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.

```
1  public static void removeIndex(int[] arr, int i) {
2      // Assume i > 0
3      int N = arr.length;
4      for (int j = i; j < N; j += 1) {
5          arr[j - 1] = arr[j];
6      }
7  }
```

1A Best and Worst Case with Iteration

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

```
1  public static void removeIndex(int[] arr, int i) { // i is independent of N
2      // Assume i > 0
3      int N = arr.length;
4      for (int j = i; j < N; j += 1) {
5          arr[j - 1] = arr[j];
6      }
7  }
```

1A Best and Worst Case with Iteration

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

```
1  public static void removeIndex(int[] arr, int i) {
2      // Assume i > 0
3      int N = arr.length;
4      for (int j = i; j < N; j += 1) { // If i is equal to N, then it'll run once
5          arr[j - 1] = arr[j];
6      }
7  }
```

1A Best and Worst Case with Iteration

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

```
1  public static void removeIndex(int[] arr, int i) {
2      // Assume i > 0
3      int N = arr.length;
4      for (int j = i; j < N; j += 1) { // Best Case - Θ(1)
5          arr[j - 1] = arr[j];
6      }
7  }
```

1A Best and Worst Case with Iteration

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

```
1  public static void removeIndex(int[] arr, int i) {
2      // Assume i > 0
3      int N = arr.length;
4      for (int j = i; j < N; j += 1) { // If i is equal to 1, then it'll run N-1 times
5          arr[j - 1] = arr[j];
6      }
7  }
```

1A Best and Worst Case with Iteration

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

```
1  public static void removeIndex(int[] arr, int i) {
2      // Assume i > 0
3      int N = arr.length;
4      for (int j = i; j < N; j += 1) { // Worst Case - Θ(N)
5          arr[j - 1] = arr[j];
6      }
7  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) {
2       int j = 0;
3       for (int i = 0; i < N; i += 1) {
4           for (; j < M; j += 1) {
5               if (slam(i, j))
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) {
11          System.out.println(“space jam”);
12      }
13  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) {
2       int j = 0;
3       for (int i = 0; i < N; i += 1) {
4           for (; j < M; j += 1) {
5               if (slam(i, j)) // For the best case, assume this is always true
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) {
11          System.out.println("space jam");
12      }
13  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) {
2       int j = 0;
3       for (int i = 0; i < N; i += 1) {
4           for (; j < M; j += 1) { // If we always break, this runs in Θ(1)
5               if (slam(i, j))
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) {
11          System.out.println("space jam");
12      }
13  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) {
2       int j = 0;
3       for (int i = 0; i < N; i += 1) { // N loops * Θ(1) = Θ(N)
4           for (; j < M; j += 1) {
5               if (slam(i, j))
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) {
11          System.out.println(“space jam”);
12      }
13  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) {
2       int j = 0;
3       for (int i = 0; i < N; i += 1) { // Θ(N)
4           for (; j < M; j += 1) {
5               if (slam(i, j))
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) { // This always takes Θ(N)
11          System.out.println(“space jam”);
12      }
13  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) { // Best Case - ♾(N)
2       int j = 0;
3       for (int i = 0; i < N; i += 1) {
4           for (; j < M; j += 1) {
5               if (slam(i, j))
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) {
11          System.out.println(“space jam”);
12      }
13  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) {
2       int j = 0;
3       for (int i = 0; i < N; i += 1) {
4           for (; j < M; j += 1) {
5               if (slam(i, j)) // For worst case, assume this is never true
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) {
11          System.out.println("space jam");
12      }
13  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) {
2       int j = 0;
3       for (int i = 0; i < N; i += 1) {
4           for (; j < M; j += 1) { // This loop runs M times in TOTAL
5               if (slam(i, j))
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) {
11          System.out.println(“space jam”);
12      }
13  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) {
2       int j = 0;
3       for (int i = 0; i < N; i += 1) { // N outer loops + M inner loops = Θ(N + M)
4           for (; j < M; j += 1) {
5               if (slam(i, j))
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) {
11          System.out.println(“space jam”);
12      }
13  }
```

