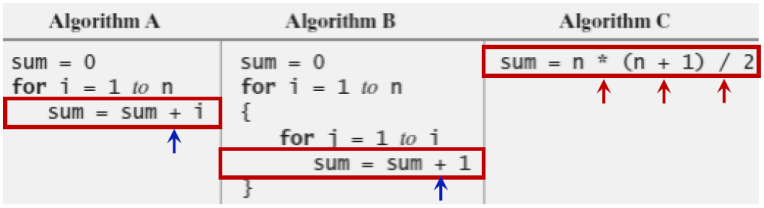
PKG 08: THE EFFICIENTCY OF ALGORITHMS and COUNTING OPERATIONS



<= In the table: The operations which are boxed in rectangles are BASIC operations.

An algorithm's basic operations are the most significant contributors to its total time requirement.

	Algorithm A	Algorithm B	Algorithm C
Additions	n	$n(n+1)/2$	1
Multiplications			1
Divisions			1
Total basic operations	n	$(n^2+n)/2$	**3**

```
Alg A ==> 3 ms <===
Alg A Result:
125000250000

Alg B ==> 93866 ms <===
Alg B Result:
125000250000

Alg C ==> 0 ms <===
Alg C Result:
125000250000
```

```
public class Sum_1_to_n_3Algorithms {

    static long startTime;
    static long endTime;

    public static void main(String[] args) {

        // Computing the sum of the consecutive intigers from 1 to n:
        long n = 500000;

        // Algorithm A
        startTime = System.currentTimeMillis();
        long sum = 0;
        for (long i = 1; i <= n; i++) {
            sum = sum + i;
        }
        endTime = System.currentTimeMillis();

        System.out.println("\nAlg A ==> " + (endTime - startTime) + " ms <===");
        System.out.println("Alg A Result: " + sum);

        // Algorithm B
        startTime = System.currentTimeMillis();
        sum = 0;
        for (long i = 1; i <= n; i++) {
            for (long j = 1; j <= i; j++) {
                sum = sum + 1;
            }
        }
        endTime = System.currentTimeMillis();

        System.out.println("\nAlg B ==> " + (endTime - startTime) + " ms <===");
        System.out.println("Alg B Result: " + sum);

        // Algorithm C
        startTime = System.currentTimeMillis();
        sum = n * (n + 1) / 2;
        endTime = System.currentTimeMillis();

        System.out.println("\nAlg C ==> " + (endTime - startTime) + " ms <===");
        System.out.println("Alg C Result: " + sum);
    }
}
```

```
COUNTING -BASIC- OPERATIONS
public class AlgorithmsABC {
    public static void main(String[] args) {
        int n = 5;

        // Algorithm A
        System.out.println("Algorithm A: ________________________________________");
        long sumA = 0;
        for (long i = 1; i <= n; i++) {
            System.out.printf("%3s %1d %2s", "i  : ", i," | sumA = sumA + i = ");
            System.out.printf("%2d %s %1d", sumA, " + ", i);
            sumA = sumA + i;
            System.out.printf("%2s %2d %n", " | sumA:", sumA);
        }

        // Algorithm B
        System.out.println("");
        System.out.println("Algorithm B: ________________________________________");
        long sumB = 0;
        for (long i = 1; i <= n; i++) {
            for (long j = 1; j <= i; j++) {
                System.out.print("i,j: " + i + "," + j + " | sumB = sumB + 1 = ");
                System.out.printf("%2d %s %1d", sumB, " + ", 1);
                sumB = sumB + 1;
                System.out.printf("%3s %2d %n", " | sumB:", sumB);
            }
        }

        // Algorithm C
        System.out.println("");
        System.out.println("Algorithm C: ________________________________________");
        long sumC = n * (n + 1) / 2;
        System.out.print("sumC = n * (n + 1) / 2");
        System.out.printf("%23s %2d", "| sumC:",  sumC);
    }
}

Algorithm A: ____________________________________
i  :  1  | sumA = sumA + i =  0  +  1 | sumA:  1
i  :  2  | sumA = sumA + i =  1  +  2 | sumA:  3
i  :  3  | sumA = sumA + i =  3  +  3 | sumA:  6
i  :  4  | sumA = sumA + i =  6  +  4 | sumA: 10
i  :  5  | sumA = sumA + i = 10  +  5 | sumA: 15

Algorithm B: ____________________________________
i,j: 1,1 | sumB = sumB + 1 =  0  +  1 | sumB:  1
i,j: 2,1 | sumB = sumB + 1 =  1  +  1 | sumB:  2
i,j: 2,2 | sumB = sumB + 1 =  2  +  1 | sumB:  3
i,j: 3,1 | sumB = sumB + 1 =  3  +  1 | sumB:  4
i,j: 3,2 | sumB = sumB + 1 =  4  +  1 | sumB:  5
i,j: 3,3 | sumB = sumB + 1 =  5  +  1 | sumB:  6
i,j: 4,1 | sumB = sumB + 1 =  6  +  1 | sumB:  7
i,j: 4,2 | sumB = sumB + 1 =  7  +  1 | sumB:  8
i,j: 4,3 | sumB = sumB + 1 =  8  +  1 | sumB:  9
i,j: 4,4 | sumB = sumB + 1 =  9  +  1 | sumB: 10
i,j: 5,1 | sumB = sumB + 1 = 10  +  1 | sumB: 11
i,j: 5,2 | sumB = sumB + 1 = 11  +  1 | sumB: 12
i,j: 5,3 | sumB = sumB + 1 = 12  +  1 | sumB: 13
i,j: 5,4 | sumB = sumB + 1 = 13  +  1 | sumB: 14
i,j: 5,5 | sumB = sumB + 1 = 14  +  1 | sumB: 15

Algorithm C: ____________________________________
sumC = n * (n + 1) / 2                | sumC: 15
```

