More Algorithmic Analysis

Discussion 08
Announcement

- Congratulations on surviving Engima!
- Weekly Survey due Tuesday 03/08
- Homework 5 due Tuesday 03/08
- Lab 7 due Friday 03/11
Review
Best Case vs. Worst Case

**Best Case**: Restrict examined situation to only the best case (independent of input size)

**Worst Case**: Restrict examined situation to only the worst case (independent of input size)

Best case, worst case, and average case can ALL be bounded by Theta, O, or Omega.
Worksheet
1A Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void removeIndex(int[] arr, int i) {
    // Assume i > 0
    int N = arr.length;
    for (int j = i; j < N; j += 1) {
        arr[j - 1] = arr[j];
    }
}
```
1A  Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void removeIndex(int[] arr, int i) {
    // i is independent of N
    // Assume i > 0
    int N = arr.length;
    for (int j = i; j < N; j += 1) {
        arr[j - 1] = arr[j];
    }
}
```
1A Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void removeIndex(int[] arr, int i) {
    // Assume i > 0
    int N = arr.length;
    for (int j = i; j < N; j += 1) {
        // If i is equal to N, then it’ll run once
        arr[j - 1] = arr[j];
    }
}
```
1A Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void removeIndex(int[] arr, int i) {
    // Assume i > 0
    int N = arr.length;
    for (int j = i; j < N; j += 1) { // Best Case - Θ(1)
        arr[j - 1] = arr[j];
    }
}
```
1A Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void removeIndex(int[] arr, int i) {
    // Assume i > 0
    int N = arr.length;
    for (int j = i; j < N; j += 1) { // If i is equal to 1, then it’ll run N-1 times
        arr[j - 1] = arr[j];
    }
}
```
1A Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void removeIndex(int[] arr, int i) {
    // Assume i > 0
    int N = arr.length;
    for (int j = i; j < N; j += 1) { // Worst Case - \( \Theta(N) \)
        arr[j - 1] = arr[j];
    }
}
```
1B Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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");
    }
}
```
1B Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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)) // For the best case, assume this is always true
                break;
        }
    }
    for (int k = 0; k < 1000 * N; k += 1) {
        System.out.println("space jam");
    }
}
```
1B Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public void comeon(int M, int N) {
    int j = 0;
    for (int i = 0; i < N; i += 1) {
        for (; j < M; j += 1) { // If we always break, this runs in $\Theta(1)$
            if (slam(i, j))
                break;
        }
    }
    for (int k = 0; k < 1000 * N; k += 1) {
        System.out.println("space jam");
    }
}
```
1B Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public void comeon(int M, int N) {
    int j = 0;
    for (int i = 0; i < N; i += 1) { // N loops * Θ(1) = Θ(N)
        for (; j < M; j += 1) {
            if (slam(i, j))
                break;
        }
    }
    for (int k = 0; k < 1000 * N; k += 1) {
        System.out.println("space jam");
    }
}
```
1B Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

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

    for (int k = 0; k < 1000 * N; k += 1) { // This always takes Θ(N)
        System.out.println("space jam");
    }
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public void comeon(int M, int N) { // Best Case - Θ(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");
    }
}
```
1B Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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)) { // For worst case, assume this is never true
                break;
            }
        }
    }
    for (int k = 0; k < 1000 * N; k += 1) {
        System.out.println("space jam");
    }
}
```
1B Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public void comeon(int M, int N) {
    int j = 0;
    for (int i = 0; i < N; i += 1) {
        for (; j < M; j += 1) { // This loop runs M times in TOTAL
            if (slam(i, j))
                break;
        }
    }
    for (int k = 0; k < 1000 * N; k += 1) {
        System.out.println("space jam");
    }
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public void comeon(int M, int N) {
    int j = 0;
    for (int i = 0; i < N; i += 1) { // N outer loops + M inner loops = Θ(N + M)
        for (; j < M; j += 1) {
            if (slam(i, j))
                break;
        }
    }
    for (int k = 0; k < 1000 * N; k += 1) {
        System.out.println("space jam");
    }
}
```
1B Best and Worst Case with Iteration

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public void comeon(int M, int N) { // Worst Case - \( \Theta(N + M) \)
    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");
    }
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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);
    }
}
```
2A Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public void andslam(int N) {
    if (N > 0) {
        for (int i = 0; i < N; i += 1) {
            for (int j = 1; j < 1024; j *= 2) { // This inner loop is independent of N
                System.out.println(i + j);
            }
        }
        andSlam(N/2);
    }
}
```
2A Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public void andSlam(int N) {
    if (N > 0) {
        for (int i = 0; i < N; i += 1) { // This whole thing is therefore \( \Theta(N) \)
            for (int j = 1; j < 1024; j *= 2) {
                System.out.println(i + j);
            }
        }
        andSlam(N/2);
    }
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public void andSlam(int N) {
    if (N > 0) {
        for (int i = 0; i < N; i += 1) { // Θ(N)
            for (int j = 1; j < 1024; j *= 2) {
                System.out.println(i + j);
            }
        }
        andSlam(N/2); // Each recursion does half as much work
    }
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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);
    }
    // N + N/2 + N/4 + ... + 1 = 2N = Θ(N) <- Zeno’s Paradox
}
```
2A Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
class Example {
    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);
        }
    }
}
```

// There's no condition that's independent of N, so best and worst case are Θ(N)
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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 + (high - low) / 2, high);
            andwelcome(arr, low + (high - low) / 2, high);
        }
    }
}
```
Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void andwelcome(int[] arr, int low, int high) {
    System.out.print("[ ");
    for (int i = low; i < high; i += 1) { // This runs in $\Theta(high-low)$
        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);
        }
    }
}
```
2B Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void andwelcome(int[] arr, int low, int high) {
    System.out.print("[ ");
    for (int i = low; i < high; i += 1) { // Θ(high - low)
        System.out.print("loyal ");
    }
    System.out.println("]");
    if (high - low > 1) {
        double coin = Math.random();
        if (coin > 0.5) { // Best case is we always trigger this
            andwelcome(arr, low, low + (high - low) / 2);
        } else {
            andwelcome(arr, low + (high - low) / 2, high);
        }
    }
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void andwelcome(int[] arr, int low, int high) {
    System.out.print("[ ");
    for (int i = low; i < high; i += 1) { // Θ(high - low)
        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); // Do half the work
        } else {
            andwelcome(arr, low + (high - low) / 2, high);
            andwelcome(arr, low + (high - low) / 2, high);
        }
    }
}
```
2B Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
class Welcome {
    public static void andwelcome(int[] arr, int low, int high) {
        System.out.print("[
        for (int i = low; i < high; i += 1) { // Θ(high - low)
            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);
            }
        } // Best Case: N + N/2 + N/4 + ... + 1 = Θ(N)
    }
}
```
2B Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void andwelcome(int[] arr, int low, int high) {
    System.out.print("[ ");
    for (int i = low; i < high; i += 1) { // Θ(high - low)
        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 { // Worst case we always trigger this
        andwelcome(arr, low, low + (high - low) / 2);
    }
} else { // Best Case: Θ(N)
    andwelcome(arr, low + (high - low) / 2, high);
}
```
**2B Best and Worst with Recursion**