1B Best and Worst Case with Iteration

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

```
1   public void comeon(int M, int N) { // Worst Case - ⍬(N + M)
2       int j = 0;
3       for (int i = 0; i < N; i += 1) {
4           for (; j < M; j += 1) {
5               if (slam(i, j))
6                   break;
7           }
8       }
9
10      for (int k = 0; k < 1000 * N; k += 1) {
11          System.out.println(“space jam”);
12      }
13  }
```

2A Best and Worst with Recursion

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

```
1   public void andslam(int N) {
2       if (N > 0) {
3           for (int i = 0; i < N; i += 1) {
4               for (int j = 1; j < 1024; j *= 2) {
5                   System.out.println(i + j);
6               }
7           }
8       andSlam(N/2);
9       }
10  }
```

2A Best and Worst with Recursion

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

```
1   public void andslam(int N) {
2       if (N > 0) {
3           for (int i = 0; i < N; i += 1) {
4               for (int j = 1; j < 1024; j *= 2) { // This inner loop is independent of N
5                   System.out.println(i + j);
6               }
7           }
8       andSlam(N/2);
9       }
10  }
```

2A Best and Worst with Recursion

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

```
1   public void andslam(int N) {
2       if (N > 0) {
3           for (int i = 0; i < N; i += 1) { // This whole thing is therefore Θ(N)
4               for (int j = 1; j < 1024; j *= 2) {
5                   System.out.println(i + j);
6               }
7           }
8       andSlam(N/2);
9       }
10  }
```

2A Best and Worst with Recursion

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

```
1   public void andslam(int N) {
2       if (N > 0) {
3           for (int i = 0; i < N; i += 1) { // Θ(N)
4               for (int j = 1; j < 1024; j *= 2) {
5                   System.out.println(i + j);
6               }
7           }
8       andSlam(N/2); // Each recursion does half as much work
9       }
10  }
```

2A Best and Worst with Recursion

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

```
1  public void andslam(int N) {
2      if (N > 0) {
3          for (int i = 0; i < N; i += 1) {
4              for (int j = 1; j < 1024; j *= 2) {
5                  System.out.println(i + j);
6              }
7          }
8      andSlam(N/2);
9      }
10 } // 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.

```
1   public void andslam(int N) {
2       if (N > 0) {
3           for (int i = 0; i < N; i += 1) {
4               for (int j = 1; j < 1024; j *= 2) {
5                   System.out.println(i + j);
6               }
7           }
8       andSlam(N/2);
9       }
10  }

    // There's no condition that's independent of N, so best and worst case are Θ(N)
```

2B Best and Worst with Recursion

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

```
1  public static void andwelcome(int[] arr, int low, int high) {
2      System.out.print(“[ ”)
3      for (int i = low; i < high; i += 1) {
4          System.out.print(“loyal ”);
5      }
6      System.out.println(“]”
7      if (high - low > 1) {
8          double coin = Math.random();
9          if (coin > 0.5) {
10             andwelcome(arr, low, low + (high - low) / 2);
11         } else {
12             andwelcome(arr, low, low + (high - low) / 2);
13             andwelcome(arr, low + (high - low) / 2, high);
14         }
15     }
16 }
```

2B Best and Worst with Recursion

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

```
1  public static void andwelcome(int[] arr, int low, int high) {
2      System.out.print(“[ ”)
3      for (int i = low; i < high; i += 1) { // This runs in Θ(high-low)
4          System.out.print(“loyal ”);
5      }
6      System.out.println(“]”
7      if (high - low > 1) {
8          double coin = Math.random();
9          if (coin > 0.5) {
10             andwelcome(arr, low, low + (high - low) / 2);
11         } else {
12             andwelcome(arr, low, low + (high - low) / 2);
13             andwelcome(arr, low + (high - low) / 2, high);
14         }
15     }
16 }
```

2B Best and Worst with Recursion

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

```
1  public static void andwelcome(int[] arr, int low, int high) {
2      System.out.print("[ ")
3      for (int i = low; i < high; i += 1) { // Θ(high - low)
4          System.out.print("loyal ");
5      }
6      System.out.println("]"
7      if (high - low > 1) {
8          double coin = Math.random();
9          if (coin > 0.5) { // Best case is we always trigger this
10             andwelcome(arr, low, low + (high - low) / 2);
11         } else {
12             andwelcome(arr, low, low + (high - low) / 2);
13             andwelcome(arr, low + (high - low) / 2, high);
14         }
15     }
16 }
```

2B Best and Worst with Recursion

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

