(more)

algorithmic analysis

asymptotics, runtime, etc.

slides
bit.ly/abhi-disc

attendance
bit.ly/abhi-attendance

announcements

1. HW 5 due 3/8 (tomorrow)
2. Lab 8 due 3/11 (friday)
3. Weekly survey due tomorrow!

cost (review)

- **time complexity**
  - time it takes to run the program if we feed it a certain input
- **space complexity**
  - how much space does running the program take up on our computer?

asymptotics (review)

- evaluate the performance of a program using math
- ignore all constants
- only care about values with reference to the input (denoted as having size ‘N’)

bounds (review)

- **big O:** upper bound in terms of the input
  - assume conditional statements evaluate to the worst case
- **big Ω:** lower bound in terms of the input
  - assume conditional statements evaluate to the best case
- **big Θ:** the tightest bound, only exists when the upper and lower bounds converge

useful sums (review)

$1 + 2 + 3 + \ldots + N = \Theta(N^2)$ -> "arithmetic" sum

$1 + 2 + 4 + 8 + \ldots + N = \Theta(N)$ -> "geometric" sum

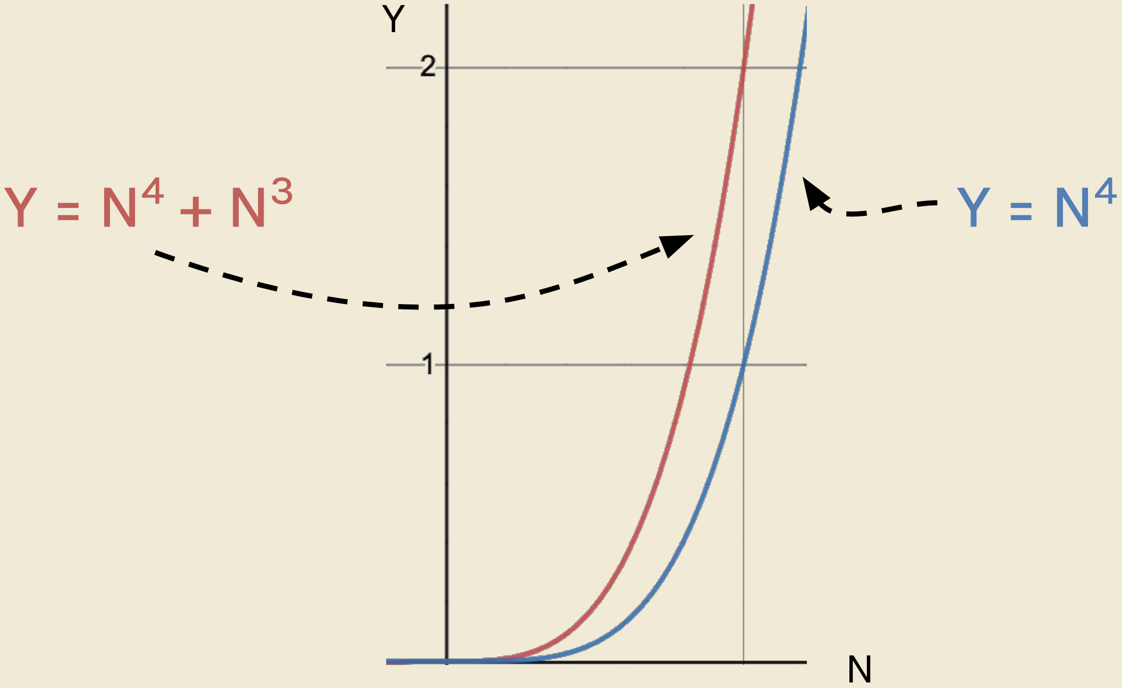
$2^0 + 2^1 + 2^2 \ldots\ldots + 2^{\log_2 N}$

