EECS3101 (Section E) Summer 2026
Tutorial 2
2D Arrays & JUnit,
Deriving Asymptotic Upper Bounds,
Dynamic Arrays (Part 1)

Chen-Wei Wang

Release Date: Thursday, May 14

No Submission Required: Complete for Learning & Test Prep

Contents

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 Asymptoic 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 asymptoic upper bound of:

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

2.3 Exercise: Deriving Asymptotic Uppder Bounds of Algorithms (1)

```
boolean containsDuplicate (int[] a, int n) {
  for (int i = 0; i < n; ) {
    for (int j = 0; j < n; ) {
      if (i != j && a[i] == a[j]) {
        return true; }
      j ++; }
    i ++; }
  return false; }
```

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

```
int sumMaxAndCrossProducts (int[] a, int n) {
  int max = a[0];
  for(int i = 1; i < n; i ++) {
    if (a[i] > max) { max = a[i]; }
  }
  int sum = max;
  for (int j = 0; j < n; j ++) {
    for (int k = 0; k < n; k ++) {
      sum += a[j] * a[k]; } }
  return sum; }
```

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

```
int triangularSum (int[] a, int n) {
  int sum = 0;
  for (int i = 0; i < n; i ++) {
    for (int j = i; j < n; j ++) {
      sum += a[j]; } }
  return sum; }
```

3 Analyzing the Constant Increments Strategy

Consider the *constant increments* strategy for dynamic arrays:

```
public class ArrayStack<E> implements Stack<E> {
  private int I;
  private int C;
  private int capacity;
  private E[] data;
  public ArrayStack() {
    I = 1000; /* arbitrary initial size */
    C = 500; /* arbitrary fixed increment */
    capacity = I;
    data = (E[]) new Object[capacity];
    t = -1;
  }
  public void push(E e) {
    if (size() == capacity) {
      /* resizing by a fixed constant */
      E[] temp = (E[]) new Object[capacity + C];
      for(int i = 0; i < capacity; i ++) {
        temp[i] = data[i];
      }
      data = temp;
      capacity = capacity + C
    }
    t++;
    data[t] = e;
  }
}
```

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