Asymptotic Analysis

Big Oh Notation

- **Defn**: Let $T(n)$ be a non-negatively valued function
  $T(n)$ is in the set $O(f(n))$ if there exist two positive constants c and n_0 such that
  $T(n) \leq cf(n)$ for all $n > n_0$
- **Usage:** $T(n)$ is in $O(f(n))$ in the [best, worst, average] case
- Gives the upper bound of the growth rate
- Example: $3\mathrm{n}^2 \in O(n^2)$
  - Notice that $3\mathrm{n}^2 \in O(n^3)$ but we seek the tightest (smallest) upper bound

Big Oh - example

- Linear search
  - Worst case: $T(n) = c_0 n + c_1$
    - $T(n) \in O(n)$
    - $c = c_0 + c_1$
    - $n_0 = 1$
  - Derivation:
    $T(n) = c_0 n + c_1 \leq c_0 n + c_1 n = (c_0 + c_1) n$

Big Oh - example

- Linear search
  - Average case: $T(n) = \frac{c_0(n+1)}{2} + c_1$
    - $T(n) \in O(n)$
    - $c = c_0 + c_1$
    - $n_0 = 1$
  - Derivation:

$$T(n) = \frac{c_0(n+1)}{2} + c_1 = \frac{c_0}{2}n + \frac{c_0}{2} + c_1 \leq \frac{c_0}{2}n + \frac{c_0}{2}n + c_1 n = (c_0 + c_1)n$$

Big Oh - example

- Linear search
  - Best case: $T(n) = c_1$
    - $T(n) \in O(1)$
    - $c = c_1$
    - $n_0 = 0$

Big Oh - example

- $T(n) = c_1 n^2 + c_2 n$
  - $c_1 n^2 + c_2 n \leq c_1 n^2 + c_2 n^2 = (c_1 + c_2) n^2$ for all $n > 1$
  - Therefore $T(n) \in O(n^2)$
  - $n_0 = 1$
  - $c = c_1 + c_2$

Big Omega

- **Defn**: Let $T(n)$ be a non-negatively valued function
  $T(n)$ is in the set $\Omega(f(n))$ if there exist two positive constants c and n_0 such that
  $T(n) \geq cf(n)$ for all $n > n_0$
- **Usage:** $T(n)$ is in $\Omega(f(n))$ in the [best, worst, average] case
- Gives the lower bound for the growth rate
- Example: $3\mathrm{n}^2 \in \Omega(n^2)$
  - Notice that $3\mathrm{n}^2 \in \Omega(n)$ but we seek the tightest (greatest) lower bound

Big Omega – example

- $T(n) = c_1 n^2 + c_2 n$
- $c_1 n^2 + c_2 n \geq c_1 n^2$ for all $n > 1$
- Therefore $T(n) = \Omega(n^2)$
  - $n_0 = 1,\ c = c_1$

Theta notation

- When $T(n) \in O(f(n))$ and $T(n) \in O(f(n))$ we say $T(n) = \Theta(f(n))$
- Simplifying rules for polynomials
  1. if $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$ then $f(n) \in O(h(n))$
  2. if $f(n) \in O(kg(n))$ for any constant $k > 0$ then $f(n) \in O\big(g(n)\big)$
     no constant factors
  3. if $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$ then $(f_1 + g_1)(n)$ is in $\in$ $\mathrm{O}\big(\max\big(g_1(n), g_2(n)\big)\big)$
     drop lower order terms
  4. if $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$ then $f_1(n)g_1(n)$ is in $\in$ $\mathrm{O}\big(g_1(n)g_2(n)\big)$
     loops

Analyzing algorithms - examples

- Example: assignment: `a = b;`
  - $\Theta(1)$
- Example 2:

```
int sum = 0;
for (int i = 1; i <= n; i++)
     sum += n;
```

  - $\Theta(n)$

Analyzing algorithms - examples

- Example 3:

```
int sum = 0;
for (int j = 1; j <= n; j++)
     for (int i = 1; i <= j; i++)
          sum++;
for (int k = 1; k <= n; k++)
     A[k] = i;
```

$$\sum_{i=1}^{n} i = \Theta(n^2)$$

$$\Theta(n)$$

  - $\Theta(n^2)$

Analyzing algorithms - examples

- Example 3:

```
int sum1 = 0;
for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j++)
            sum1++;
```

