

Lecture 7 - January 28

Asymptotic Analysis of Algorithms, Arrays and Linked Lists

*Deriving Upper Bounds from Code
Inserting into an Array
Sorting Orders*

Announcements/Reminders

- Assignment 1 solution released
- *splitArrayHarder*: an extended version released
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- Contact Information of TAs on common eClass site

Determining the Asymptotic Upper Bound (1)

approx.

vs. Counting # B_3

```
1 int maxOf (int x, int y) {  
2     int max = x; /  
3     if (y > x) { /  
4         max = y; /  
5     }  
6     return max; /  
7 }
```

$$O\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{6}\right) = O(4 \cdot n^0)$$

$$= O(n^0)$$
$$O(1)$$

Determining the Asymptotic Upper Bound (2)

```
1 int findMax (int[] a, int n) {  
2     currentMax = a[0]; // n iterations  
3     for (int i = 1; i < n; ) {  
4         if (a[i] > currentMax)  
5             currentMax = a[i]; }  
6         i++; }  
7     return currentMax; } 1
```

$$O(\underbrace{1}_{\text{2}} + \underbrace{n \cdot 1}_{\text{3}} + \underbrace{1}_{\text{4~avg}}) = O(n)$$

$$[\underline{a}, \underline{b}]$$

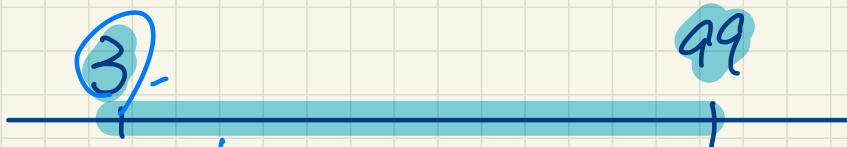


$$\underline{b-a+1}.$$

integers in the
closed interval

$$[0, n-1] \rightarrow (n-1) - 0 + 1 = \textcircled{n}$$

$$[\underline{3}, \underline{99}]$$



$$99 - 3$$

$$99 - 3 + 1$$

Determining the Asymptotic Upper Bound (3.1)

```
1 boolean containsDuplicate (int[] a, int n) {  
2     for (int i = 0; i < n; ) {  
3         for (int j = 0; j < n; ) {  
4             if (i != j && a[i] == a[j]) {  
5                 return true; }  
6             j++; }  
7         i++; } O(1)  
8     return false; } O(1)
```

* P1 gets executed
for every combination
of (i, j)

* P2 gets exec.
for every value of i

Patterns of Loop Counter

outer loop
min $i = 0$
 $i \in [0, n-1]$
 n iterations

i	j	0	1	2	\dots	$n-1$	n its.
0	0	1	2	\dots	$n-1$	n its.	$n \cdot n$ calls in the matrix (i, j)
1	0	1	2	\dots	$n-1$	n its	# i values
2	0	1	2	\dots	$n-1$	n its	# j values
\vdots	$i+1$ to n # i values						
$n-1$	0	1	2	\dots	$n-1$	n its	n combinations

$$O(n^2 \cdot 1 + n \cdot 1 + 1)$$

$$= O(n^2 + n + 1)$$

$$= O(n^2)$$

Determining the Asymptotic Upper Bound (3.2)

```
1 boolean containsDuplicate (int[] a, int n) {  
2     for (int i = 0; i < X; 10,000 {  
3         for (int j = 0; j < n; ) {  
4             if (i != j && a[i] == a[j]) {  
5                 return true; }  
6             j++; }  
7             i++; }  
8     return false; }
```

$$\begin{aligned} O(10,000 \cdot n) \\ = O(n) \end{aligned}$$

```
1 boolean containsDuplicate (int[] a, int n) {  
2     for (int i = 0; i < X; 1M {  
3         for (int j = 0; j < X; 1M {  
4             if (i != j && a[i] == a[j]) {  
5                 return true; }  
6             j++; }  
7             i++; }  
8     return false; }
```

$$\begin{aligned} O(1M \cdot 1M) \\ = O(\cancel{X} \cdot n^0) \\ = O(1) \end{aligned}$$

a.length

b.length

```
1 boolean containsDuplicate (int[] a, int n) {  
2     for (int i = 0; i < n; ) {  
3         for (int j = 0; j < n; m) {  
4             if (i != j && a[i] == a[j]) {  
5                 return true; }  
6             j++; }  
7         i++; }  
8     return false; }
```

$$O(n \cdot m) = \cancel{O}(n^2)$$

Determining the Asymptotic Upper Bound (4)

```
1 int sumMaxAndCrossProducts (int[] a, int n) {  
2     int max = a[0]; 1  
3     for (int i = 1; i < n; i++) { 1  
4         if (a[i] > max) { max = a[i]; } 1  
5     }  
6     int sum = max; 1  
7     for (int j = 0; j < n; j++) { 1  
8         for (int k = 0; k < n; k++) { 1  
9             sum += a[j] * a[k]; } } 1  
10    return sum; } 1
```