Algorithm A	Algorithm B	Algorithm C
n	$n\,(n+1)\,/\,2$	1 1 1
n	$(n^2+n)\,/\,2$	3

ALGORITHM EFFICIENCY, COUNTING -BASIC- OPERATIONS

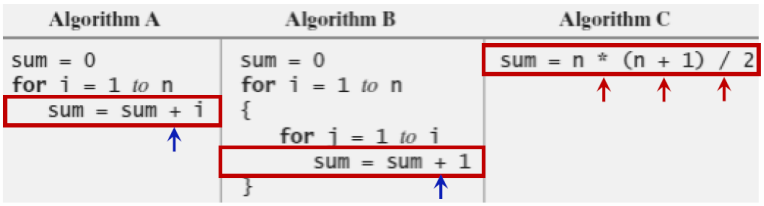
FRANK CARRANO:

- Analysis of Algorithm is the process of measuring the complexity of algorithm.

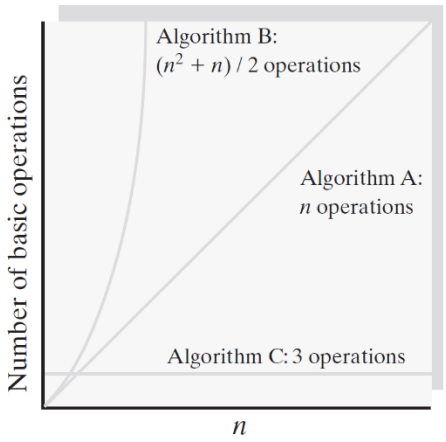
- Algorithm analysis is all about understanding growth rate. That is as the amount of data gets bigger, how much more resource will the algorithm require?

- Typical (simplified) growth rate functions are simple because the effect of an inefficient algorithm is not noticeable when the problem is small => Focus on large problems or we only care about large values of n when comparing algorithms. We consider only the dominant term in each growth rate function.

- ***Please experiment with file BigO-GrowthRates.xlsx on the File Manager.***



	Algorithm A	Algorithm B	Algorithm C
Additions	n	$n\,(n+1)\,/\,2$	1
Multiplications			1
Divisions			1
Total basic operations	n	$(n^2+n)\,/\,2$	**3**



- Basic operation of algorithms A and B is addition.
- Basic operations of algorithm C are addition, multiplication, and division.

- A requires n additions of i to sum in the body of the loop.
  -> A requires time that increases linearly with n. (Considering basic ops only.)
  -> A requires time directly proportional to n.
  -> A's growth-rate function is n.
  -> A is O(n).

- B requires time directly proportional to $(n^2 + n)/2$. Focus on large problems -> Only consider the dominant term in each growth-rate function -> Large value of n only.
  -> B is $O(n^2)$

- C requires time that is constant and independent of the value of n, always 3 ops.
  -> C is O(1)

Big Oh Notation

- O(1), constant : The same amount of time regardless of the amount of data.

- O(log n), logarithmic: The data being used is decreased by roughly 50% each time through the algorithm.
- O(n), linear : Time to complete will grow in direct proportion to the amount of data.
- O (n log n), n log n : Every element must be visited at least once. Each pass reduces the possible lists in half.
- $O(n^2)$, quadratic : Time to complete is proportional to the square of the amount of data.

- ***Please utilize the materials on the File Manager.***

COUNTING -ALL- OPERATIONS

EXAMPLE

Count ALL operations before and in the for-loop including arithmetic operations, assignment, *return, compare...*

```
int i, n = 10, sum = 0;
for (i = 0; i <= 2n; i++) {
      sum += i + 3;
}
```

Before the for-loop:

```
      n = 10, sum = 0:
                          2 ops, 2 assignment operations
```

Inside the body of the for-loop, EXE:

```
      sum += i + 3;
                          3 ops, EXE, body execution sum = sum + i + 3 requires
                                      1 assignment operation and 2 addition operations.
```

