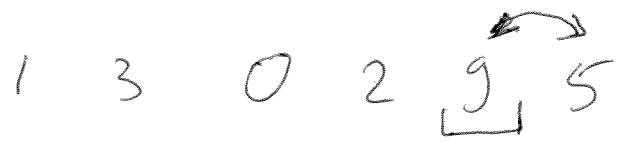
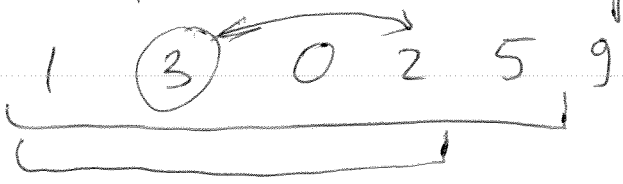
COMP 250 – Lecture 11

- Sorting; lower bounds, proving correctness
  - ↳ other algs (eg. binary search) work faster on sorted arrays

SelectionSort (MergeSort, QuickSort)



- Find max element → put it in last place



index i: last element of interest

1 2 0 3 5 9

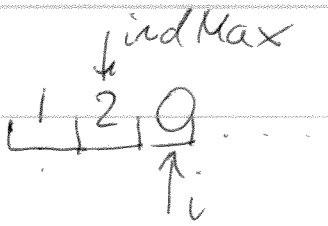
Helper alg – find index of max element

Helper alg – swap

Algorithm findMaxIndex(a, n)

Input: Array a of length n

Output: ~~Array a with~~ index of max element




```
int indMax ← 0
  max ← a[0]
for i ← 1 to n-1                    } O(n)
    if a[i] > max then              } indMax indicates greatest seen SO far
        max ← a[i]                  } a[indMax] ≥ a[j] ∀ j ≤ i
        indMax ← i                  }
    endif                           }
endfor
return indMax
```

At end: $a[indMax] \geq a[i] \ \forall\ i = 0 \ldots n-1$

Lower bound on running time: how fast can the alg run and still behave correctly?

findMaxIndex → each element of a is touched once

Can we do better? No!

If any element is untouched, adversary can hide the max there → alg could be wrong.

Adversarial argument → make sure that code works against any data given (useful to reason in security/encryption applications)

Loop invariant $\forall j \leq i,\ a[indMax] \geq a[j]$ Holds every time we go through the loop

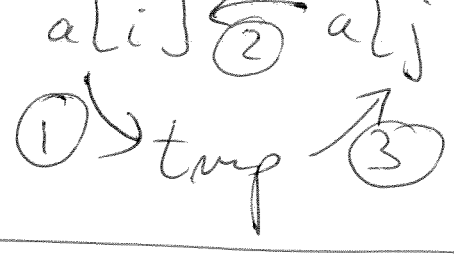
Alg

for (. . .) → loop invariant
for (. . .) → loop invariant
} ⇒
⇒ end result is correct

Algorithm swap (a, i, j)
Input: Array a, indices i, j
Output: a with content at a[i] and a[j] reversed

```
tmp ← a[i]
a[i] ← a[j]
a[j] ← tmp
```

O(1)



~~SE~~ Algorithm SelectionSort (a, n)
Input: array a of size n; a has comparable elements
Output: a has content in increasing order

```
int i ← n-1 // index of last element we're considering
while i > 0
    int indMax ← findMaxIndex(a, i) // O(n)? O(i)
    swap (a, indMax, i)
    i ← i-1
end while
```

n times

findMaxIndex(a, i) → a[indMax] ≥ a[j] ∀ j ≤ i

swap (a, indMax, i) → a[i] ≥ a[j] ∀ j ≤ i

Product rule: $O(n \cdot n) = O(n^2)$

Loop 'runs' n times

More precise calculation of running time:

1st findMaxIndex → n elements

2nd - - - $n-1$

3 - - - - - $n-2$

. . . . .

1

$n + (n-1) + (n-2) \cdots + 2 + 1$ "operations"

$n+1$

$n+1$

. . . . .

$(n+1) \cdot \frac{n}{2}$ → Proof by induction

$O(n^2)$

Loop invariant: after while i

$a[i] \geq a[j] \quad \forall j < i$ (and also $a[i] \leq a[k] \; \forall k > i$)

For array to be sorted correctly:

$$a[j] \leq a[i] \quad \forall j \leq i, \forall i = 0 \ldots n-1$$

Directly implied by loop invariant!

a) Used an already proven correctness;
"Put our faith" in fn calls

b) Build Proof based on pieces of code.