```
1  public static void andwelcome(int[] arr, int low, int high) {
2      System.out.print(“[ ”)
3      for (int i = low; i < high; i += 1) { // Θ(high - low)
4          System.out.print(“loyal ”);
5      }
6      System.out.println(“]”
7      if (high - low > 1) {
8          double coin = Math.random();
9          if (coin > 0.5) {
10             andwelcome(arr, low, low + (high - low) / 2); // Do half the work
11         } else {
12             andwelcome(arr, low, low + (high - low) / 2);
13             andwelcome(arr, low + (high - low) / 2, high);
14         }
15     }
16 }
```

2B Best and Worst with Recursion

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

```
1  public static void andwelcome(int[] arr, int low, int high) {
2      System.out.print("[ ")
3      for (int i = low; i < high; i += 1) { // Θ(high - low)
4          System.out.print("loyal ");
5      }
6      System.out.println("]"
7      if (high - low > 1) {
8          double coin = Math.random();
9          if (coin > 0.5) {
10             andwelcome(arr, low, low + (high - low) / 2);
11         } else {
12             andwelcome(arr, low, low + (high - low) / 2);
13             andwelcome(arr, low + (high - low) / 2, high);
14         }
15     } // Best Case: N + N/2 + N/4 + ... + 1 = Θ(N)
16 }
```

2B Best and Worst with Recursion

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

```
1  public static void andwelcome(int[] arr, int low, int high) {
2      System.out.print("[ ")
3      for (int i = low; i < high; i += 1) { // Θ(high - low)
4          System.out.print("loyal ");
5      }
6      System.out.println("]"
7      if (high - low > 1) {
8          double coin = Math.random();
9          if (coin > 0.5) {
10             andwelcome(arr, low, low + (high - low) / 2);
11         } else { // Worst case we always trigger this
12             andwelcome(arr, low, low + (high - low) / 2);
13             andwelcome(arr, low + (high - low) / 2, high);
14         }
15     } // Best Case: Θ(N)
16 }
```

2B Best and Worst with Recursion

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

```
1  public static void andwelcome(int[] arr, int low, int high) {
2      System.out.print("[ ")
3      for (int i = low; i < high; i += 1) { // Θ(high - low)
4          System.out.print("loyal ");
5      }
6      System.out.println("]"
7      if (high - low > 1) {
8          double coin = Math.random();
9          if (coin > 0.5) {
10             andwelcome(arr, low, low + (high - low) / 2);
11         } else {
12             andwelcome(arr, low, low + (high - low) / 2); // Each level does N work
13             andwelcome(arr, low + (high - low) / 2, high);
14         } // Because each call does N/2 of previous and there are two calls
15     } // Best Case: Θ(N)
16 }
```

2B Best and Worst with Recursion

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

```
1  public static void andwelcome(int[] arr, int low, int high) {
2      System.out.print("[ ")
3      for (int i = low; i < high; i += 1) { // Θ(high - low)
4          System.out.print("loyal ");
5      }
6      System.out.println("]"
7      if (high - low > 1) {
8          double coin = Math.random();
9          if (coin > 0.5) {
10             andwelcome(arr, low, low + (high - low) / 2);
11         } else {
12             andwelcome(arr, low, low + (high - low) / 2);
13             andwelcome(arr, low + (high - low) / 2, high);
14         }
15     } // Best Case: Θ(N), Worst Case: N work * logN levels = Θ(NlogN)
16 }
```

2C Best and Worst with Recursion

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

