Week 4

Big-O, Comparing, Sorting

Big-O Big-Question: What is Big-O?

General definition: “How quickly runtime grows relative to the input, as the input gets arbitrarily large”

Breakdown:

- How quickly the runtime grows...
- ...relative to the input…
- ...as the input gets arbitrarily large

Source: https://www.interviewcake.com/article/java/big-o-notation-time-and-space-complexity

Big-O Big-Misconception

Big-O does **NOT** describe how fast your algorithm is. It is a measure of the rate your runtime increases as input size increases.

```
// O(1)
public int fastMax(int a, int b)
{
    if(a > b)
        return a;
    else
        return b;
}
```

```
// O(1)
public int slowMax(int a, int b)
{
    // Take a long, long nap
    Thread.sleep(1000000000);

    if(a > b)
        return a;
    else
        return b;
}
```

It’s time for...

Big-O Big-Trivia!!!

Guess the correct Big-O time for these examples!

Question 1

```
public void swap(int[] arr, int i, int j)
{
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

Question 1

Answer: O(1)

```
public void swap(int[] arr, int i, int j)
{
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

Question 2

```
public long sum(int[] arr)
{
    long total = 0;

    for(int i : arr)
        total += i;

    return total;
}
```

Question 2

Answer: O(n)

```
public long sum(int[] arr)
{
    long total = 0;

    for(int i : arr)
        total += i;

    return total;
}
```

Question 3

```
public boolean hasDuplicates(int[] arr)
{
    for(int i = 0; i < arr.length; i++)
    {
        for(int j = i+1; j < arr.length; j++)
        {
            if(arr[i] == arr[j]) {
                return true;
            }
        }
    }
    return false;
}
```

Question 3

Answer: $O(n^2)$

```
public boolean hasDuplicates(int[] arr)
{
    for(int i = 0; i < arr.length; i++)
    {
        for(int j = i+1; j < arr.length; j++)
        {
            if(arr[i] == arr[j]) {
                return true;
            }
        }
    }
    return false;
}
```

Question 4

```
public int roughLog2(int n)
{
    int sum = 0;
    for(int i = 1; i < n; i *= 2)
    {
        sum += 1;
    }

    return sum;
}
```

Question 4

Answer: O(log n)

```
public int roughLog2(int n)
{
    int sum = 0;
    for(int i = 1; i < n; i *= 2)
    {
        sum += 1;
    }

    return sum;
}
```

Question 5

```
public void printPairSums(int[] arr)
{
    System.out.println("These are the numbers:");
    for(int number : arr)
    {
        System.out.println(number);
    }

    System.out.println("And these are their sums:");
    for(int number1 : arr)
    {
        for(int number2 : arr)
        {
            System.out.println(number1 + number2);
        }
    }
}
```

Source: https://www.interviewcake.com/article/java/big-o-notation-time-and-space-complexity

Question 5

Answer: $O(n^2)$

```
public void printPairSums(int[] arr)
{
    System.out.println("These are the numbers:");
    for(int number : arr)
    {
        System.out.println(number);
    }

    System.out.println("And these are their sums:");
    for(int number1 : arr)
    {
        for(int number2 : arr)
        {
            System.out.println(number1 + number2);
        }
    }
}
```

Source: https://www.interviewcake.com/article/java/big-o-notation-time-and-space-complexity

Question 6

```
public int compute(int n)
{
    int total = 0;
    for(int i = 1; i < n; i++)
    {
        int k = 1;
        while(k < i)
        {
            k = k + k;
        }
        while(k > 1)
        {
            k /= 2;
            total += 1;
        }
    }
    return total;
}
```

Source: http://www.cburch.com/cs/280/test/r0/print.html

Question 6

Answer: O(n log n)

```
public int compute(int n)
{
    int total = 0;
    for(int i = 1; i < n; i++)
    {
        int k = 1;
        while(k < i)
        {
            k = k + k;
        }
        while(k > 1)
        {
            k /= 2;
            total += 1;
        }
    }
    return total;
}
```

Source: http://www.cburch.com/cs/280/test/r0/print.html

Question 7

```
public int[] combine(int[] arr1, int[] arr2)
{
    int n = arr1.length;
    int m = arr2.length;
    int[] out = new int[n + m];

    for(int i = 0; i < n; i++)
    {
        out[i] = arr1[i];
    }
    for(int i = 0; i < m; i++)
    {
        out[i + n] = arr2[i];
    }
    return out;
}
```

Question 7

Answer: O(n+m)

```
public int[] combine(int[] arr1, int[] arr2)
{
    int n = arr1.length;
    int m = arr2.length;
    int[] out = new int[n + m];

    for(int i = 0; i < n; i++)
    {
        out[i] = arr1[i];
    }
    for(int i = 0; i < m; i++)
    {
        out[i + n] = arr2[i];
    }
    return out;
}
```

Question 8

```java
public int getLargestNumber(int[] arr)
{
    //Too many values, just guess
    if(arr.length > 100000)
    {
        return Integer.MAX_VALUE;
    }

    int largest = arr[0];
    for(int i : arr)
    {
        largest = Math.max(largest, i);
    }

    return largest;
}
```

Question 8

Answer: O(1)

```
public int getLargestNumber(int[] arr)
{
    //Too many values, just guess
    if(arr.length > 100000)
    {
        return Integer.MAX_VALUE;
    }

    int largest = arr[0];
    for(int i : arr)
    {
        largest = Math.max(largest, i);
    }

    return largest;
}
```

Question 9

```
public static boolean isPrime(int x)
{
    if(x <= 1)      { return false; }
    if(x == 2)      { return true; }
    if(x % 2 == 0) { return false; }

    for(int i = 3; i <= Math.sqrt(x); i += 2)
    {
        if(x % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

Question 9

Answer: O(√n)

```
public static boolean isPrime(int x)
{
    if(x <= 1)      { return false; }
    if(x == 2)      { return true; }
    if(x % 2 == 0) { return false; }

    for(int i = 3; i <= Math.sqrt(x); i += 2)
    {
        if(x % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

Big-O Big-Final Question

```
public void bogoSort(int[] arr)
{
    while(isUnsorted(arr))
    {
        randomlyShuffle(arr);
    }
}
```

Big-O Big-Final Question **Answer: Undefined**

```
public void bogoSort(int[] arr)
{
    while(isUnsorted(arr))
    {
        randomlyShuffle(arr);
    }
}
```

Moral of the story, please do not ask questions like this

Comparator vs. Comparable

Comparable

```
class Point implements Comparable<Point> {

    int x;
    int y;

    // Sort by x values
    public int compareTo(Point p)
    {
        return this.x - p.x;
    }
}
```

Comparator

```
class PointXComparator implements Comparator<Point> {

    @Override
    public int compare(Point p0, Point p1)
    {
        return p0.x - p1.x;
    }

}
```

How to use Comparable/Comparator

```
List<Point> list = new ArrayList<Point>();

// Using Comparable
Collections.sort(list);

//Using Comparator
Collections.sort(list, new PointXComparator());
```

Benefit of Comparator

You can use anonymous classes to make that comparator!

```
//Using Comparator
Collections.sort(list, new Comparator<Point>() {

    @Override
    public int compare(Point p0, Point p1) {
        return p0.x - p1.x;
    }

});
```

When to use Comparable/Comparator

Comparable

- When there's only one way to sort objects
- When there's a "default" way to sort objects

Comparator

- When there's more than one way to sort objects, so more than one type of comparison is needed

Sorting

This is a helpful website which presents many if not all the algorithms covered in this class. If you're having trouble understanding any sorting algorithms, or in the future binary tree algorithms, look at this site.