CS 61B Spring 2020 | Small Group Tutoring
Section 5: Complexity, Number Representation | Worksheet 7

1 List'em all!

List all the asymptotic runtimes from quickest to slowest.
$\theta(n^2), \theta(n^{0.5}), \theta(\log n), \theta(3^n), \theta(c), \theta(n^{n!})\theta(n), \theta(n\log n), \theta(n!), \theta(n^n), \theta(2^n)$

Solution: $\theta(c), \theta(\log n), \theta(n^{0.5}), \theta(n), \theta(n\log n), \theta(n^2), \theta(2^n), \theta(3^n), \theta(n!), \theta(n^n), \theta(n^{n!})$

2 What's that runtime?

For each of the methods below, please specify the runtime in BigO, BigΘ or BigΩ Notation. Please give the tightest bound possible.

Solution: $\theta(n^3)$, the i loop runs n, the j loop runs n, each time doing a function that takes n, so $n*n*n = n^3$.

_____
```
private static void f(int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            linear(n); // runs in linear time with respect to input
        }
    }
}
```

Solution: $\theta(nlogn)$,

_____
```
private static void g(int n) {
    if (n < 1) return;
    for(int i = 0; i < n; i++) {
       linear(100);
    }
    g(n/2);
    g(n/2);
}
```

Solution: $O(n)$

```
_________ private static void h(int n) {
              Random generator = new Random();
              for(int i = 0; i < n; i++) {
                 if(generator.nextBoolean()) {
                     /* nextBoolean returns true with
                        probability .5. */
                     break;
                 }
              }
          }
```

Solution: $\theta(n)$

```
_________ private static void i(int n) {
              if (n < 1) return;
              for(int i = 0; i < n; i++) {
                 System.out.println("Yow!");
              }
              i((999 * n) / 1000);
          }
```

3 How fast?

Given a IntList of length N, provide the runtime bound for each operation. Recall that IntList is the naive linked list implementation from class.

Operations	Runtime
size()	$\theta(N)$
get(int index)	$O(N)$
addFirst(E e)	$\theta(1)$
addLast(E e)	$\theta(N)$
addBefore(E e, Node n)	$O(N)$
remove(int index)	$O(N)$
remove(Node n)	$O(N)$
reverse()	$\theta(N)$

4 Sum 'em Up

1. Define a function, sumTo, that takes a sorted int[] array and an int x and returns true if two numbers in the array sum to x and false otherwise. For example, if given the following input: [1, 2, 4, 7, 8, 10] and $x = 12$, the function should return true.

   $\theta(n^2)$ solution:

```
public boolean sumTo(int[] nums, int x) {
    for(int i = 0; i < nums.length - 1; i++) {
        for(int j = i + 1; j < nums.length; j++) {
            if (nums[i] + nums[j] == x) {
                return true;
            }
        }
    }
    return false;
}
```

   $\theta(n)$ solution, utilizing the fact that the array is ordered, we maintain two pointers to small and large values, increasing the small pointer when the sum is too small, and decreasing the large pointer when the sum is too big. If the pointers ever point to the same element, we know that no pair will sum to x:

```
public boolean sumTo(int[] nums, int x) {
    int front = 0;
    int back = nums.length - 1;
    while (front != back) {
        if (nums[front] + nums[back] == x) {
            return true;
        }
        else if (nums[front] + nums[back] > x) {
            back -= 1;
        } else {
            front += 1;
        }
    }
    return false;
}
```

2. Provide the tightest possible runtime bound on your solution.
   Solution: See solution above.

5 Number Representation

Convert the following **4**-bit numbers from signed integers to binary, and from binary to signed integers.

Decimal: 7 Binary: **Solution:** 0111

Decimal: -5 Binary: **Solution:** 1011 (5 = 0101, so if we flip the bits and add one we get 1010 + 1 = 1011 or we can compute it directed with $-1*2^3+1*2^1+1*2^0=-8+2+1=-5$)

Decimal: **Solution:** -8 Binary: 1000

Decimal: (3 + 7) Binary: **Solution:** 1010 (math in binary: 0011 + 0111 = 1010)

Now what is the decimal representation of the binary number from the previous part? **Solution:** -6
Integer overflow because 10 is too large to represent with 4 bits as a signed number

Now for the questions below, consider that we are no longer working with 4-bit numbers, but rather 64 bit numbers.

Decimal: $1 << 2$ Binary: **Solution:** 0100

What is the decimal representation of this? **Solution:** 4

Decimal: $10 >> 2$ Binary: **Solution:** 0010

What is the decimal representation of this? **Solution:** 2
Maybe worth nothing that shifting left by x is multiplying by 2^x, shifting right is floor dividing by 2^x, where x is positive (as long as there is no overflow).

Given a number x, how do we determine if it's even or odd using bit and boolean operators?
Solution: If x & $1 == 1$, then it is odd.

How do we determine whether x is a power of 2?
Solution: In bit representation, x can only have a single 1. If $(x \;\&\; (x-1)) == 0$, then x is a power of 2.
(ex: 4 is 0100 and 3 is 0011. 6 is 0110 and 5 is 0101. Notice how if x is a power of 2, then x - 1 turns every 0 after the 1 in x into a 1 and the single 1 in x into a 0.)

What is a number that can be represented as a 64 bit signed binary number but its absolute value cannot? (x can be represented but $|x|$ cannot)
Solution: Integer.MIN_VALUE
When we flip the bits and add 1, we get back Integer.MIN_VALUE. Integer.MIN_VALUE in binary is 10000....00, so flipping the bits we get 01111...11. When we add 1, we get back 10000...00.