# CS 61B Discussion 8 Spring 2015

## 1 Conceptual Check

Order the following big-O runtimes from most to least efficient:

$O(n \log n), O(1), O(2^n), O(n^2), O(\log n), O(n), O(n!)$

$O(1) \subset O(logn) \subset O(n) \subset O(nlogn) \subset O(n^2) \subset O(2^n) \subset O(n!)$

Are the statements in the right column true or false? If false, correct the asymptotic notation $(\Omega, \Theta, O)$. $\Omega(\cdot)$ is the opposite of $O(\cdot)$, i.e. $f(n) = \Omega(g(n)) \iff g(n) = O(f(n))$.

| | | | |
|---|---|---|---|
| $f(n) = 100$ | $g(n) = 1$ | $f(n) \in \Omega(g(n))$ | True |
| $f(n) = n^2 + n$ | $g(n) = 0.1n^2$ | $f(n) \in \Theta(g(n))$ | True |
| $f(n) = 2^n$ | $g(n) = 2^{2n} + 100$ | $f(n) \in \Theta(g(n))$ | False: $O$ |
| $f(n) = n^{100}$ | $g(n) = 2^n + n\ logn$ | $f(n) \in O(g(n))$ | True |
| $f(n) = 3 \log n$ | $g(n) = n^2 + n + \log n$ | $f(n) \in \Omega(g(n))$ | False: $O$ |
| $f(n) = n \log n$ | $g(n) = (\log n)^2$ | $f(n) \in O(g(n))$ | False: $\Omega$ |

## 2 Analyzing Runtime

Give the worst case runtime in $\Theta(\cdot)$ notation. Extra: Give the best case runtime in $\Theta(\cdot)$.

A. Use $M$ and $N$ in your result. `bump()` is a constant time function that returns a boolean.
Worst: $\Theta(M+N)$
Best: $\Theta(N)$

```
int j = 0;
for (int i = 0; i < N; i += 1) {
    for (; j < M; j += 1) {
        if (bump(i, j))
            break; // terminates only the inner loop
    }
}
```

B. Use $N$ in your result.
$\Theta(N^2)$

```
public static boolean mystery(int[] arr) {
    arr = bilbosort(arr); // creates sorted copy of arr in Θ(N log N) time
    int N = arr.length;
    for (int i = 0; i < N; i += 1) {
        boolean x = false;
        for (int j = 0; j < N; j += 1) {
            if (i != j && arr[i] == arr[j])
                x = true;
        }
        if (!x)
            return false;
```

```
    }
    return true; } // } on same line for vertical space reasons
```

C. Use $N$ in your result, where $N$ is the length of `arr`. Assume `arr` is a sorted array of unique elements. Say we call `mystery2(arr, 0, arr.length)`.
$\Theta(log N)$

```
public static int mystery2(int[] arr, int low, int high) {
    if (high <= low)
        return -1;
    int mid = (low + high) / 2; // (later, see http://goo.gl/gQI0FN )
    if (arr[mid] == mid)
        return mid;
    else if (mid > arr[mid])
        return mystery2(arr, mid + 1, high);
    else
        return mystery2(arr, low, mid);
}
```

**Amanda's Additional for Awesome Asymptotic And Algorithmic Allstars with Alliteration:** What are `mystery()` and `mystery2()` doing? Assuming an int can appear in `arr` at most twice, can you rewrite `mystery()` with a better runtime?

`mystery()` returns **false if** there are unique ints in the array, and **true if** all ints have a duplicate. A $\Theta(N)$ algorithm is to XOR all ints together and **return true if** the result is 0.

`mystery2()` looks **for** an index i such that arr[i] == i and returns it, otherwise it returns -1 **if** no such index exists.

## 3 Optimizing Algorithms (Extra Problem)

Given an integer x and a **sorted** array A[] of N distinct integers, design an algorithm to find if there exists indices i and j such that `A[i] + A[j] == x`.

Let's start with the naive solution.

```
public static boolean findSum(int[] A, int x) {
    for (int i = 0; i < A.length; i++){
        for (int j = 0; j < A.length; j++){
            if (A[i] + A[j] == x)
                return true;
            }
        }
    }
    return false;
}
```

Can we do this faster? Hint: Does order matter here?

```
public static boolean findSumFaster(int[] A, int x){
    int left = 0;
    int right = A.length - 1;
    while (left <= right){
        if (A[left] + A[right] == x)
            return true;
        else if (A[left] + A[right] < x)
```

```
            left++;
        else
            right--;
    }
    return false;
}
```

What is the runtime of both these algorithms?

$N = A.length$

Naive: $\Theta(N^2)$

Optimized: $\Theta(N)$