$\Downarrow$

N

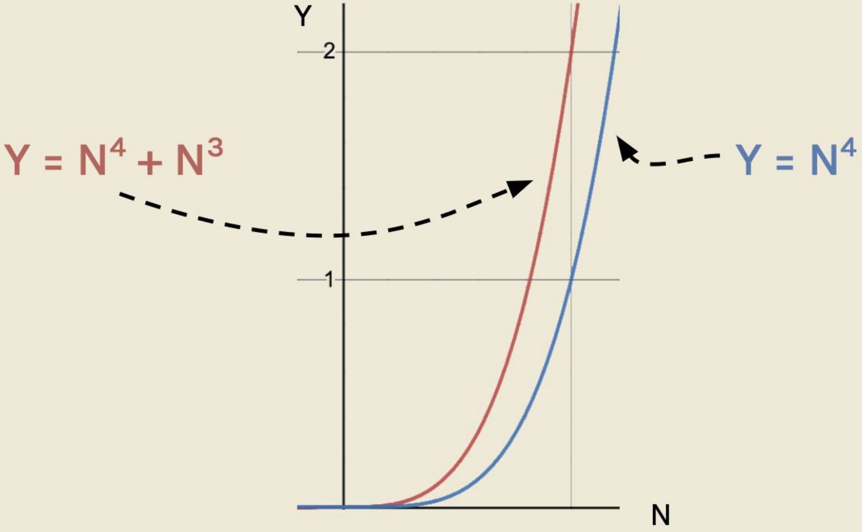
$$\Theta(N^2 + \log N) = \Theta(N^2)$$
?
$$\Theta(N^4 + N^3) = \Theta(N^4)$$
$$\Theta(2^N + N^{314159265359}) = \Theta(2^N)$$

$$\Theta(N^4 + N^3) = \Theta(N^4)?$$



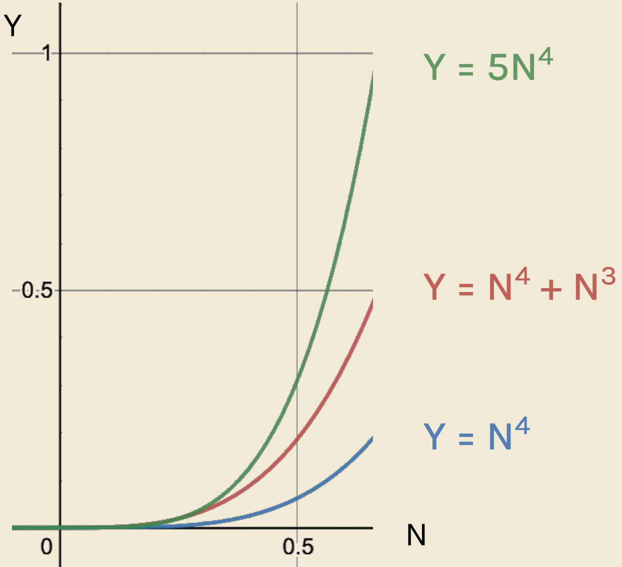
$\Theta(N^4 + N^3) = \Theta(N^4)$? how?

- seems to me like $y = N^4$ *strictly* < $y = N^4 + N^3$



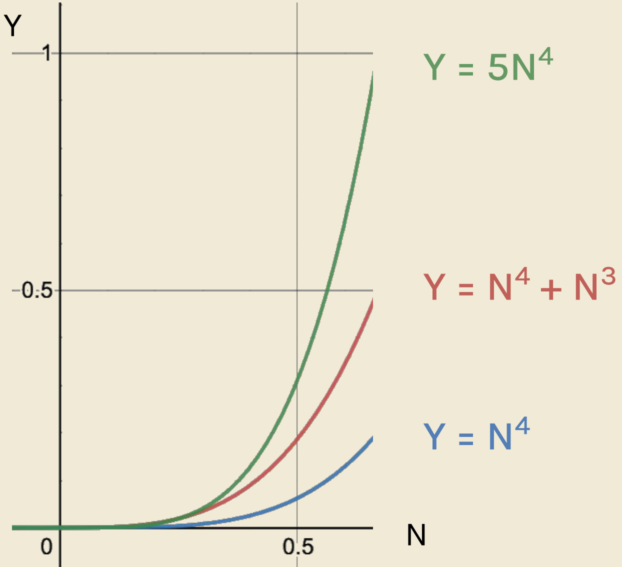
$\Theta(N^4 + N^3) = \Theta(N^4)$? how?