$$O(1 + n + 1 + n^2 + 1)$$

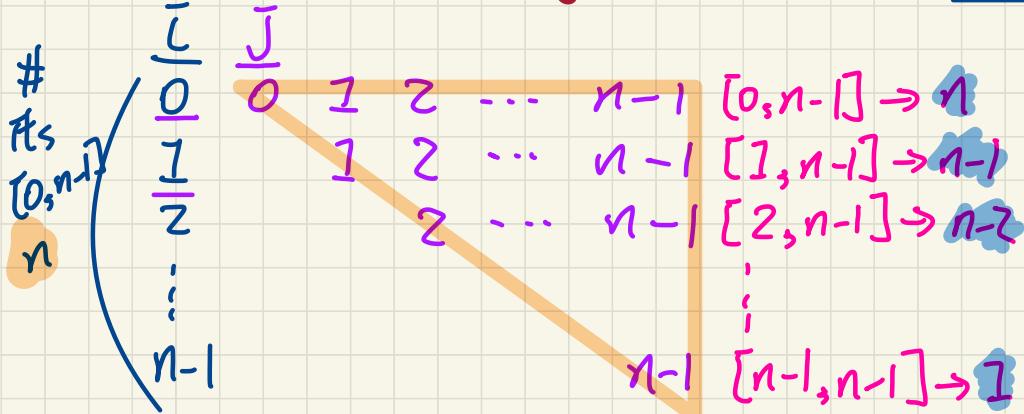
$$= O(n^2 + n + 3) = O(n^2)$$

Determining the Asymptotic Upper Bound (5)

```
1 int triangularSum (int[] a, int n) {  
2     int sum = 0; //  
3     for (int i = 0; i < n; i++) {  
4         for (int j = i; j < n; j++) {  
5             sum += a[j]; } }  
6     return sum; } | ① exec for each (i,j)
```

Pattern of (\bar{i}, \bar{j})

$$\# (\bar{i}, \bar{j}) = \frac{n + (n-1) + (n-2) + \dots + 1}{2} = \frac{(n+1) \cdot n}{2} = O(n^2)$$



$$O(\underbrace{1}_{2} + \underbrace{n^2 \cdot \frac{1}{2}}_{25} + \underbrace{1}_{26}) = O(n^2 + \sum) = O(n^2)$$

Asymptotic Upper Bound: Arithmetic Sequence/Progression

$$\frac{\bar{c} + 0 \cdot c}{\downarrow \text{start term}} + (\bar{c} + c) + (\bar{c} + 2 \cdot c) + \dots + (\bar{c} + (n-1) \cdot c)$$

terms : n

Common difference

Sum :

$$\frac{[\bar{c} + (\bar{c} + (n-1) \cdot c)] \cdot n}{2} = \frac{c \cdot n^2 + (2\bar{c} - c) \cdot n}{2}$$

$\hookrightarrow O(n^2)$

object creation : O(1)

```
String[] insertAt(String[] a, int n, String e, int i)
1 String[] result = new String[n + 1];    1
  for(int j = 0; j <= i - 1; j++) { result[j] = a[j]; }
1 result[i] = e; [0, i-1] → i iterations → worst case: n iterations
→ for(int j = i + 1; j <= n; j++) { result[j] = a[j-1]; }
1 return result; [i+1, n] → n-(i+1)+1 → n-i iterations
```

Example:

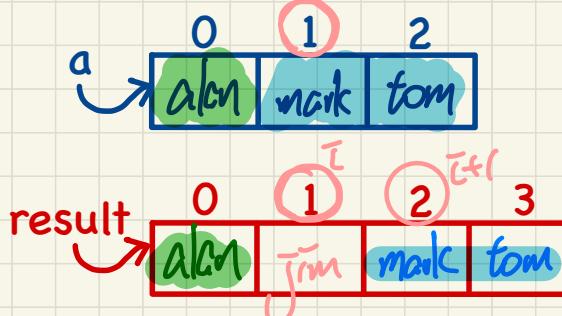
The diagram illustrates the state of an array during an insertion operation. The array is initially defined as `{alan, mark, tom}`. A variable `a` points to the first element, labeled `0`. The elements are shown in a box: `alan`, `mark`, and `tom`. A `for` loop is indicated above, with annotations `for 2nd for loop` and `a` pointing to the current element. The value `jim` is being inserted at index `0`. The resulting array is shown below, with `jim` at index `0` and the original elements shifted to the right: `jim`, `alan`, `mark`, and `tom`. The index `0` is highlighted in yellow.

Exercise: insertAt({alan, mark, tom}, 3, jim, 3)

Diagram illustrating the insertion step of the insertion sort algorithm. A stack frame shows variables n , e , i , and $\text{result}[j]$. The array a is shown with indices i and j . An annotation shows the insertion of 'e' at index i into the array, with indices from 0 to $n-1$ labeled. The code snippet shows the assignment $\text{result}[j] = a[i]$, followed by annotations for worst case iterations and the final iteration count.

Example: $O(\underline{l} + n \cdot l + l + (n - \underline{l}) \cdot l + l)$
= $O(n)$

`insertAt({alan, mark, tom}, 3, jim, 1)`



worst cast for 1st for bop