# COMP-250



Moving to Otto MAAS 112 - Friday Jan 10

## List Intersection

250: Alice Bob Carl
206: Bob Joker David

How many elements are in common?

In example: 1

2 arrays: a of n elements
b of m elements

a, b have same type of elements that can be compared

Two indices i, j to keep track where we are

Algorithm listIntersection(a, n, b, m)
Input: array a of size n and array b of size m
Output: Number of elements in common
int i, j // indices for the arrays
int intersect // nr. of elements in common
intersect ← 0
for i ← 0 to n-1 do // for each el. of a
  for j ← 0 to m-1 do // look for it in b } search
    if a[i] = b[j] then
      intersect ← intersect + 1; break
return intersect;

1) All elements in a are inspected
2) intersect is incremented only if an a[i] is found in b


Running time: how many comparisons?

$1+2+3=6$

All elems in a are distinct; same in b

Worst case: intersect = 0; 9 comparisons

General: $\underbrace{m+m+\cdots+m}_{n \text{ times}} = m \cdot n$

(each elem in a is compared to m elements in b)

"$O(m \cdot n)$"

1) Complexity or running time dep on size of input
2) Also dep on content of input
   - "Best" case
   - "Worst" case ($O()$)
   - "Average" case

Suppose $k$ of $a$'s elements are in $b$

$n-k$ not in $b$

For $(n-k)$ → look all the way to <u>end</u> of $b$

$\Rightarrow (n-k) \cdot m$ comparisons

Others: found somewhere

Rough guess: on average found in "middle"

$\Rightarrow k \cdot \frac{m}{2}$

Total time: $(n-k) \cdot m + k \cdot \frac{m}{2} = n \cdot m - \frac{k \cdot m}{2}$

dep on $k$!

Further assumptions on $k$

List intersection requires searching in an array
Searching is important for other algs
Write search separately

Algorithm search (a, n, val)
Input: Array a of size n, value val
Output: true if val is in a, false otherwise
int i ← 0 // index for a
while i < n do
    if a[i] = val return true
    i ← i+1 // if not found, keep looking
return false

To use search algorithm only need to know its interface

≃ n Comp
O(n)

Algorithm listIntersection (a, n, b, m) ← simpler by using building block
Input, output as before
int i
int intersect ← 0 // as before
for i ← 0 to n-1 do // loop over a
    if search (b, m, a[i]) then
        intersect ← intersect + 1
return intersect

n · Running time of Search on b ⇒ n · m

Interface of an alg as a contract

Hides details of implementation.

Libraries → only need to know interface of fns/ methods / data structures

Abstraction → methods
→ data
⇒ General, re-use parts build complex stuff

Running times also computed based on the running times of building blocks