# Algorithm Design and Time Analysis

Miles Jones	MTThF 8:30-9:50	CSE 4258

August 5, 2016

# Today's Plan

Analyzing algorithms that solve other problems (besides sorting and searching)

Designing better algorithms

- pre-processing
- re-use of computation

#### **Summing Triples: WHAT**

Given a list of real numbers

$$a_{1}$$
,  $a_{2}$ , ...,  $a_{n}$ 

look for three indices, i, j, k (each between 1 and n) such that

$$a_i + a_j = a_k$$

Does the list 3,6,5,7,8 have a summing triple?

A. Yes: 1,2,3

B. Yes: 1,3,5

C. No

#### **Summing Triples: WHAT**

Given a list of real numbers

$$a_1$$
,  $a_2$ , ...,  $a_n$ 

look for three indices, i, j, k (each between 1 and n) such that

$$a_i + a_j = a_k$$

Design an algorithm to look for summing triples

```
SumTriples1(a_1,\ldots,a_n : real numbers)
     for i := 1 to n
          for j := 1 to n
               for k := 1 to n
                    if a_i + a_j = a_k then return true
     return false
                          What's the order of the runtime of this algorithm?
                          A. O(1)
                          B. O(n)
                          C. O(n^2)
                          D. O(n^3)
                          E. None of the above
```

```
SumTriples1(a_1,\ldots,a_n: 	ext{real numbers}) for i:=1 to n for j:=1 to n for k:=1 to n if a_i+a_j=a_k then return true return false
```

**Improvements??** 

```
SumTriples1(a_1,\ldots,a_n: 	ext{real numbers})

for i:=1 to n. Eliminate redundancy for j:=\cancel{k} to n

for k:=1 to n

if a_i+a_j=a_k then return true

return false
```

 $SumTriples2(a_1,\ldots,a_n : real numbers)$ for i := 1 to nEliminate redundancy for j := i to nfor k := 1 to nif  $a_i + a_j = a_k$  then return true return false What's the order of the runtime of this algorithm? A. O(1) B. O(n) C.  $O(n^2)$ D. O(n<sup>3</sup>) E. None of the above

 $SumTriples2(a_1,\ldots,a_n: real numbers)$ 

for i := 1 to n

Eliminate redundancy

for j := i to n

for k := 1 to n

if  $a_i + a_j = a_k$  then return true

return false

**Improvements??** 

#### Reframing what we did:

return false

```
SumTriples2(a_1,\ldots,a_n: 	ext{real numbers}) for i:=1 to n for j:=i to n For each candidate sum a_i+a_j, for k:=1 to n do linear search to find it if a_i+a_j=a_k then return true
```

**Improvements??** 

 $SumTriples2(a_1,\ldots,a_n : real numbers)$ 

$$\label{eq:for} \begin{aligned} \mathbf{for} \ i := 1 \ \mathbf{to} \ n \\ \mathbf{for} \ j := i \ \mathbf{to} \ n \end{aligned} \qquad \mathsf{For} \ \mathsf{each} \ \mathsf{candidate} \ \mathsf{sum} \ \mathsf{a_i+a_j},$$

 $\mathbf{for}\; k := 1\; \mathbf{to}\; n$  do linear search to find it  $\mathbf{if}\; a_i + a_j = a_k \; \mathbf{then}\; \mathbf{return}\; true$  return false

We have a faster search than linear search!

 $SumTriples3(a_1,\ldots,a_n: real numbers)$ 

for j := i to n

For each candidate sum  $a_i+a_j$ ,

**if**  $BinarySearch(a_i + a_j; a_1, \dots, a_n)$ 

then return true

return false

for i := 1 to n

do binary search to find it

**How long would this take?** 

- A. O(n<sup>3</sup>)
- B. O(n<sup>2</sup>)
- C. O(n<sup>2</sup> log n)
- D. O(n log n)

```
SumTriples3(a_1, \ldots, a_n : real numbers)
```

 $\begin{aligned} \mathbf{for} \ i := 1 \ \mathbf{to} \ n \\ \mathbf{for} \ j := i \ \mathbf{to} \ n \end{aligned} \qquad \mathsf{For} \ \mathsf{each} \ \mathsf{candidate} \ \mathsf{sum} \ \mathsf{a_i+a_j},$ 

