

## 2

June 7, 2016

**Remark on Analysis** In general, analyze growth rate of runtime as a function of input size  $n$ . (i.e. take max over all input of size  $n$ ).

Model isn't perfect. Constant factors do matter in practice, worst case can be overly pessimistic.

### 0.0.1 3SUM Problem

Given an array  $a_i$  of numbers, are there 3 numbers (taken with replacement) that sum up to  $t$ ?

State-of-the-art:  $O(\frac{n^2}{\log n})$

```
In [4]: #  $O(N^3)$ 
def three_sum_1(numbers, t):
    for a in numbers:
        for b in numbers:
            for c in numbers:
                if a + b + c == t:
                    return True
    return False

#  $O(N^2)$ 
def three_sum_2(numbers, t):
    sums = []
    for a in numbers:
        for b in numbers:
            sums.append(a+b)
    residuals = [t - x for x in numbers]
    return len(set(sums) & set(residuals)) > 0

# helper function:  $O(N)$ 
def find_common(a1, a2):
    x1 = 0
    x2 = 0
    while x1 < len(a1) and x2 < len(a2):
        if a1[x1] == a2[x2]:
            return True
        elif a1[x1] < a2[x2]:
```

```

        x1 += 1
    else:
        x2 += 1
    return False

# O(N^2 log N)
def three_sum_3(numbers, t):
    sums = []
    for a in numbers:
        for b in numbers:
            sums.append(a+b)
    residuals = [t - x for x in numbers]
    sums.sort()
    residuals.sort()
    return find_common(sums, residuals)

# O(N^2)
def three_sum_4(numbers, t):
    numbers.sort()
    residuals = [t - x for x in reversed(numbers)] # guaranteed to be sorted
    for a in numbers:
        sums = [x + a for x in numbers] # guaranteed to be sorted
        if find_common(sums, residuals):
            return True
    return False

In [5]: test_array = [1, 2, 3, 4, 5]
        assert three_sum_1(test_array, 12)
        assert three_sum_2(test_array, 12)
        assert three_sum_3(test_array, 12)
        assert three_sum_4(test_array, 12)

```

## Math Review

- $O$  = upper bound
- $\Omega$  = lower bound
- $\Theta$  = tight bound
- $o$  = loose upper bound
- $\omega$  = loose lower bound

Def: let  $f, g : \mathbb{N} \rightarrow \mathbb{R}$ .  $f(n) \in O(g(n))$  iff  $\exists c > 0, n_0 > 0$  s.t.  $\forall n \geq n_0, f(n) \leq cg(n)$

Def:  $f(n) \in \Omega(g(n))$  iff  $\exists c > 0, n_0 > 0$  s.t.  $\forall n \geq n_0, f(n) \geq cg(n)$

Def:  $f(n) \in \Theta(g(n))$  iff  $f(n) \in O(g(n))$  and  $f(n) \in \Omega(g(n))$

Def:  $f(n) \in o(g(n))$  iff  $\forall c > 0, \exists n_0 > 0$  s.t.  $\forall n \geq n_0, f(n) < cg(n)$

Def:  $f(n) \in \omega(g(n))$  iff  $\forall c > 0, \exists n_0 > 0$  s.t.  $\forall n \geq n_0, f(n) > cg(n)$