CSM 61B
Fall 2022

Disjoint Sets & Asymptotics

Mentoring 6: September 28 - 30, 2022

1 Asymptotics

The **running time** of a program can be modeled by the number of instructions executed by the computer. To simplify things, for this problem suppose arithmetic operators (`+`, `-`, `*`, `/`), logical operators (`&&`, `||`, `!`), comparison (`==`, `<`, `>`), assignment, field access, array indexing, and so forth take 1 unit of time. `(6 + 3 * 8) / 3` would take 3 units of time, one for each arithmetic operator.

While this measure is fine for simple operations, many problems in computer science depend on the size of the input: `fib(3)` executes almost instantly, but `fib(10000)` will take much longer to compute.

Asymptotic analysis is a method of describing the run-time of an algorithm *with respect* to the size of its input. We can now say,

> The run-time of `fib` is, at most, within a factor of 2^N where N is the size of the input number.

Or, in formal notation, `fib(n)` $\in O(2^N)$.

Formulas:

- If there are N terms in the sequence $1, 2, 3, 4, 5, \cdots, N$, or N/C terms in the sequence for some constant, C.

  $1 + 2 + 3 + 4 + 5 + \cdots + N \in \Theta(N^2)$

- If there are $\log_2 N$ terms in the sequence $1, 2, 4, 8, 16, \cdots, N$, or $\log_2 N/C$ terms for some constant, C.

  $1 + 2 + 4 + 8 + 16 + \cdots + N \in \Theta(N)$

  Or, the number of nodes in a bushy tree is equal to k^h where k is the branching factor and h is the height of the tree

  Remember: All logarithms are proportional to each other by the Change of Base formula.

Tips:

1. Try some small sample inputs to get a better intuition of what the function's runtime is like. What is the function doing with the input? How does the runtime change as the input size increases? Can you spot

any 'gotchas' in the code that might invalidate our findings for larger inputs?

2. Try to find the best and worst cases of the function given what you know about how the function works. This is a good sanity check for your later observations.
3. If the function is recursive, draw a call tree to map out the recursive calls. This breaks the problem down into smaller parts that we can analyze individually. Once each part of the tree has been analyzed, we can then reassemble all the parts to determine the overall runtime of the function.
4. If the function has a complicated loop (that doesn't involve recursion), draw a bar graph to map out how much work the body of the loop executes for each iteration.

1.1 Define, in your own words, each of the following asymptotic notation.

(a) O

(b) Ω

(c) Θ

1.2 Give a tight asymptotic runtime bound for `containsZero` as a function of N, the size of the input array in the *best case, worst case, and overall.*

```
public static boolean containsZero(int[] array) {
    for (int value : array) {
        if (value == 0) {
            return true;
        }
    }
    return false;
}
```

2 Weighted Quick Union

```
public class WQUnion {
    private int[] parentArr; // Each node's index in this array contains the index of their
   parent node. If the node has no parent, the index stores the negative of the size of the
    tree it is in.
    public WQUnion<Type>(int size);
        parentArr = new int[size];
        for (int i = 0; i < size; i++) {
            parentArr[i] = -1;
        }
    private int findRoot(int x) {
        int root = x;
        while (parentArr[root] >= 0) {
            root = parentArr[root];
        }
        return root;
    }
    public void connect(int x, int y) {
        ???
    }
    public boolean isConnected(int x, int y) {
        if(findRoot(x) == findRoot(y)) {
            return true;
            }
        return false;
    }
}
```

2.1 Implement `connect` for `UnionTree` such that for nodes x and y, the root of the node with the smaller sized union becomes the child of the root of the node with the larger sized union. (If they are equally sized, make the child of the root of x the child of y.)

```
public void connect(int x, int y) {
        ______________________________________________;
        ______________________________________________;
        if (_________________________________________________________) {
            ______________________________ += ___________________________;
            ______________________________ = ____________________________;
        }
        else {
            ______________________________ += ___________________________;
            ______________________________ = ____________________________;
        }
    }
```

2.2 What is the worst case bound for height increase per node increase, given the `connect` method you implemented in the last part?

2.3 What are the worst case and best case runtime bounds for `connect`?

3 Scarlet Koi

3.1 (a) Give a $O(\cdot)$ runtime bound as a function of N, `sortedArray.length`.

```
private static boolean scarletKoi(int[] sortedArray, int x, int start, int end) {
    if (start == end || start == end - 1) {
        return sortedArray[start] == x;
    }
    int mid = end + ((start - end) / 2);
    return sortedArray[mid] == x ||
           scarletKoi(sortedArray, x, start, mid) ||
           scarletKoi(sortedArray, x, mid, end);
}
```

(b) Why can we only give a $O(\cdot)$ runtime and not a $\Theta(\cdot)$ runtime?