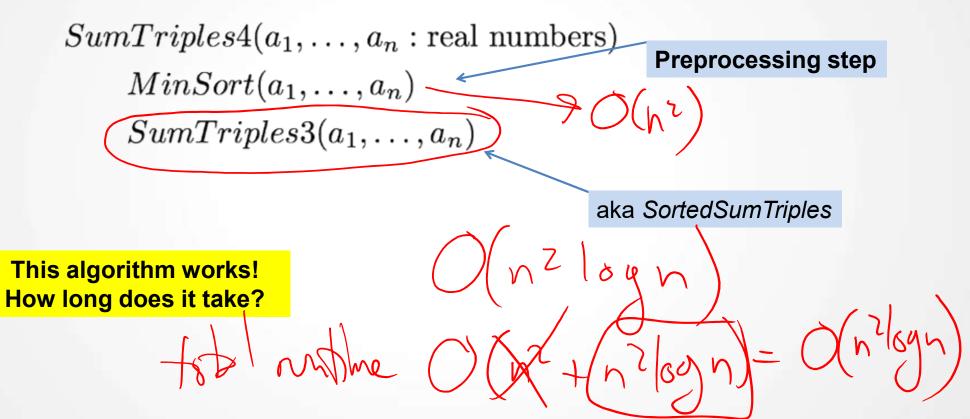
**if**  $BinarySearch(a_i + a_j; a_1, \dots, a_n)$ 

then return true

Does this algorithm really work???

```
SumTriples3(a_1,\ldots,a_n: 	ext{real numbers}) for i:=1 to n for j:=i to n For each candidate sum a_i+a_j, if BinarySearch(a_i+a_j;a_1,\ldots,a_n) then return true do binary search to find it
```

Does this algorithm really work???



```
SumTriples4(a_1,\ldots,a_n): real numbers) MinSort(a_1,\ldots,a_n) O(n²) SumTriples3(a_1,\ldots,a_n) O(n² log n)
```

Sum is maximum: O(n<sup>2</sup> log n)

```
SumTriples4(a_1,\ldots,a_n) : real numbers) MinSort(a_1,\ldots,a_n) O(n²) SumTriples3(a_1,\ldots,a_n) O(n² log n)
```

Sum is maximum: O(n<sup>2</sup> log n)

Have we made progress?
Can we do better?

- SumTriples4 does better than O(n³).
- Using a faster sort won't help overall.
- But .... fastest known algorithm: O(n²)

### "Tight"?

To know that we've actually made improvements, need to make sure our original analysis was not overly pessimistic.

A **tight** bound for runtime is a function g(n) so that the runtime is in  $\Theta(g(n))$ 

The big-O class for our algorithm : upper bound.

Now want matching big- $\Omega$ : lower bound.

# Summing Triples: WHEN (1)

$$SumTriples1(a_1,\ldots,a_n: real numbers)$$

for 
$$i := 1$$
 to  $n$ 

for 
$$j := 1$$
 to  $n$ 

for 
$$k := 1$$
 to  $n$ 

if  $a_i + a_j = a_k$  then return true

return false

What's the **lower bound** order of the **worst** case runtime of this algorithm?

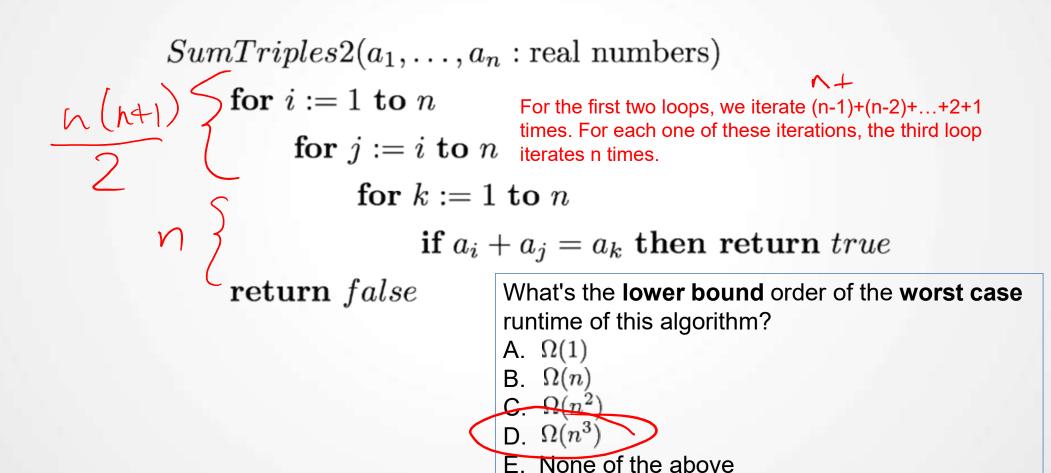
- A.  $\Omega(1)$
- B.  $\Omega(n)$
- C.  $\Omega(n^2)$
- D  $\Omega(n^3)$
- E. None of the above



# Summing Triples: WHEN (1)

Strategy: work from the inside out

# Summing Triples: WHEN (2)



# Summing Triples: WHEN (2)

```
SumTriples2(a_1,\ldots,a_n: 	ext{real numbers}) for i:=1 to n for j:=i to n for k:=1 to n if a_i+a_j=a_k then return true return false
```

Observe: in both these examples, the product rule for calculating the nested loop runtime gave us tight upper bounds ... is that always the case?