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void aandwecome(int[] arr, int low, int high) {
    System.out.print("[ ");
    for (int i = low; i < high; i += 1) { // \(\Theta(hi - lo)\)
        System.out.print("loyal ");
    }
    System.out.println("]");
    if (high - low > 1) {
        double coin = Math.random();
        if (coin > 0.5) {
            aandwecome(arr, low, low + (high - low) / 2);
        } else {
            aandwecome(arr, low, low + (high - low) / 2); // Each level does N work
            aandwecome(arr, low + (high - low) / 2, high); // Because each call does N/2 of previous and there are two calls
        }
    } // Best Case: \(\Theta(N)\)
}
```
2B Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void andwelcome(int[] arr, int low, int high) {
    System.out.print("[ ");
    for (int i = low; i < high; i += 1) { // \(\Theta\)\((\text{high} - \text{low})\)
        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 + (high - low) / 2, high);
        }
    }
    // Best Case: \(\Theta(N)\), Worst Case: N work * \(\log N\) levels = \(\Theta(N\log N)\)
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```
public int tothe(int N) {
    if (N <= 1) {
        return N;
    }
    return tothe(N - 1) + tothe(N - 1) + tothe(N - 1);
}
```
2C Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public int tothe(int N) {
    if (N <= 1) { // This takes constant time
        return N;
    }
    return tothe(N - 1) + tothe(N - 1) + tothe(N - 1);
}
```
2C Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

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

```
1 + 3 + 9 + ... + 3^N = Θ(3^N)
```

continued...
2D Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static void recurse(int N) {
    return helper(N, N/2);
}

private static int helper(int N, int M) {
    if (N <= 1) {
        return N;
    }
    for (int i = 1; i < M; i *= 2) {
        System.out.println(i);
    }
    return helper(N - 1, M) + helper(N - 1, M);
}
```
2D Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
1  public static void recurse(int N) {
2     return helper(N, N/2);
3  }
4  private static int helper(int N, int M) {
5      if (N <= 1) {
6          return N;
7      }
8      for (int i = 1; i < M; i *= 2) { // Each run takes logM time
9          System.out.println(i);
10      }
11      return helper(N - 1, M) + helper(N - 1, M);
12  }
```
2D Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```
public static void recurse(int N) {
    return helper(N, N/2);
}

private static int helper(int N, int M) {
    if (N <= 1) {
        return N;
    }
    for (int i = 1; i < M; i *= 2) {
        System.out.println(i);
    }
    return helper(N - 1, M) + helper(N - 1, M);
}
```

\[\log M (1 + 2 + \ldots + 2^N)\]
\[= \Theta (2^N \log M)\]
\[= \Theta (2^N \log N)\]
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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);
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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) { // This always runs in \(\Theta((\text{low} + \text{hi})/2)\)
        System.out.println(arr[i]);
    }
    return arr[mid] == tgt || find(tgt, arr, lo, mid) || find(tgt, arr, mid, hi);
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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) // Best case this is true
           || find(tgt, arr, mid, hi);
}
```
2E Best and Worst with Recursion Extra

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static boolean find(int tgt, int[] arr) {
    // Best case: $\Theta(N)$ - just one loop
    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);
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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) { // Worst case, we keep recursing until here
        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);
}
```
2E Best and Worst with Recursion Extra

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
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);
}
```
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

Putative bounds:

\[ N/2 + N + 2N + \ldots + N^2 = \Theta(N^2) \]
Best and Worst with Recursion

Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```java
public static boolean find(int tgt, int[] arr) { // Best Case: Θ(N), Worst Case: Θ(N^2)
    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);
}
```