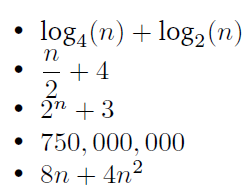
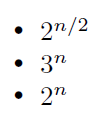
Section 02: Asymptotic Analysis Problems

1. **Comparing growth rates**
   1. Simplify each of the following functions to a big-O bound in terms of *n*. Then order them from fastest to slowest in terms of asymptotic growth. (By “fastest”, we mean which function increases the most rapidly as *n* increases.)
   2. **Order each of these more esoteric functions from fastest to slowest in terms of asymptotic growth. (By “fastest”, we mean which function increases the most rapidly as *n* increases.) Also state a simplified big-O bound for each.

# Modeling code

For each of the following code blocks, give a summation that represents the worst-case runtime in terms of *n*.

(a)

**int** x = 0;

**for** (**int** i = 0; i < n; i++) {

**for** (**int** j = 0; j < i; j++) { x++;

}

}

(b)

**int** x = 0;

**for** (**int** i = n; i >= 1; i /= 2) {

x += i;

}

# Finding bounds

For each of the following code blocks, construct a mathematical function modeling the worst-case runtime of the code in terms of *n*. Then, give a big-O bound of your model.

* 1. **int** x = 0;

**for** (**int** i = 0; i < n; i++) {

**for** (**int** j = 0; j < n \* n / 3; j++) { x += j;

}

}

* 1. **int** x = 0;

**for** (**int** i = n; i >= 0; i -= 1) {

**if** (i % 3 == 0) {

## break;

} **else** {

x += n;

}

}

* 1. **int** x = 0;

**for** (**int** i = 0; i < n; i++) {

**if** (i % 5 == 0) {

**for** (**int** j = 0; j < n; j++) {

**if** (i == j) {

x += i \* j;

}

}

}

}

* 1. **int** x = 0;

**for** (**int** i = 0; i < n; i++) {

**if** (n < 100000) {

**for** (**int** j = 0; j < n; j++) { x += 1;

}

} **else** {

x += 1;

}

}

# Case Analysis

For each of the following methods:

1. What are the possible code models (runtime functions) in terms of n, the size of the data structure?
2. What’s the best case / worst case code model? Or are they the same and there’s only one case? Talk about what specific inputs / state of the data structure are required to trigger the best / worst case code model you’re discussing.
   1. ArrayList

data; // a field for the array that stores all the values size; // a field to keep track of the number of valid values

insert(index, value) // inserts the given value at the given index

**public void** insert (index, value) {

**for** (**int** i = size; i > index; i–) { data[i] = data[i - 1];

}

data[index] = value; size++;

}

size() // returns the number of valid elements in the list

**public int** size () {

**return** size;

}

* 1. LinkedDictionary

ListNode front; // just imagine that this is just like ListNode

// with a .next field, but also has a .key and

// .value field

get(key) // returns the value associated with the given key

**public** V get(key) {

ListNode current = front;

**while** (current != **null**) {

**if** (current.key == key) {

**return** current.value;

} **else** {

current = current.next;

}

}

## return null;

}

* 1. BubbleSort(array)

**public void** bubbleSort(**int** arr[]){

**boolean** swapped;

**for** (**int** i = 0; i < arr.length; i++) { swapped = **false**;

**for** (**int** j = 0; j < arr.length - 1; j++){

**if** (arr[j] > arr[j + 1]){

// swap arr[j] and arr[j+1]

**int** temp = arr[j]; arr[j] = arr[j + 1]; arr[j + 1] = temp; swapped = **true**;

}

}

// IF no two elements were

// swapped by inner loop, then finish method

**if** (swapped == **false**){

## return;

}

}

}