```
1  public int tothe(int N) {
2      if (N <= 1) {
3          return N;
4      }
5      return tothe(N - 1) + tothe(N - 1) + tothe(N - 1);
6  }
```

2C Best and Worst with Recursion

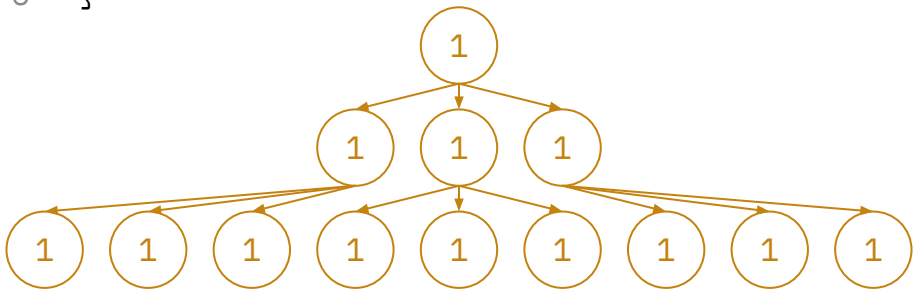
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```
1  public int tothe(int N) {
2      if (N <= 1) { // This takes constant time
3          return N;
4      }
5      return tothe(N - 1) + tothe(N - 1) + tothe(N - 1);
6  }
```

2C Best and Worst with Recursion

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

```
1  public int tothe(int N) {
2      if (N <= 1) {
3          return N;
4      }
5      return tothe(N - 1) + tothe(N - 1) + tothe(N - 1);
6  }
```



continued…

$1 + 3 + 9 + \ldots + 3^N = \Theta(3^N)$

2D Best and Worst with Recursion

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

```
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) {
9          System.out.println(i);
10     }
11     return helper(N - 1, M) + helper(N - 1, M);
12 }
```

2D Best and Worst with Recursion

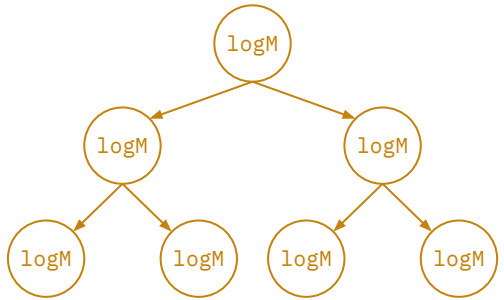
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```
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.

```
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) {
9           System.out.println(i);
10      }
11      return helper(N - 1, M) + helper(N - 1, M);
12  }
```



continued…

$$\text{logM}(1 + 2 + \ldots + 2^N) = \Theta(2^N \text{logM}) = \Theta(2^N \text{logN})$$

2E Best and Worst with Recursion *Extra*

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

```
1   public static boolean find(int tgt, int[] arr) {
2       int N = arr.length;
3       return find(tgt, arr, 0, N);
4   }
5   private static boolean find(int tgt, int[] arr, int lo, int hi) {
6       if (lo == hi || lo + 1 == hi) {
7           return arr[lo] == tgt;
8       }
9       int mid = (lo + hi) / 2;
10      for (int i = 0; i < mid; i += 1) {
11          System.out.println(arr[i]);
12      }
13      return arr[mid] == tgt || find(tgt, arr, lo, mid)
14                             || find(tgt, arr, mid, hi);
15  }
```

2E Best and Worst with Recursion *Extra*

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

```
1  public static boolean find(int tgt, int[] arr) {
2      int N = arr.length;
3      return find(tgt, arr, 0, N);
4  }
5  private static boolean find(int tgt, int[] arr, int lo, int hi) {
6      if (lo == hi || lo + 1 == hi) {
7          return arr[lo] == tgt;
8      }
9      int mid = (lo + hi) / 2;
10     for (int i = 0; i < mid; i += 1) { // This always runs in Θ((low + hi)/2)
11         System.out.println(arr[i]);
12     }
13     return arr[mid] == tgt || find(tgt, arr, lo, mid)
14                            || find(tgt, arr, mid, hi);
15 }
```

2E Best and Worst with Recursion *Extra*

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

```
1  public static boolean find(int tgt, int[] arr) {
2      int N = arr.length;
3      return find(tgt, arr, 0, N);
4  }
5  private static boolean find(int tgt, int[] arr, int lo, int hi) {
6      if (lo == hi || lo + 1 == hi) {
7          return arr[lo] == tgt;
8      }
9      int mid = (lo + hi) / 2;
10     for (int i = 0; i < mid; i += 1) {
11         System.out.println(arr[i]);
12     }
13     return arr[mid] == tgt || find(tgt, arr, lo, mid) // Best case this is true
14                            || find(tgt, arr, mid, hi);
15 }
```

