

CS1020: DATA STRUCTURES AND ALGORITHMS I

Tutorial 9 – Analysis of Algorithms

(Week 11, starting 28 March 2016)

1. Big-O Complexity

Remember, Big-O time complexity gives us an idea of the **growth rate** of a function. In other words, for a **large input size N**, as **N increases**, in what **order of magnitude** is the volume of statements executed expected to increase?

Rearrange the following functions in increasing order of their **big-O** complexity:

$4n^2$	$\log_3 n$	$20n$	$n^{2.5}$
$n^{0.00000001}$	$\log n!$	n^n	2^n
2^{n+1}	2^{2n}	3^n	$n \log n$
$100 n^{2/3}$	$\log [(\log n)^2]$	$n!$	$(n-1)!$

2. Analyzing Time Complexity

Find the big-O time complexity of each of the following code fragments:

(a)

```
int i = 1;
while (i <= n) {
    System.out.println("*");
    i = 2 * i;
}
```

(b)

```
int i = n;
while (i > 0) {
    for (int j = 0; j < n; j++)
        System.out.println("*");
    i = i / 2;
}
```

(c)

```
while (n > 0) {
    for (int j = 0; j < n; j++)
        System.out.println("*");
    n = n / 2;
}
```

(d)

```
for (int i = 0; i < n; i++) // loop 1
    for (int j = i+1; j > i; j--) // loop 2
        for (int k = n; k > j; k--) // loop 3
            System.out.println("*");
```

(e)

```
void foo(int n) {
    if (n <= 1)
        return;
    doOhOne(); // doOhOne() runs in O(1) time
    foo(n/2);
    foo(n/2);
}
```

(f)

```
void foo(int n) {
    if (n <= 1)
        return;
    doOhN(); // doOhN() runs in O(n) time
    foo(n/2);
    foo(n/2);
}
```

3. Efficient Algorithm Design

Given an integer array A and an integer k, we want to find all pairs of elements in the array that add up to k. For each pair found, print the two numbers. If it is not possible to find any such combination, print "Not possible."

The elements in A are unique. The array elements and k could be positive, zero, or negative. You do not need to print the other permutation that represents the same pair.

Sample Run 1

```
Enter size of array: 10
Enter the elements: 2 5 6 11 -1 0 25 -3 1 8
Enter the value of k: 22
-3 25
```

Sample Run 2

```
Enter size of array: 10
Enter the elements: 2 5 6 11 -1 0 25 -3 1 8
Enter the value of k: 50
Not possible.
```

There are two (to four) algorithms to solve this problem. One is a brute force algorithm, while the others are more efficient but require more code to implement. Design and analyze the big-O time complexity of two algorithms, then implement them.

```
public void findPairsBruteForce(int[] input, int target) {
    /* TODO */
}
public void findPairsImproved(int[] input, int target) {
    mergeSort(input); // O(n log n) sort given to you
    // Efficiency should be better than the brute force algorithm!
}
```

^ ^
_

O(1)

weeks left!