CS 61B Discussion 8 Spring 2016

1 Which is faster?

For each example below, there are two algorithms solving the same problem. Given the asymptotic runtimes for each, is one of the algorithms **guaranteed** to be faster? If so, which? And if neither is always faster, explain why. Assume the algorithms have very large input (so N is very large).

A. Algorithm 1: $\Theta(N)$, Algorithm 2: $\Theta(N^2)$

Algorithm 1: $\Theta(N)$ - straight forward, Θ gives tightest bounds.

Algorithm 1: $\Omega(N)$, Algorithm 2: $\Omega(N^2)$

Neither, something in $\Omega(N)$ could also be in $\Omega(N^2)$

Algorithm 1: $O(N)$, Algorithm 2: $O(N^2)$

Neither, something in $O(N^2)$ could also be in $O(1)$

Algorithm 1: $\Theta(N^2)$, Algorithm 2: $O(\log N)$

Algorithm 2: $O(\log N)$ - Algorithm 2 cannot run SLOWER than $O(\log N)$ **while** Algorithm 1 is constrained on best and worst **case** by $\Theta(N^2)$.

Algorithm 1: $O(N \log N)$, Algorithm 2: $\Omega(N \log N)$

Neither, Algorithm 1 CAN be faster, but is not guaranteed - it is guaranteed to be "as fast as or faster" than Algorithm 2.

Would your answers above change if we did not assume that N was very large?

Technically, no. But asymptotics are only applicable when considering behavior as N gets large. Consider **this** example: N^2 is asymptotically larger than $10000N$, yet when N is less than 10000, $10000N$ is larger than N^2.

2 More Runtime Analyzing

A. How many times is `lobsterPainting` called? Give your answer in Θ notation in terms of N, assuming `lobsterPainting` does not crash or call any methods.

```java
for (int i = 1; i < N/2; i++) {
    for (int j = i - 1; j < N/2 + 1; j++) {
        lobsterPainting(i, j);
    }
}
```

$\Theta(N^2)$ - First run of inner loop is $\frac{N}{2}$, next is $\frac{N}{2}-1$, etc. This goes $\frac{N}{2}$ times, which makes it $\Theta((\frac{N}{2})^2)$, so asymptotically, it is $\Theta(N^2)$

B. How about here?

```java
for (int i = N - 1; i > 0; i /= 2) {
    for (int j = 0; j < i; j++) {
        lobsterPainting(i, j);
    }
}
```

$\Theta(N)$ - If you add the numbers up (ignoring the -1 because it does not matter), it is N + N/2 + N/4 + ... which is less than 2N.
They have seen **this** example in lecture except it went $i* = 2$ instead of

$i/=2$. Even though outer loop looks $\log N$, because the number of times the inner one changes, it is linear rather than anything **else**

C. Bonus: And here?

```
public static void crabDrawing(int N) {
    for (int i = 1; i < N; i *= 2) {
        lobsterPainting(i, i);
        crabDrawing(i);
    }
}
```

$\Theta(N)$ - This one is a little funky, it is actually $\Theta(2^{\log_x N})$ where x is how you scale i (so $x=2$ in **this case** because $i*=2$). Explanation:

First, observe that you make a call **for** each power of 2 less than N

The i-th recursive call in turn makes calls **for** each power of 2 less than i

Starting with the base **case** of $N=1$, there is 1 lobster

The next call (2^1) has 1 + 1 = 2 lobsters

The next call (2^2) has lobsters equal to ((lobsters **for** $N=1$) + (lobsters **for** $N=2$) + 1) = 4

And so forth, the i-th call (with $N=2^i$) has 2^i lobsters. (You can show **this** by induction)

The solution becomes $\sum_{i=0}^{\lfloor(\rfloor log_2 N)} 2^i$ which asymptotically is $2^{log_2 N}=N$

3 More? Of Course More

Describe the best-case and worst-case runtimes of the function individually using Θ. Then use them to describe the overall runtime of the function in terms of Θ (if possible) or O/Ω if not.

A. Assume `arr` is a **sorted** array of **unique** elements of size N. Example of calling this method would be: `hopps(sortedArr, 0, sortedArr.length)`.

Best **case** $\Theta(1)$ (Big-Omega)

Worst **case** $\Theta(\log N)$ (Big-O)

```
public static int hopps(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 hopps(arr, mid + 1, high);
    else
        return hopps(arr, low, mid);
}
```

Bonus: What is `hopps` doing?

Finding **if** there is an element in arr such that arr[i] = i and returning it. If there is no such element, then it returns -1.

B. Assume `str` is a String of characters of size N.

$\Theta(N)$ all around (best and worst)

The second **for** loop may end early, but the first always runs **for** N iterations.

```java
public static char wilde(String str) {
    Map<Character,Integer> map = new HashMap<>();
    for (char c : str.toCharArray()) {
        if (map.containsKey(c)) {
            map.put(c, map.get(c) + 1);
        } else {
            map.put(c, 1);
        }
    }
    for (int i = 0; i < str.length(); i++) {
        if (map.get(str.charAt(i)) == 1) {
            return str.charAt(i);
        }
    }
    return 0; // 0 represents the NULL character
}
```

Bonus: What is `wilde` doing?

`Finds the first unique char in str and returns it. If there is no such unique char, return 0.`

Bonus's Bonus: Can you do it with only 1 `for` loop?

`Use 2 data structures instead of 1, using one to store all the elements that have had only 1 occurrence so far and another to store all the characters we have seen that have duplicates:`

```java
Set<Character> repeats = new HashSet<>();
List<Character> uniques = new ArrayList<>();
for (int i = 0; i < str.length(); i++) {
    char chara = str.charAt(i);
    if (repeats.contains(chara)) {
        continue;
    }
    if (uniques.contains(chara)) {
        uniques.remove((Character) chara);
        repeats.add(chara);
    } else {
        uniques.add(chara);
    }
}
return uniques.get(0);
```

`NOTE: This algorithm is NOT linear time - removing from the ArrayList takes N time, so this algorithm is actually` $\Theta(N^2)$

4 Have You Ever Went Faster? (Extra)

Given an integer x and a **sorted** array A[] of N distinct integers, design an algorithm to find if there exists distinct indices i, j, and k such that `A[i] + A[j] + A[k] == x`. Feel free to write pseudocode instead of Java. Your code should run in $\Theta(N^2)$ time.

```
public static boolean sum3(int[] arr, int x) {
    for(int i = 0; i < arr.length; i++) {
        int j = i+1;
        int k = arr.length-1;
        while(j < k) {
            int sum = arr[i] + arr[j] + arr[k];
            if(sum == x) {
                return true;
            } else if(sum < x) {
                j++;
            } else if(sum > x) {
                k--;
            }
        }
    }
    return false;
}
```