CS 61B Asymptotic Analysis Spring 2020

1 Best and Worst Case

For the following functions, provide asymptotic bounds for the best case and worst case runtimes in $\Theta(\cdot)$ notation.

(a) Give the best and worst case runtimes in terms of M and N. Assume that `slam()` $\in \Theta(1)$ and returns a boolean.

```
public void comeon(int M, int N) {
    int j = 0;
    for (int i = 0; i < N; i += 1) {
        for (; j < M; j += 1) {
            if (slam(i, j))
                break;
        }
    }

    for (int k = 0; k < 1000 * N; k += 1) {
        System.out.println("space jam");
    }
}
```

(b) *Extra:* Give the best case and worst case runtimes for `find` in terms of N, where N is the length of the input array `arr`.

```
public static boolean find(int tgt, int[] arr) {
    int N = arr.length;
    return find(tgt, arr, 0, N);
}
private static boolean find(int tgt, int[] arr, int lo, int hi) {
    if (lo == hi || lo + 1 == hi) {
        return arr[lo] == tgt;
    }
    int mid = (lo + hi) / 2;
    for (int i = 0; i < mid; i += 1) {
        System.out.println(arr[i]);
    }
    return arr[mid] == tgt || find(tgt, arr, lo, mid)
                           || find(tgt, arr, mid, hi);
}
```

2 Best and Worst Case with Recursion

For the following recursive functions, provide asymptotic bounds for the best case and worst case runtimes in $\Theta(\cdot)$ notation.

(a) Give the runtime in terms of N.

```
public void andslam(int N) {
    if (N > 0) {
        for (int i = 0; i < N; i += 1) {
            for (int j = 1; j < 1024; j *= 2) {
                System.out.println(i + j);
            }
        }
        andslam(N / 2);
    }
}
```

(b) Give the runtime for `andwelcome(arr, 0, N)` in terms of N, where N is the length of the input array `arr`. `Math.random()` returns a double with a value from the range [0,1).

```
public static void andwelcome(int[] arr, int low, int high) {
    System.out.print("[ ");
    for (int i = low; i < high; i += 1) {
         System.out.print("loyal  ");
    }
    System.out.println("]");
    if (high - low > 1) {
        double coin = Math.random();
        if (coin > 0.5) {
            andwelcome(arr, low, low + (high - low) / 2);
        } else {
            andwelcome(arr, low, low + (high - low) / 2);
            andwelcome(arr, low + (high - low) / 2, high);
        }
    }
}
```

(c) Give the runtime in terms of N.

```java
public int tothe(int N) {
    if (N <= 1) {
        return N;
    }
    return tothe(N - 1) + tothe(N - 1) + tothe(N - 1);
}
```

(d) *Extra:* Give the runtime in terms of N.

```java
public static void spacejam(int N) {
    if (N == 1) {
        return;
    }
    for (int i = 0; i < N; i += 1) {
        spacejam(N-1);
    }
}
```

3 Hey you watchu gon do?

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 (i.e. N is very large).

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

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

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

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

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

Why do we need to assume that N is large?