01 firstLast6

```
public boolean firstLast(int[] nums) {
  boolean answer = false;

  int len = nums.length;
  int first = nums[0];
  int last = nums[len-1];

  if (first == 6) {
    answer = true;
  }

  if (last == 6) {
    answer = true;
  }

  return answer;
}

public boolean firstLast6(int[] nums) {
  boolean answer = false;

  int len = nums.length;
  int first = nums[0];
  int last = nums[len-1];

  if (first == 6 || last == 6) {
    answer = true;
  }

  return answer;
}
```

02 sameFirstLast

```
public boolean sameFirstLast(int[] nums) {
  boolean answer = false;

  int len = nums.length;

  if (len >= 1) {
    int first = nums[0];
    int last = nums[len-1];

    if (first == last) {
      answer = true;
    }
  }

  return answer;
}
```

03 makePi

```
public int[] makePi() {
  int[] list = new int[3];

  list[0] = 3;
  list[1] = 1;
  list[2] = 4;

  return list;
}

public int[] makePi() {
  int[] list = { 3, 1, 4 };
  return list;
}

public int[] makePi() {
  int[] list = new int[] { 3, 1, 4 };
  return list;
}
```

04 commonEnd

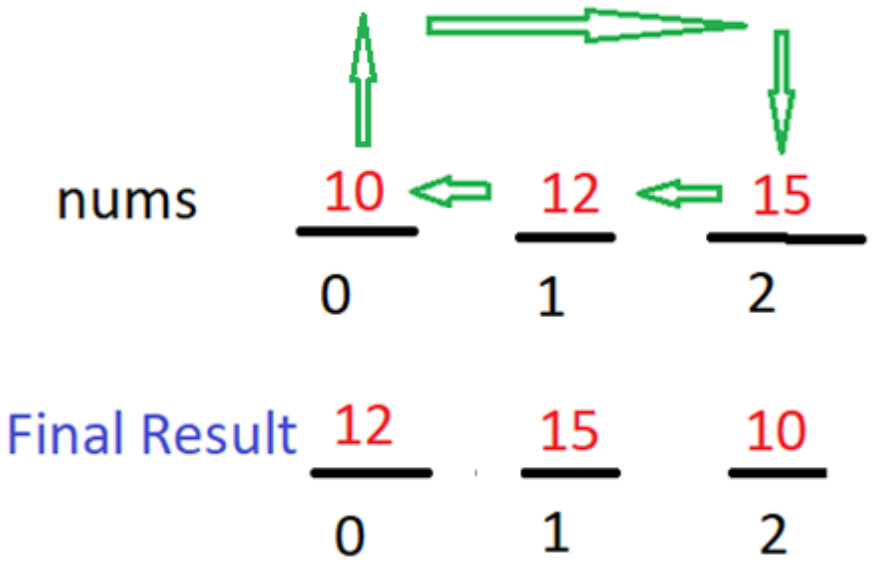
```
public boolean commonEnd(int[] a, int[] b) {
  boolean answer = false;

  int firstA = a[0];
  int firstB = b[0];

  int lenA = a.length;
  int lenB = b.length;

  int lastA = a[lenA-1];
  int lastB = b[lenB-1];
  if (firstA == firstB || lastA == lastB) {
    answer = true;
  }

  return answer;
}

public boolean commonEnd(int[] a, int[] b) {
  boolean answer = false;

  int firstA = a[0];
  int firstB = b[0];
  if (firstA == firstB) {
    answer = true;
  }

  int lenA = a.length;
  int lenB = b.length;

  int lastA = a[lenA-1];
  int lastB = b[lenB-1];
  if (lastA == lastB) {
    answer = true;
  }

  return answer;
}
```

05 sum3

```
public int sum3(int[] nums) {
  int first = nums[0];
  int second = nums[1];
  int third = nums[2];

  int sum = first + second + third;
  return sum;
}
```

06 rotateLeft3

```
public int[] rotateLeft3(int[] nums) {
  int[] list = new int[3];

  int first = nums[0];
  int second = nums[1];
  int third = nums[2];

  list[0] = second;
  list[1] = third;
  list[2] = first;
  return list;
}

public int[] rotateLeft3(int[] nums) {
  int first = nums[0];
  int second = nums[1];
  int third = nums[2];

  int[] list = { second, third, first };

  return list;
}
```



07 reverse3

```
public int[] reverse3(int[] nums) {
  int first = nums[0];
  int second = nums[1];
  int third = nums[2];

  int[] list = { third, second, first  };
  return list;
}

public int[] reverse3(int[] nums) {
  int first = nums[0];
  int second = nums[1];
  int third = nums[2];

  int[] list = new int[3];
  list[0] = third;
  list[1] = second;
  list[2] = first;
  return list;
}
```

