

EECS3101 (Section Z) Winter 2026
Tutorial 02
2D Arrays & JUnit,
Deriving Asymptotic Upper Bounds,
Dynamic Arrays (Part 1)

Chen-Wei Wang

Release Date: Thursday, January 15

No Submission Required: Complete for Learning & Test Prep

Contents

1	Two-Dimensional Arrays	2
1.1	Exercise: Finding the Row with the Maximum Sum	2
1.2	Exercise: Is a 2D Array a Rectangle?	2
2	Asymptotic Upper Bounds	3
2.1	Exercise: Proving Asymptotic Upper Bound	3
2.2	More Exercises: Proving Asymptotic Upper Bounds	3
2.3	Exercise: Deriving Asymptotic Upper Bounds of Algorithms (1)	3
2.4	Exercise: Deriving Asymptotic Upper Bounds of Algorithms (2)	3
2.5	Exercise: Deriving Asymptotic Upper Bounds of Algorithms (3)	3
3	Analyzing the Constant Increments Strategy	4

1 Two-Dimensional Arrays

1.1 Exercise: Finding the Row with the Maximum Sum

Given a 2D array **a** of integers, find out the row (i.e., a one-dimensional array) which has the *maximum* sum.

Assume. **a** is not empty, and such a row (with the *maximum* sum) is unique.

1.2 Exercise: Is a 2D Array a Rectangle?

Given a 2D array **a** of integers, determine if it is a rectangle.

Assume. **a** is not empty.

Hint. How does a rectangle look like?

2 Asymptotic Upper Bounds

2.1 Exercise: Proving Asymptotic Upper Bound

Prove: The function $f(n) = 5n^4 - 3n^3 + 2n^2 - 4n + 1$ is $O(n^4)$.

2.2 More Exercises: Proving Asymptotic Upper Bounds

Determine and prove the asymptotic upper bound of:

- $5n^2 + 3n \cdot \log n + 2n + 5$
- $20n^3 + 10n \cdot \log n + 5$
- $3 \cdot \log n + 2$
- 2^{n+2}
- $2n + 100 \cdot \log n$

2.3 Exercise: Deriving Asymptotic Upper Bounds of Algorithms (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 ++; }  
8     return false; }
```

2.4 Exercise: Deriving Asymptotic Upper Bounds of Algorithms (2)

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

2.5 Exercise: Deriving Asymptotic Upper Bounds of Algorithms (3)

```
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; }
```

3 Analyzing the Constant Increments Strategy

Consider the *constant increments* strategy for dynamic arrays:

```
1 public class ArrayStack<E> implements Stack<E> {
2     private int I;
3     private int C;
4     private int capacity;
5     private E[] data;
6     public ArrayStack() {
7         I = 1000; /* arbitrary initial size */
8         C = 500; /* arbitrary fixed increment */
9         capacity = I;
10        data = (E[]) new Object[capacity];
11        t = -1;
12    }
13    public void push(E e) {
14        if (size() == capacity) {
15            /* resizing by a fixed constant */
16            E[] temp = (E[]) new Object[capacity + C];
17            for(int i = 0; i < capacity; i++) {
18                temp[i] = data[i];
19            }
20            data = temp;
21            capacity = capacity + C
22        }
23        t++;
24        data[t] = e;
25    }
26 }
```

(Task 1) Derive the tightest asymptotic upper bound on the average/amortized running time of the **push** operation.