#### When is the product rule for nested loops tight?

**Nested code:** 

If Guard Condition is O(1) and body of the loop has runtime  $O(T_2)$  in the worst case and run at most  $O(T_1)$  iterations, then runtime is

$$O(\overline{T_1T_2})$$

But what if many t<sub>k</sub> are much better than the worst case?

Given two sorted lists

$$a_1$$
,  $a_2$ , ...,  $a_n$  and  $b_1$ ,  $b_2$ , ...,  $b_n$ 

determine if there are indices i,j such that

$$a_i = b_j$$

Design an algorithm to look for indices of intersection

#### Given two sorted lists

$$a_1$$
,  $a_2$ , ...,  $a_n$  and  $b_1$ ,  $b_2$ , ...,  $b_n$ 

determine if there are indices i,j such that

$$a_i = b_j$$

#### **High-level description:**

- Use linear search to see if b<sub>1</sub> is anywhere in first list, using early abort
- Since b<sub>2</sub>>b<sub>1</sub>, start the search for b<sub>2</sub> where the search for b<sub>1</sub> left off
- And in general, start the search for  $b_j$  where the search for  $b_{j-1}$  left off



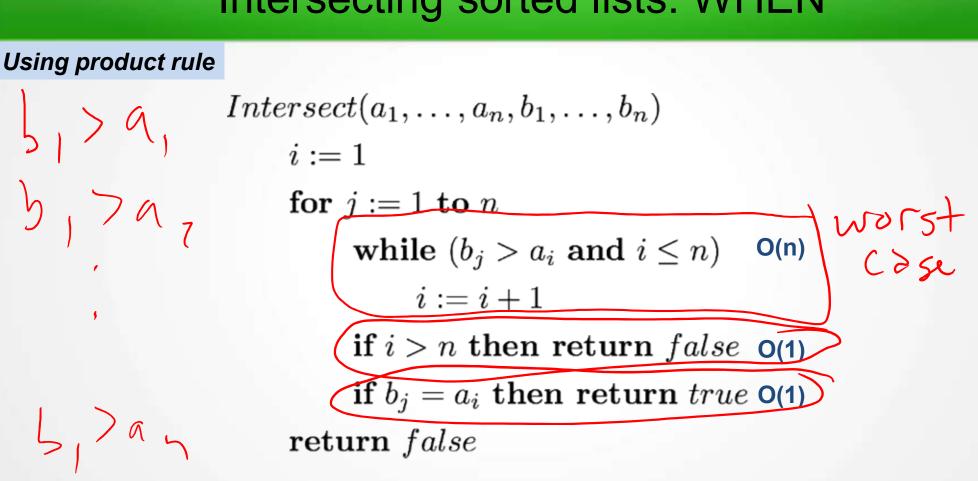
# XELan

#### Intersecting sorted lists: WHY

```
egin{aligned} Intersect(a_1,\dots,a_n,b_1,\dots,b_n)\ &i:=1\ &	ext{for } j:=1 	ext{ to } n\ &	ext{while } (b_j>a_i 	ext{ and } i\leq n)\ &i:=i+1\ &	ext{if } i>n 	ext{ then return } false\ &	ext{if } b_j=a_i 	ext{ then return } true\ &	ext{return } false\ \end{aligned}
```

To practice: trace examples & generalize argument for correctness

#### Using product rule



#### Using product rule

$$Intersect(a_1, \dots, a_n, b_1, \dots, b_n)$$
  $i := 1$  for  $j := 1$  to  $n$ 



return false

Total: O(n²)

#### More careful analysis ...

 $Intersect(a_1, \ldots, a_n, b_1, \ldots, b_n)$  i := 1 for j := 1 to n

Every time this is executed (except last time in each iteration of for loop), i is incremented. If i ever reaches n+1, the program terminates (returns)

while 
$$(b_j > a_i \text{ and } i \leq n)$$
  
 $i := i + 1$ 

if i > n then return falseif  $b_j = a_i$  then return truereturn false

#### More careful analysis ...

 $Intersect(a_1, \dots, a_n, b_1, \dots, b_n)$  i := 1 for j := 1 to n

This executes O(2n) times total (across all iterations of for loop)

while 
$$(b_j > a_i \text{ and } i \leq n)$$
  
 $i := i + 1$ 

if i > n then return falseif  $b_j = a_i$  then return truereturn false

#### More careful analysis ...

 $Intersect(a_1, \dots, a_n, b_1, \dots, b_n)$  i := 1 for j := 1 to n

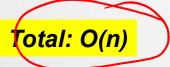
This executes O(2n) times total (across all iterations of for loop)

while 
$$(b_j > a_i \text{ and } i \leq n)$$
  
 $i := i + 1$ 

if i > n then return false

if  $b_j = a_i$  then return true

return false



product rule analysis wasn't tight in this case!