08 maxEnd3

```
public int[] maxEnd3(int[] nums) {
  int first = nums[0];
  int last = nums[2];
  int larger = Math.max( first, last );

  int[] list = { larger, larger, larger };
  return list;
}

public int[] maxEnd3(int[] nums) {
  int[] list = new int[3];

  int first = nums[0];
  int last = nums[2];
  int larger = Math.max( first, last );

  list[0] = larger;
  list[1] = larger;
  list[2] = larger;

  return list;
}
```

09 sum2

```
public int sum2(int[] nums) {
  int first = 0; // default empty list
  int second = 0; // default empty list
  int len = nums.length;

  if (len == 1) {
    first = nums[0];
  }

  if (len >= 2) {
    first = nums[0];
    second = nums[1];
  }

  int sum = first + second;;
  return sum;
}
```

10 middleWay

```
public int[] middleWay(int[] a, int[] b) {
  int middleA = a[1];
  int middleB = b[1];
  int[] list = { middleA, middleB };
  return list;
}

public int[] middleWay(int[] a, int[] b) {
  int[] list = new int[2];

  int middleA = a[1];
  int middleB = b[1];

  list[0] = middleA;
  list[1] = middleB;
  return list;
}
```

11 makeEnds

```
public int[] makeEnds(int[] nums) {
  int[] list = new int[2];

  int len = nums.length;
  int first = nums[0];
  int last = nums[len-1];

  list[0] = first;
  list[1] = last;
  return list;
}

public int[] makeEnds(int[] nums) {
  int len = nums.length;
  int first = nums[0];
  int last = nums[len-1];

  int[] list = { first, last };
  return list;
}
```

12 has23

```
public boolean has23(int[] nums) {
  boolean answer = false;

  int first = nums[0];
  if (first == 2 || first == 3) {
    answer = true;
  }

  int second = nums[1];
  if (second == 2 || second == 3) {
    answer = true;
  }

  return answer;
}
```

13 no23

```
public boolean no23(int[] nums) {
  boolean answer = false;

  if ( !has23( nums) ) {
    answer = true;
  }

  return answer;
}

// RE-USE METHOD from Previous Problem
public boolean has23(int[] nums) {
  boolean answer = false;

  int first = nums[0];
  if (first == 2 || first == 3) {
    answer = true;
  }

  int second = nums[1];
  if (second == 2 || second == 3) {
    answer = true;
  }

  return answer;
}
```

14 makeLast

```
public int[] makeLast(int[] a) {
  int lenA = a.length;
  int lastA = a[lenA-1];

  int lenB = lenA * 2;
  int[] b = new int[lenB];

  b[lenB-1] = lastA;

  return b;
}
```

15 double23

```
public boolean double23(int[] nums) {
  boolean answer = false;
  int len = nums.length;
  if (len == 2) {
    int first = nums[0];
    int second = nums[1];
    if (first == 2 && second == 2) {
      answer = true;
    }
    if (first == 3 && second == 3) {
      answer = true;
    }
  }
  return answer;
}

public boolean double23(int[] nums) {
  boolean answer = false;
  int len = nums.length;
  if (len == 2) {
    int first = nums[0];
    int second = nums[1];
    if (first == second) {
      if (first == 2 || first == 3) {
        answer = true;
      }
    }
  }
  return answer;
}

public boolean double23(int[] nums) {
  boolean answer = false;
  int len = nums.length;
  if (len == 2) {
    int first = nums[0];
    int second = nums[1];
    if ( first == second && (first == 2 || first == 3) ) {
      answer = true;
    }
  }
  return answer;
}
```

16 fix23

```
public int[] fix23(int[] nums) {
  int first = nums[0];
  int second = nums[1];
  int third = nums[2];

  if (first ==2 && second == 3) {
    nums[1] = 0;
  }

  if (second ==2 && third == 3) {
    nums[2] = 0;
  }

  return nums;
}
```

17 start1

```
public int start1(int[] a, int[] b) {
  int count = 0;

  int lenA = a.length;
  if (lenA > 0) {
    int firstA = a[0];
    if (firstA == 1) {
      count = count + 1;
    }
  }

  int lenB = b.length;
  if (lenB > 0) {
    int firstB = b[0];
    if (firstB == 1) {
      count = count + 1;
    }
  }

  return count;
}
```

18 biggerTwo

```
public int[] biggerTwo(int[] a, int[] b) {
  int[] list = new int[2];

  int firstA = a[0];
  int secondA = a[1];
  int sumA = firstA + secondA;

  int firstB = b[0];
  int secondB = b[1];
  int sumB = firstB + secondB;

  if (sumA > sumB) {
    list = a;
  }
  if (sumB > sumA) {
    list = b;
  }
  if (sumA == sumB) {
    list = a;
  }

  return list;
}
```