Nested loop, each running n times
$\Theta(n^2)$

```
int sum2 = 0;
for (int i = 1; i <= n; i++)
      for (int j = 1; j <= i; j++)
            sum2++;
```

$$\sum_{i=1}^{n} i = \Theta(n^2)$$

- $T(n) = \Theta(n^2)$

Analyzing algorithms - examples

- Example 3:

```
int sum1 = 0;
for (int k = 1; k <= n; k *= 2)
      for (int j = 1; j <= n; j++)
            sum1++;
```

$$\sum_{k=1}^{\log n} n = \Theta(n \log n)$$

```
int sum2 = 0;
for (int k = 1; k <= n; k *= 2)
      for (int j = 1; j <= k; j++)
            sum2++;
```

$$\sum_{k=0}^{\log n - 1} 2^k = \Theta(n)$$

- $T(n) = \Theta(n \log n)$

Binary Search

```
int binary(int k, int array[], int left, int right) {
    int l = left - 1, r = right + 1;
    while (l + 1 != r) {
        int mid = (l + r) / 2;
        if (k < array[mid])
            r = mid;
        else if (k > array[mid])
            l = mid;
        else
            return mid;
    }
    return -1;
}
```

Determining Θ

- while-loops
  - Same technique as for for-loops
- if-statement
  - Take the most expensive branch
- switch-statement
  - Take most expensive case
- Method/function call
  - Complexity of the method/function

Example: Determining Θ for while-loop

```java
public static void main(String[] args) {
    int n = 16, i = 0, count = 0;
    while (i++ < n) {
        int j = 1;

        while(j < n){
            count++;
            j = j * 2;
        }
    }
    System.out.println(count);
}
```

Outer while repeats n times. The values of i are (0, 1, …, n-1)

For each iteration of the outer while-loop, the inner while-loop repeats log n times. The values of j are $(2^0, 2^1, 2^2, \ldots, 2^{\lfloor \log n \rfloor})$

The complexity of the program is therefore $\Theta(n \times \log n) = \Theta(n \log n)$

This variable determines the running time, so we let it be n, the input size.

Examp… etermining Θ if statements

```
if (value < 6) {
    System.out.println(lookupTable[value]);
} else {
    int res = 1;
    for (int i = 1; i <= value; i++)
        res *= i;
    System.out.println(res);
}
```

The if part runs in constant time, since we only inspect one array position and print the value. $\Theta(1)$

The else part runs in $\Theta(n)$

The most expensive branch of the if-else statement runs in $\Theta(n)$ time, therefore, we conclude that the complexity of the whole if-else statement is $\Theta(n)$

Example: Determining Θ for switch statement

```
switch (option) {
    case 1:
        int sum = 0;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                sum += a[i][j];
        System.out.println("sum = " + sum);
        break;
    case 2:
        int diaSum = 0;
        for (int i = 0; i < n; i++)
            diaSum += a[i][i];
        System.out.println("Diagonal sum = " + diaSum);
        break;
    case 3:
        int rnd1 = (int)(Math.random()*n);
        int rnd2 = (int)(Math.random()*n);
        System.out.println("Random pick = a[" + rnd1 + "][" + rnd2 + "] = " + a[rnd1][rnd2] );
        break;
    default:
    System.out.println("wow");
}
```

$\Theta(n^2)$

$\Theta(n)$

$\Theta(1)$, assuming Math.random() runs in constant time

$\Theta(1)$

The most expensive branch of the switch statement runs in $\Theta(n^2)$ time, therefore, we conclude that the complexity of the whole switch statement is $\Theta(n^2)$

Example: Determining Θ for method call

```
Arrays.sort(intList);

for (int i = 0; i < intList.length; i++) {// linear search
    if (intList[i] == target) {
        System.out.println("Found at position " + i);
        break;
}
```

The implantation of this method runs in $\Theta(n \log n)$ – according to Java Docs

Linear search runs in $\Theta(n)$

Total running time $\mathrm{T(n)} = \Theta(n \log n + n) = \Theta(n \log n)$

Analyzing **problems**

- Analyze algorithms that solve the problem
- Upper bound
  - Upper bound of the best known algorithm
- Lower bound
  - Lower bound for every possible (not only the known ones) algorithm
    - Usually requires some interesting proofs

Space Bounds

- Time bounds
  - For algorithms
- Space bounds
  - For data structures
- Space/Time tradeoff principle
  - Save time by using more space and vice versa
  - Example: look-up table for factorial function

Example: space/time tradeoff

- A Java implementation of a $\Theta(n)$ factorial function
  - Save space at the expense of time

```
static long factorialC(int n) {
     assert n >= 0 && n <= 20 : "input out of range";
     long fact = 1;

     for (int i = 1; i <= n; i++) {
          fact *= i;
     }

     return fact;
}
```

Example: space/time tradeoff

- A Java implementation of a $\Theta(1)$ factorial function
  - Save time at the expense of space

```java
// precomputed factorial values stored in array
static long fact[]= {1L, 1L, 2L, 6L, 24L, 120L, 720L, 5040L,
         40320L, 362880L, 3628800L, 39916800L, 479001600L,
         6227020800L, 87178291200L, 1307674368000L,
         20922789888000L, 355687428096000L, 6402373705728000L,
         121645100408832000L, 2432902008176640000L};

static long factorialM(int n) {
    assert n >= 0 && n <= 20 : "input out of range";
    return fact[n]; // read a value from an array and return
}
```