For-loop, the control structure and the body:

```
      1. i = 0            1 ops: 1 assignment operation, ONCE
      2. i <= 2*n         2 ops: 1 comparison and 1 multiplication, every time
      3. EXE              3 ops: for-loop body statements, when step 2 is satisfied
      4. i = i+1          2 ops: 1 assignment and 1 addition, when step 3 happens
```

for (i = 0; i <= 2n; EXE i++)				
i	Assignment	Comparison	Body Execution	Incrementation
0	1	1*2	1*3	1*2
1	0	1*2	1*3	1*2
2	0	1*2	1*3	1*2
3	0	1*2	1*3	1*2
4	0	1*2	1*3	1*2
5	0	1*2	1*3	1*2
...	...	...	...	...
Subtotal at n	1	(n+1)*2	(n+1)*3	(n+1)*2
Subtotal at n+1	1	(n+2)*2	(n+2)*3	(n+2)*2
...	...	...	...	...
Subtotal at 2n-1	1	2n*2	2n*3	2n*2
Subtotal at 2n	1	(2n+1)*2	(2n+1)*3	(2n+1)*2
Subtotal at 2n+1	1	(2n+2)*2	(2n+1)*3	(2n+1)*2

Total Operations:

```
      - Before for-loop:   2 ops

      - For-loop:                    1  +    (2n+2)*2   +  (2n+1)*3   +   (2n+1)*2
                 =                   1  +     4n + 4    +   6n + 3    +    4n + 2
                 =  14n + 10 ops

      - TOTAL:      14n + 12 ops
```

```
COUNTING -ALL- OPERATIONS

/**
 * Count the number of operations (ops) in the control structure of the for-loop.
 *
 * * * ORDER:
 * * 1. i = 0   happens ONCE
 * * 2. i < n   happens every time
 * * 3. EXE     for-loop body statements execute only when step 2 is satisfied
 * * 4. i++     happens only when step 3 happens
 *
 * for (int i = 0;        i < n;                  i++ )
 * i    assignment     comparison      assignment & addition
 * 0        1              1              1           1
 * 1        0              1              1           1
 * 2        0              1              1           1
 * 3        0              1              1           1
 * ...
 * n        0              1              0           0
 *
 * Total: 1 + (n + 1) + 2n = 3n + 2
 */

// EXAMPLE
//  Count all operations in the for-loops including:
//  arithmetic operations, assignment, return, compare...
//  int i, n = 10, sum = 0;
//  for (i = 0; i <= 2n; i++) {
//       sum += i + 3;
//  }

public class NumberOfOperationsInForLoopPractice {

    public static void main(String[] args) {

        int i, n = 10, sum = 0; // A, 2 assignments, 2 ops, ONCE

        // B: When for-loop starts, i = 0. 1 assignment, 1 ops, ONCE ONLY
        // C: i <= 2*n. 1 multiplication and 1 comparison, 2 ops, ONCE EACH LOOP
        // D: sum = sum + i + 3. 1 assignment and 2 additions, 3 ops, ONCE EACH LOOP
        // E: i = i + 1. 1 addition and 1 assignment, 2 ops, ONCE EACH LOOP
        int opTotal = 1;        // B. Counter not included.

        //
        for (i = 0; i <= 2*n; i++, opTotal += 2) { // E

            sum += i + 3;       // D
            opTotal += 2 + 3;   // C and D. Counter not included.
            System.out.println("i: " + i + "\tTotal OPS: " + opTotal);

        }

        // F: When i = 2n +1, +1 multiplication and +1 comparison.
        opTotal += 2 + 2;       // A and F. Counter not included.
        System.out.println("i: " + i + "\tTotal OPS: " + opTotal);

        // 1 + (2n + 2)*2 + (2n + 1)*2 = 8n + 7
        //                   (2n + 1)*3 = 6n + 3
        //                           +2 = 2
        //                              = 14n + 12
        // if n == 10, sum = 152
    }
}
```

```
i: 0  Total OPS: 6
i: 1  Total OPS: 13
i: 2  Total OPS: 20
i: 3  Total OPS: 27
i: 4  Total OPS: 34
i: 5  Total OPS: 41
i: 6  Total OPS: 48
i: 7  Total OPS: 55
i: 8  Total OPS: 62
i: 9  Total OPS: 69
i: 10 Total OPS: 76
i: 11 Total OPS: 83
i: 12 Total OPS: 90
i: 13 Total OPS: 97
i: 14 Total OPS: 104
i: 15 Total OPS: 111
i: 16 Total OPS: 118
i: 17 Total OPS: 125
i: 18 Total OPS: 132
i: 19 Total OPS: 139
i: 20 Total OPS: 146
i: 21 Total OPS: 152
```

COUNTING -ALL- OPERATIONS, PRACTICE

```
int i, n = 10, sum = 0;
for (i = 0; i <= n; i++) {
      sum += 1;
}
```

```
int i, n = 10, sum = 0;
for (i = 0; i <= 2n; i++) {
      sum += i + 3;
}
```

```
int i, n = 20, sum = 1;
for (i = 0; i <= 2n; i++) {
      sum += i + 5;
}
```