 3, max = 4
[4, -1]

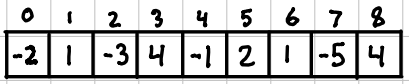
sum = 5, max = 5 [3,5]
[4, -1, 2]



sum = 6, max = 6 [3,6]
[4, -1, 2, 1]

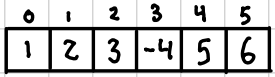


sum = 1, max = 6
[4, -1, 2, 1, -5]

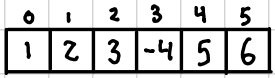


sum = 5, max = 6
[4, -1, 2, 1, -5, 4]

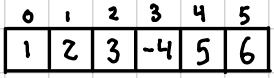
public static int maxSubarray(int[] nums)



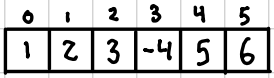
= 13



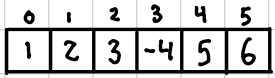
sum = 1, max = 1 [0,0]
[1]



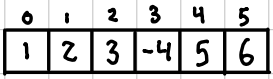
sum = 3, max = 3 [0,1]
[1,2]



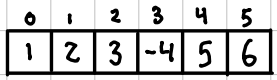
sum = 6, max = 6 [0,2]
[1,2,3]



sum = 2, max = 6
[1,2,3,-4]

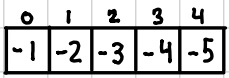


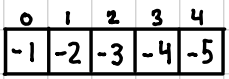
sum = 7, max = 7 [0,4]
[1,2,3,-4,5]

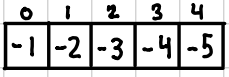


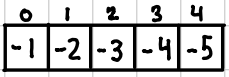
sum = 13, max = 13 [0,5]
[1,2,3,-4,5]

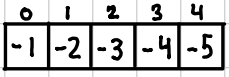
public static int maxSubarray(int[] nums)

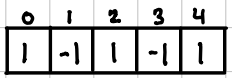


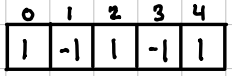


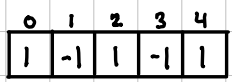


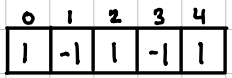












⋮

```
if(nums.length==0)
    return 0
int sum = 0
int max = nums[0]
for(int i=0; i<nums.length; i++)
    sum = sum + nums[i]
    if(sum > max)
        max = sum
    if(nums[i] > max)
        sum = nums[i]
        max = sum
return max
```

seems to work

*Understanding

```
public static int maxSubarray(int[] nums)
    if(nums.length ==0)
        return 0
    int max = nums[0]
    int sum = nums[0]
    for(int i=1; i<nums.length; i++)
        sum = Math.max(nums[i], sum+nums[i])
        max = Math.max(max, sum)
    return max
```

0	1	2	3	4	5	6	7	8	
-2	1	-3	4	-1	2	1	-5	4	s=-2
i									m=-2

0	1	2	3	4	5	6	7	8		
-2	1	-3	4	-1	2	1	-5	4	1 > (-2+1)	→ sum = 1
	i								-2 < 1	→ max = 1

0	1	2	3	4	5	6	7	8		
-2	1	-3	4	-1	2	1	-5	4	-3 < (1+(-3))	→ sum = -2
		i							1 > -2	→ max = 1

0	1	2	3	4	5	6	7	8		
-2	1	-3	4	-1	2	1	-5	4	4 > (-2+4)	→ sum = 4
			i						1 < 4	→ max = 4

0	1	2	3	4	5	6	7	8		
-2	1	-3	4	-1	2	1	-5	4	-1 < (4+(-1))	→ sum = 3
				i					4 > 3	→ max = 4

0	1	2	3	4	5	6	7	8		
-2	1	-3	4	-1	2	1	-5	4	2 < (3+2)	→ sum = 5
					i				4 < 5	→ max = 5

0	1	2	3	4	5	6	7	8		
-2	1	-3	4	-1	2	1	-5	4	1 < (5+1)	→ sum = 6
						i			5 < 6	→ max = 6

0	1	2	3	4	5	6	7	8		
-2	1	-3	4	-1	2	1	-5	4	-5 < (6+(-5))	→ sum = 1
							i		6 > 1	→ max = 6

0	1	2	3	4	5	6	7	8		
-2	1	-3	4	-1	2	1	-5	4	4 < (1+4)	→ sum = 5
								i	6 > 5	→ max = 6