2E Best and Worst with Recursion *Extra*

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

```
1  public static boolean find(int tgt, int[] arr) { // Best case: Θ(N) - just one loop
2      int N = arr.length;
3      return find(tgt, arr, 0, N);
4  }
5  private static boolean find(int tgt, int[] arr, int lo, int hi) {
6      if (lo == hi || lo + 1 == hi) {
7          return arr[lo] == tgt;
8      }
9      int mid = (lo + hi) / 2;
10     for (int i = 0; i < mid; i += 1) {
11         System.out.println(arr[i]);
12     }
13     return arr[mid] == tgt || find(tgt, arr, lo, mid)
14                            || find(tgt, arr, mid, hi);
15 }
```

2E Best and Worst with Recursion *Extra*

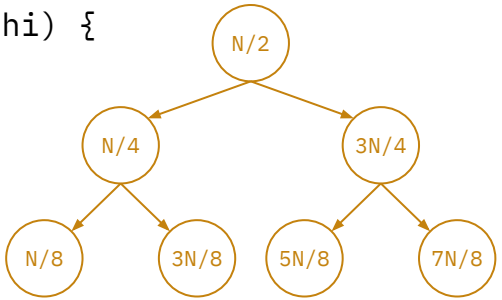
Provide asymptotic bounds for the best and worst case runtimes in theta notation.

```
1  public static boolean find(int tgt, int[] arr) {
2      int N = arr.length;
3      return find(tgt, arr, 0, N);
4  }
5  private static boolean find(int tgt, int[] arr, int lo, int hi) {
6      if (lo == hi || lo + 1 == hi) { // Worst case, we keep recursing until here
7          return arr[lo] == tgt;
8      }
9      int mid = (lo + hi) / 2;
10     for (int i = 0; i < mid; i += 1) {
11         System.out.println(arr[i]);
12     }
13     return arr[mid] == tgt || find(tgt, arr, lo, mid)
14                            || find(tgt, arr, mid, hi);
15 }
```

2E Best and Worst with Recursion *Extra*

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

```
1   public static boolean find(int tgt, int[] arr) {
2       int N = arr.length;
3       return find(tgt, arr, 0, N);
4   }
5   private static boolean find(int tgt, int[] arr, int lo, int hi) {
6       if (lo == hi || lo + 1 == hi) {
7           return arr[lo] == tgt;
8       }
9       int mid = (lo + hi) / 2;
10      for (int i = 0; i < mid; i += 1) {
11          System.out.println(arr[i]);
12      }
13      return arr[mid] == tgt || find(tgt, arr, lo, mid)
14                             || find(tgt, arr, mid, hi);
15  }
```



2E Best and Worst with Recursion *Extra*

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

```
1   public static boolean find(int tgt, int[] arr) {
2       int N = arr.length;
3       return find(tgt, arr, 0, N);
4   }
5   private static boolean find(int tgt, int[] arr, int lo, int hi) {
6       if (lo == hi || lo + 1 == hi) {
7           return arr[lo] == tgt;
8       }
9       int mid = (lo + hi) / 2;
10      for (int i = 0; i < mid; i += 1) {
11          System.out.println(arr[i]);
12      }
13      return arr[mid] == tgt || find(tgt, arr, lo, mid)
14                             || find(tgt, arr, mid, hi);
15  }
```

$N/2 + N + 2N + \ldots + N^2$
$= \Theta(N^2)$

2E Best and Worst with Recursion *Extra*

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

```
1  public static boolean find(int tgt, int[] arr) { // Best Case: Θ(N), Worst Case: Θ(N^2)
2      int N = arr.length;
3      return find(tgt, arr, 0, N);
4  }
5  private static boolean find(int tgt, int[] arr, int lo, int hi) {
6      if (lo == hi || lo + 1 == hi) {
7          return arr[lo] == tgt;
8      }
9      int mid = (lo + hi) / 2;
10     for (int i = 0; i < mid; i += 1) {
11         System.out.println(arr[i]);
12     }
13     return arr[mid] == tgt || find(tgt, arr, lo, mid)
14                            || find(tgt, arr, mid, hi);
15 }
```