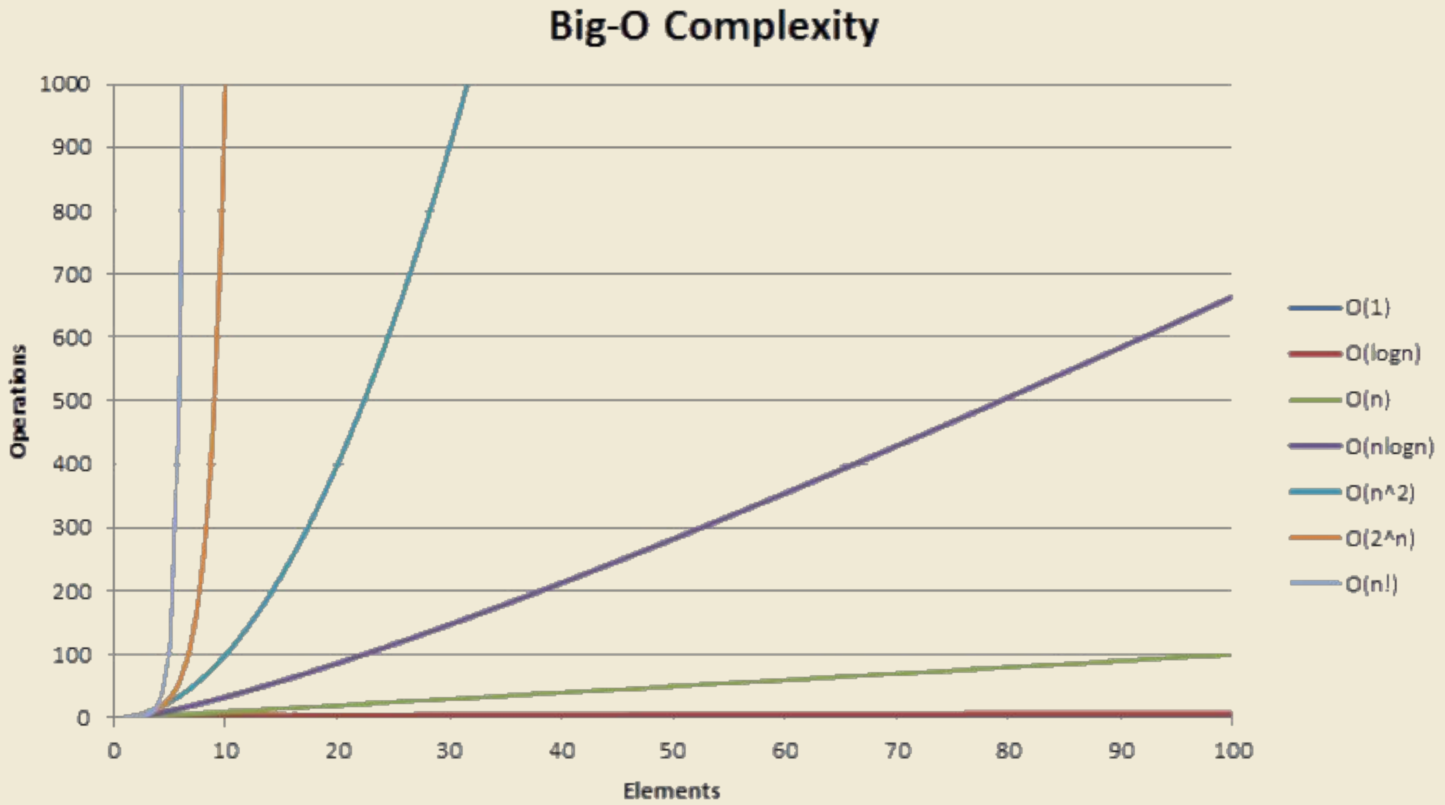
- seems to me like $y = N^4$ *strictly* < $y = N^4 + N^2$
- consider $y = 5N^4$



$\Theta(N^4 + N^3) = \Theta(N^4)$? how?