19 makeMiddle

```
public int[] makeMiddle(int[] nums) {

  int len = nums.length;
  int mid = len / 2;

  int firstMiddle = nums[mid-1];
  int secondMiddle = nums[mid];

  int[] list = { firstMiddle, secondMiddle };

  return list;
}
```

```
public int[] makeMiddle(int[] nums) {
  int[] list = new int[2];

  int len = nums.length;
  int mid = len / 2;

  int firstMiddle = nums[mid-1];
  int secondMiddle = nums[mid];
  list[0] = firstMiddle;
  list[1] = secondMiddle;

  return list;
}
```

20 plusTwo

```
public int[] plusTwo(int[] a, int[] b) {
  int[] c = new int[4];

  int firstA = a[0];
  int secondA = a[1];
  int firstB = b[0];
  int secondB = b[1];

  c[0] = firstA;
  c[1] = secondA;
  c[2] = firstB;
  c[3] = secondB;

  return c;
}
```

21 swapEnds

```
public int[] swapEnds(int[] nums) {
  int len = nums.length;

  int first = nums[0];
  int last = nums[len-1];

  nums[0] = last;
  nums[len-1] = first;

  return nums;
}
```

22 midThree

```
public int[] midThree(int[] nums) {

  int len = nums.length;
  int mid = len / 2;

  int beforeMiddle = nums[mid-1];
  int middle = nums[mid];
  int afterMiddle = nums[mid+1];
  int[] list = {  beforeMiddle, middle, afterMiddle };
  return list;
}
```

```
public int[] midThree(int[] nums) {
  int[] list = new int[3];

  int len = nums.length;
  int mid = len / 2;

  int beforeMiddle = nums[mid-1];
  int middle = nums[mid];
  int afterMiddle = nums[mid+1];

  list[0] = beforeMiddle;
  list[1] = middle;
  list[2] = afterMiddle;

  return list;
}
```

23 maxTriple

```
public int maxTriple(int[] nums) {
  int len = nums.length;
  int mid = len / 2;

  int first = nums[0];
  int last = nums[len-1];
  int middle = nums[mid];

  int largestFL = Math.max( first,last );
  int largestFLM = Math.max( largestFL, middle );

  return largestFLM;
}
```

24 frontPiece

```
public int[] frontPiece(int[] nums) {
  int len = nums.length;
  int size = Math.min(2,len);
  int[] list = new int[size];

  if (len == 1) {
    int first = nums[0];
    list[0] = first;
  }
  if (nums.length >= 2) {
    int first = nums[0];
    int second = nums[1];
    list[0] = first;
    list[1] = second;
  }

  return list;
}

public int[] frontPiece(int[] nums) {
  int[] list = { };

  int len = nums.length;
  if (len == 1) {
    int first = nums[0];
    list = new int[] { first }; // reinitialize
  }
  if (nums.length >= 2) {
    list = new int[2];
    int first = nums[0];
    int second = nums[1];
    list = new int[] { first, second }; // reinitialize
  }
  return list;
}
```

25 unlucky1

```
public boolean unlucky1(int[] nums) {
  boolean unlucky = false;
  int len = nums.length;

  if (len >= 2) {
    int first = nums[0];
    int second = nums[1];
    if (first == 1 && second == 3) {
      unlucky = true;
    }

    if (len >= 3) {
      int third = nums[2];
      if (second == 1 && third == 3) {
        unlucky = true;
      }
    }

    int last = nums[len-1];
    int secondToLast = nums[len-2];
    if (secondToLast == 1 && last == 3) {
      unlucky = true;
    }

  }

  return unlucky;
}
```

26 make2

```
public int[] make2(int[] a, int[] b) {
  int[] list = new int[2];
  int lenA = a.length;

  if (lenA >= 2) {
    int firstA = a[0];
    int secondA = a[1];
    list = new int[] { firstA, secondA };  // reinitialize
  }

  if (lenA == 1) {
    int firstA = a[0];
    int firstB = b[0];
    list = new int[] { firstA, firstB };  // reinitialize
  }

  if (lenA == 0) {
    int firstB = b[0];
    int secondB = b[1];
    list = new int[] { firstB, secondB };  // reinitialize
  }

  return list;
}
```

27 front11

```
public int[] front11(int[] a, int[] b) {
  int[] list = {};
  int lenA = a.length;
  int lenB = b.length;

  if (lenB == 0 && lenA >= 1) {
    int firstA = a[0];
    list = new int[] { firstA };  // reinitialize
  }

  if (lenA == 0 && lenB >= 1) {
    int firstB = b[0];
    list = new int[] { firstB };  // reinitialize
  }

  if (lenA >= 1 && lenB >= 1) {
    int firstA = a[0];
    int firstB = b[0];
    list = new int[] { firstA, firstB };  // reinitialize
  }

  return list;
}
```