

An abstract graphic on the left side of the slide, consisting of a network of thin, light-blue lines and small circles, resembling a circuit board or a neural network. The lines are vertical and horizontal, with some diagonal connections, and the circles are placed at various points along these lines.


GROUP 2

XAITHENG

MARSHALL

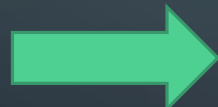
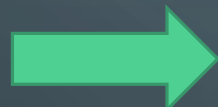
RESULTS

group 2 took place 4. The sum of places is 8, the sum of medians is 1797.0

- Group 18 (not in the class): Kevin Arhelger
- Group 17 (not in the class): Dan Stelljes, Shawn Seymour
- Group 11 (1st place in the class): Shawn, Brian
- Group 2 (2nd place in the class): Xai, Marshall 
- Group 1 (3rd place in the class): Roch, Andy

BIG-PICTURE DESCRIPTION

- Created Object Array
- Presort original list
- Populate Object Array
- Sort Object Array by product of primes
- Convert back to Strings



```
public static String[] sort(String[] toSort) {  
    elementAndFactorProduct[] twoSort = new elementAndFactorProduct[toSort.length];  
  
    Arrays.sort(toSort);  
  
    twoSort[0] = new elementAndFactorProduct(toSort[0]);  
  
    for(int i = 1; i < toSort.length; i++){  
        if(toSort[i].equals(toSort[i-1])){  
            twoSort[i] = new elementAndFactorProduct(twoSort[i-1].element, twoSort[i-1].productOfPrimes);  
        } else {  
            twoSort[i] = new elementAndFactorProduct(toSort[i]);  
        }  
    }  
  
    Arrays.sort(twoSort, new PrimesComparator());  
    for(int i = 0; i < toSort.length; i++){  
        toSort[i] = twoSort[i].element;  
    }  
    return toSort;  
}
```

BIG-PICTURE DESCRIPTION


- Created Object Class
elementAndFactorProduct

```
public class elementAndFactorProduct {  
    public static int[] knownPrimes = new int[]{2,3,5};  
    public String element = "";  
    public long productOfPrimes = 0;  
  
    /*  
     * Constructor:  
     * Input: String (presumably a number)  
     * Converts the String to a long  
     * Stores the original element value as a long, and the 'productOfPrimes' value as a long  
     */  
    public elementAndFactorProduct(String ele){  
        element = ele;  
        long convertedElement = new Long(element);  
        productOfPrimes = productOfPrimeFactors(convertedElement);  
    }  
  
    /*  
     * Constructor 2:  
     * Input: String (presumably a number), long  
     * Stores the original element value as a long, and the second input value as a long ('productOfPrimes')  
     */  
    public elementAndFactorProduct(String ele, long PoP){  
        element = ele;  
        productOfPrimes = PoP;  
    }  
}
```

EFFICIENCY

- Theoretical Efficiency:
 - Worst Case: A bunch of different large primes
 - Best Case: All numbers are 1
 - Both Cases are in Efficiency Class $\text{BigTheta}(n^2)$
 - To calculate the key values (product of lowest primes) there is a loop to check each element, and a loop to calculate the prime factors of each element

INTERESTING FEATURES

- Holder = the element
- Bound = square root of the element
- Steps for calculating prime:
 - Checks if number is 1
 - Checks if it has 2, 3, or 5 as a factor
 - Performs this for loop code chunk: 
 - Checks for a second prime above Bound if it only has one factor

```
//edited to match prime number sequence 1 and 2 (6n+5 and 6n+1 respectively)
for (int i = 1; (6*i+1) <= bound; i++) {
    if ((holder % (6*i+1)) == 0) { // the first found factor must be prime
        if (prime1 == 1) {
            prime1 = (6*i + 1);
        } else { // the second found factor is a prime or a power of the first one
            if ((6*i+1) % prime1 != 0) { // now we know it's a prime
                prime2 = (6*i + 1);
                break;
            }
        }
    }
}

if ((holder % (6*i+5)) == 0) { // the first found factor must be prime
    if (prime1 == 1) {
        prime1 = (6*i + 5);
    } else { // the second found factor is a prime or a power of the first one
        if ((6*i+5) % prime1 != 0) { // now we know it's a prime
            prime2 = (6*i + 5);
            break;
        }
    }
}
}
```

CLOSING THOUGHTS

- What worked:
 - Object Class
 - Prime finding sequences
- What didn't work:
 - Having an array of predefined prime numbers
 - Tried Quicksort
 - Tried a separate case for product of primes value 2
- What we would have done differently
 - Found a way to do this without solving for primes