- seems to me like $y = N^4$ *strictly* $<$ $y = N^4 + N^2$
- consider $y = 5N^4$
- $N^4 < N^4 + N^2 < 5N^4$





Big-O Complexity
Operations
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
Elements
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

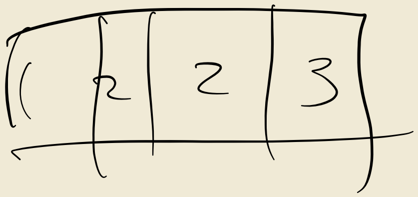
analyzing a program

- choose an operation to count
- figure out the order of growth of the operation
    - exact counting OR
    - inspection

[inspired by josh hug 2019, lec 13]

analyzing a program - dup

```java
private static boolean dup(int[] A) {
    int N = A.length;
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++) {
            if (A[i] == A[j]) {
                return true;
            }
        }
    }
    return false;
}
```



[inspired by josh hug 2019, lec 13]()

dup(int[] A) runtime

```
private static boolean dup(int[] A) {
    int N = A.length;
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++) {
            if (A[i] == A[j]) {
                return true;
            }
        }
    }
    return false;
}
```

operation to count?

inspired by josh hug 2019, lec 13

dup(int[] A) runtime

```
private static boolean dup(int[] A) {
    int N = A.length;
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++) {
            if (A[i] == A[j]) {
                return true;
            }
        }
    }
    return false;
}
```

operation to count

inspired by josh hug 2019, lec 13

dup(int[] A) runtime

```java
private static boolean dup(int[] A) {
    int N = A.length;
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++) {
            if (A[i] == A[j]) {
                return true;
            }
        }
    }
    return false;
}
```

N = 6

i \ j	0	1	2	3	4	5
0		==	==	==	==	==
1			==	==	==	==
2				==	==	==
3					==	==
4						==
5						

[inspired by josh hug 2019, lec 13]()

dup(...) exact runtime

$$C = (N - 1) + (N - 2) + \ldots + 2 + 1$$

N = 6

i \ j	0	1	2	3	4	5
0		==	==	==	==	==
1			==	==	==	==
2				==	==	==
3					==	==
4						==
5						

inspired by josh hug 2019, lec 13

dup(...) exact runtime

$$C = (N - 1) + (N - 2) + \ldots + 2 + 1$$
$$= 1 + 2 + \ldots + (N - 2) + (N - 1)$$

N = 6

i \ j	0	1	2	3	4	5
0		==	==	==	==	==
1			==	==	==	==
2				==	==	==
3					==	==
4						==
5						

[inspired by josh hug 2019, lec 13]

dup(...) exact runtime

$$C = (N - 1) + (N - 2) + \ldots + 2 + 1$$
$$= 1 + 2 + \ldots + (N - 2) + (N - 1)$$

useful sums

$1 + 2 + 3 + \ldots + N = \Theta(N^2)$ -> “arithmetic” sum

$1 + 2 + 4 + 8 + \ldots + N = \Theta(N)$ -> “geometric” sum

N = 6

i \ j	0	1	2	3	4	5
0		==	==	==	==	==
1			==	==	==	==
2				==	==	==
3					==	==
4						==
5						

inspired by josh hug 2019, lec 13

dup(...) exact runtime

$$C = (N - 1) + (N - 2) + \ldots + 2 + 1$$

$$= \left(1 + 2 + \ldots + (N - 2) + (N - 1)\right)$$

N^2

$\Theta(N^2)$

useful sums

$1 + 2 + 3 + \ldots + N = \Theta(N^2)$ -> “arithmetic” sum

$1 + 2 + 4 + 8 + \ldots + N = \Theta(N)$ -> “geometric” sum

N = 6

i \ j	0	1	2	3	4	5
0		==	==	==	==	==
1			==	==	==	==
2				==	==	==
3					==	==
4						==
5						

