CS 61B Spring 2026

Asymptotics II

Exam-Level Discussion 07: February 23, 2026

1 Algorithm Analysis

(a) Say we have a function `findMax` that iterates through an unsorted int array one time and returns the maximum element found in that array. Give the tightest lower and upper bounds ($\Omega(\cdot)$ and $O(\cdot)$) of `findMax` in terms of `N`, the length of the array. Is it possible to define a $\Theta(\cdot)$ bound for `findMax` in the general case?

(b) Give the worst case and best case runtime in terms of M and N. Assume `ping` runs in $\Theta(1)$ and returns an `int`.

```
for (int i = N; i > 0; i--) {
    for (int j = 0; j <= M; j++) {
        if (ping(i, j) > 64) { break; }
    }
}
```

Worst case runtime: Θ (_____)	Best case runtime: Θ (_____)

(c) Below we have a function that returns `true` if every `int` has a duplicate in the array, and `false` if there is any unique int in the array. Assume `sort(array)` is in $\Theta(N \log N)$ and returns `array` sorted.

```
public static boolean noUniques(int[] array) {
    array = sort(array);
    int N = array.length;
    for (int i = 0; i < N; i += 1) {
        boolean hasDuplicate = false;
        for (int j = 0; j < N; j += 1) {
            if (i != j && array[i] == array[j]) {
                hasDuplicate = true;
            }
        }
        if (!hasDuplicate) return false;
    }
    return true;
}
```

Give the worst case and best case runtime where $N =$ `array.length`.

Worst case runtime: Θ (_____)	Best case runtime: Θ (_____)

2 The Re-Cursed Swamp

Sometimes, it isn't possible to give a theta bound for an entire function. In these cases, it's best to analyze the best and worst-case inputs and evaluate the runtime on those to produce a "tightest" upper and lower bound.

(a) Consider the function below…

```
public static int curse(int N) {
    if (N % 2 = 0 || N <= 0) {
        return 0;
    } else {
      for (int i = 0; i < N; i++) {
        System.out.println("You have been cursed!");
      }
      return curse(N - 2);
    }
}
```

What type of input will result in the best-case runtime? What type of input will result in the worst-case runtime?

Give a tight Ω and O bound that correspond to the lower and upper bounds on this function's runtime. That is, don't just say sometihng like $O(2^N)$ which is technically true, but useless.

Feel free to use the tree-drawing technique from questions 1 and 2!

Lower bound: Ω (_____)	Upper bound: O (_____)

(b) Give the tightest runtime bound(s) for the function below. We can assume the `System.arraycopy` method takes $\Theta(N)$ time, where N is the number of elements copied. The official signature is `System.arrayCopy(Object sourceArr, int srcPos, Object dest, int destPos, int length)`. Here, `srcPos` and `destPos` are the starting points in the source and destination arrays to start copying and pasting in, respectively, and `length` is the number of elements copied.

```
public static void silly(int[] arr) {
    if (arr.length <= 1) {
      return;
    }

    int newLen = arr.length / 2;
    int[] firstHalf = new int[newLen];
    int[] secondHalf = new int[newLen];

    System.arraycopy(arr, 0, firstHalf, 0, newLen);
    System.arraycopy(arr, newLen, secondHalf, 0, newLen);

    silly(firstHalf);
    silly(secondHalf);
}
```

(c) Given that **exponentialWork** runs in $\Theta(3^N)$ time with respect to input N, give the tightest runtime bound(s) for **yellowWood**.

Hint: This one is hard! Drawing trees will be of utmost importance. If you suspect that the runtime cannot be theta-bounded, drawing one for the best case and one for the worst case can be a good idea.

```
public void yellowWood(int N) {
    if (Math.random() > 0.9) {
      twoPathsDiverge(N, 2);
    } else {
      twoPathsDiverge(N, 1);
    }
}

private void twoPathsDiverge(int N, int j) {
  if (N <= 1) {
    return;
  }
  exponentialWork(N);
  for (int i = 0; i < 3; i++) {
    twoPathsDiverge(N - j, j);
  }
}
```

3 Asymptotics is Fun!

(a) Using the function **g** defined below, what is the runtime of the following function calls? Write each answer in terms of **N**. Feel free to draw out the recursion tree if it helps.

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

g(N, 1): Θ (____)	g(N, 2): Θ (____)

(b) Suppose we change line 6 to **g(N - 1, x)** and change the stopping condition in the for loop to **i <= f(x)** where **f** returns a random number between 1 and **x**, inclusive. For the following function calls, find the tightest Ω and big O bounds. Feel free to draw out the recursion tree if it helps.

```
public static void g(int N, int x) {
    if (N == 0) {
        return;
    }
    for (int i = 1; i <= f(x); i++) {
        g(N - 1, x);
    }
}
```

g(N, 2): Ω (____)	g(N, 2): O (____)
g(N, N): Ω (____)	g(N, N): O (____)