inspired by josh hug 2019, lec 13

dup(...) geometric

N = 6

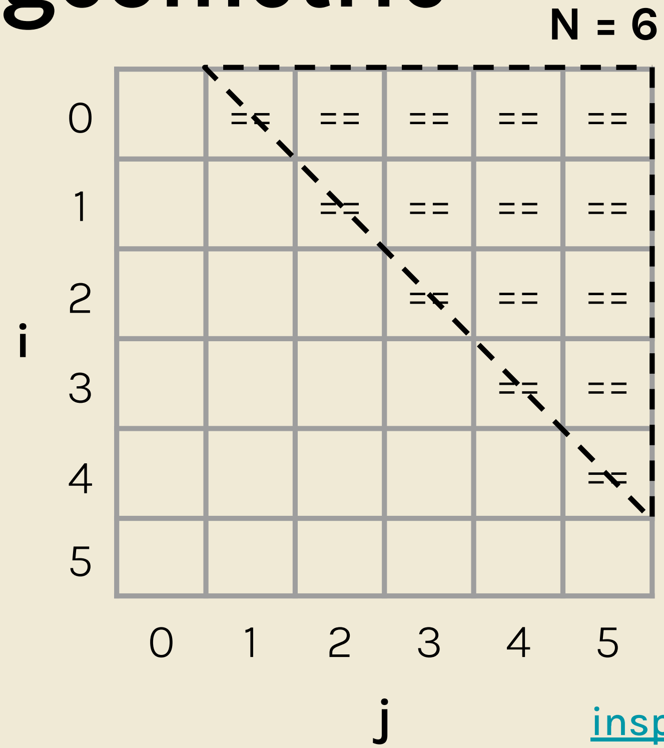
i

	0	1	2	3	4	5
0		==	==	==	==	==
1			==	==	==	==
2				==	==	==
3					==	==
4						==
5						

j

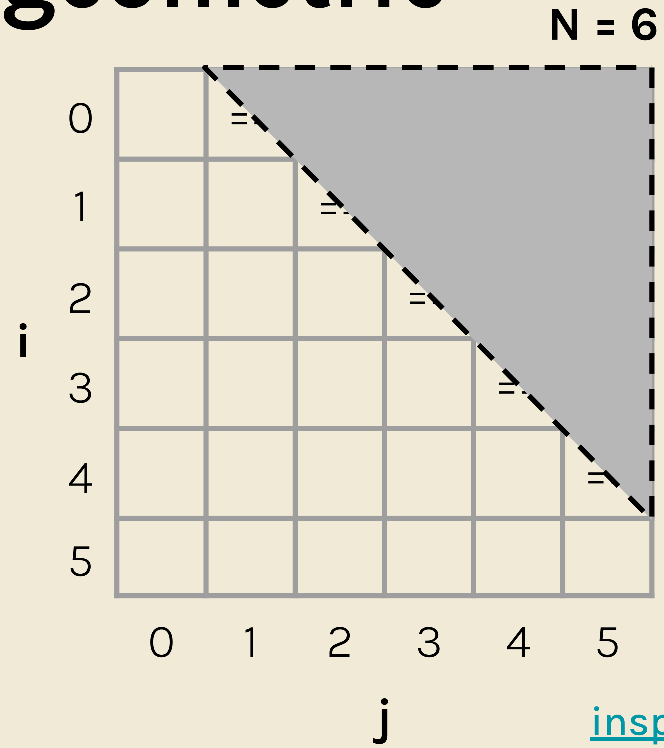
[inspired by josh hug 2019, lec 13]

dup(...) geometric



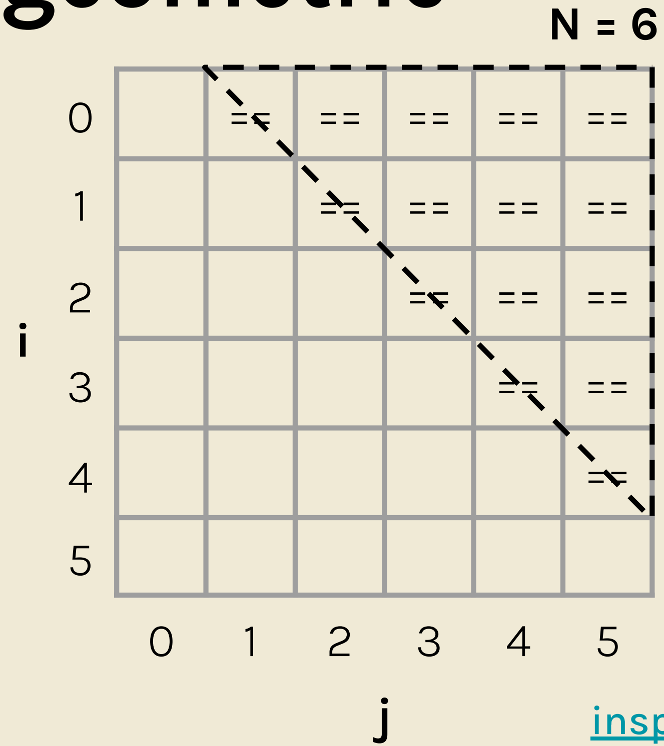
inspired by josh hug 2019, lec 13

dup(...) geometric



[inspired by josh hug 2019, lec 13]

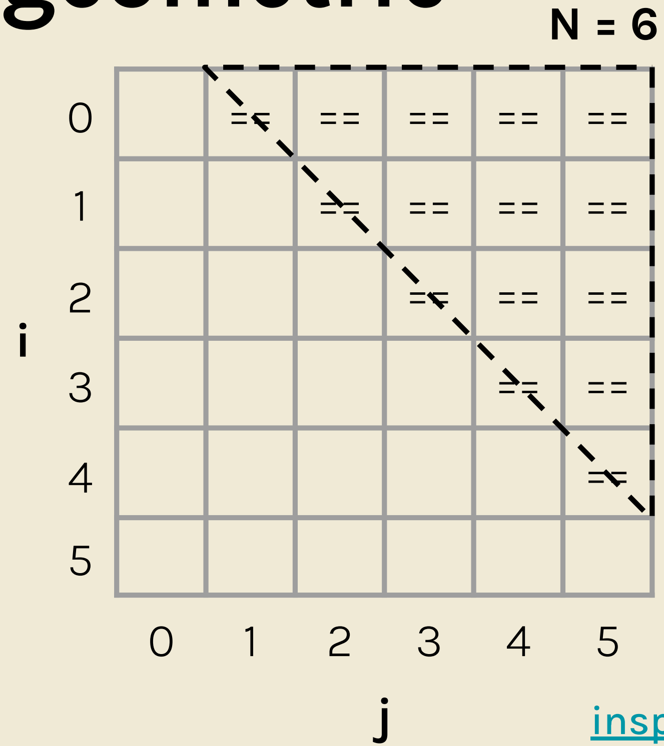
dup(...) geometric



$A = \frac{1}{2}bh$

[inspired by josh hug 2019, lec 13]

dup(...) geometric



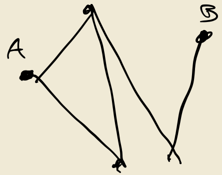
$A = \frac{1}{2}bh$

$A = \frac{1}{2}(N - 1)(N - 1)$

inspired by josh hug 2019, lec 13

dup(...) geometric

N = 6



i \ j	0	1	2	3	4	5
0		==	==	==	==	==
1			==	==	==	==
2				==	==	==
3					==	==
4						==
5						

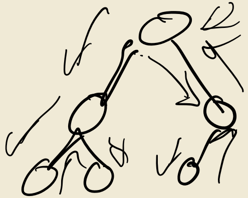
$A = \frac{1}{2}bh$

$A = \frac{1}{2}(N - 1)(N - 1)$

'==' grows on the scale of $\Theta(N^2)$

[inspired by josh hug 2019, lec 13]()

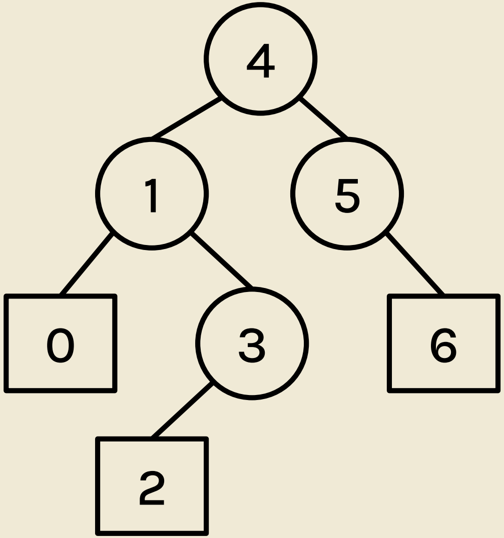
tree traversals



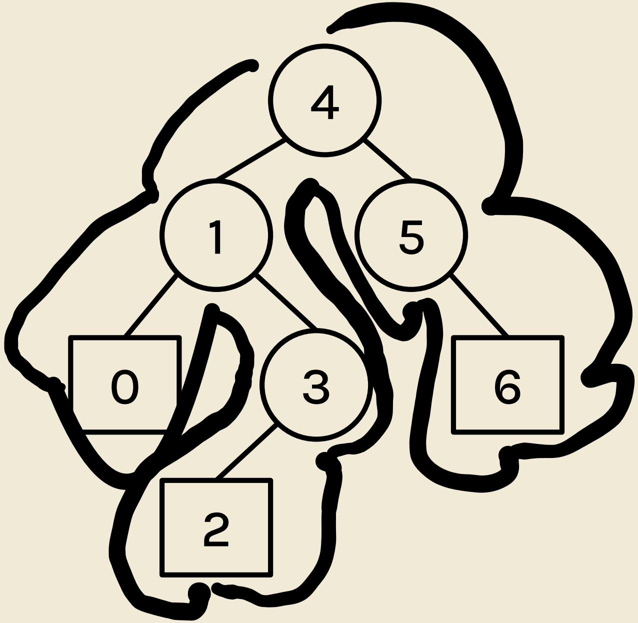
- **preorder:** visit node, then traverse children
- **inorder:** traverse left child, then node, then traverse right child
- **postorder:** traverse children, then visit node

tree traversals (preorder)

4, 1, 0, 3, 2, 5, 6

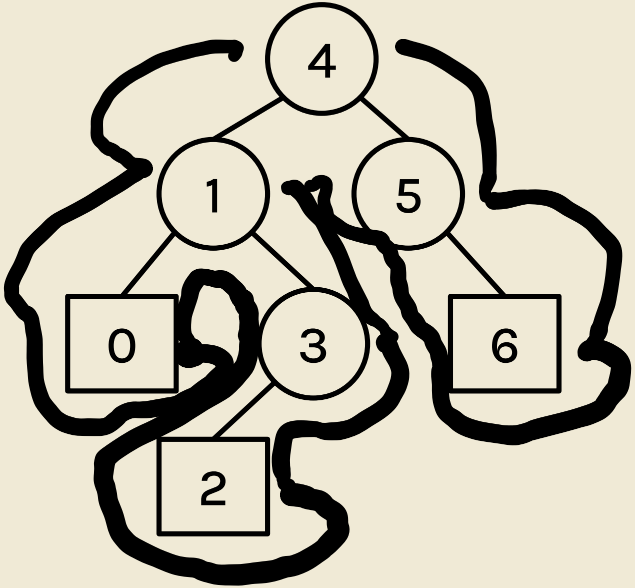


tree traversals (inorder)



0, 1, 2
3, 4, 5
6

tree traversals (postorder)



0, 2, 3, 1, 6
5, 4

worksheet

(on 61B website)

1A Best and Worst Case with Iteration

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

```
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.

```
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.

```
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.

```
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 0, then it'll run N 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.

```
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 - Θ(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.

```
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.

```
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 Θ(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.

```
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.

```
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");
    }
}
```

1B Best and Worst Case with Iteration

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

```
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.

```
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.

```
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");
    }
}
```

1B Best and Worst Case with Iteration

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

```
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.

```
public void comeon(int M, int N) { // Worst Case - Θ(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");
    }
}
```

2A Best and Worst with Recursion

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

```
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.

```
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);
    }
}
```

Best Case: $\Theta(N)$
Worst Case: $\Theta(N)$

2B Best and Worst with Recursion

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

```
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);
        }
    }
}
```

2B Best and Worst with Recursion

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

```
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);
        }
    }
}
```

Best Case: $\Theta(N)$
Worst Case: $\Theta(N\log N)$

2C Best and Worst with Recursion

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.

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

Best Case: $\Theta(N)$
Worst Case: $\Theta(3^N)$

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);
}
```

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);
}
```

Best Case: $\Theta(2^N \log N)$
Worst Case: $\Theta(2^N \log N)$

2E Best and Worst with Recursion *Extra*

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

```
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);
}
```

2E Best and Worst with Recursion *Extra*

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

```
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);
}
```

Best Case: $\Theta(N)$
Worst Case: $\Theta(N^2)$

attendance

bit.ly/abhi-attendance

feedback

bit.ly/abhi-feedback

slides: **bit.ly/abhi-disc**