Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2025

Brute Force Practice

```
**ALGORITHM** BubbleSort($A$)
    //Input: an array $A[0..n-1]$
    for $i \leftarrow 0..n-2$ do
        for $j \leftarrow 0..n-2-i$ do
            if $A[j+1] < A[j]$ then
                swap $A[j+1]$ and $A[j]$
```

Size Metric:

Basic operations:

Best/average/worst cases?

Number of comparisions/worst case number of swaps:

$C(n) =$

```
**ALGORITHM** SELECTIONSORT($A$)
    //Input: an array $A[0..n-1]$
    **for** $i \leftarrow 0..n-2$ **do**
        $min \leftarrow i$
        **for** $j \leftarrow i+1..n-1$ **do**
            **if** $A[j] < A[min]$ **then**
                $min \leftarrow j$
        swap $A[i]$ and $A[min]$
```

Number of comparisons:

$C(n) =$

Number of swaps:

$S(n) =$

The brute-force **sequential search** algorithm searches for a *key* element k in an array of elements by starting at the beginning of the array and scanning through sequentially element by element until it finds k, or it reaches the end of the array without finding it. Below is pseudocode for sequential search with two parts missing. Study the code and then fill in the boxes with the missing pseudocode.

```
ALGORITHM SequentialSearch($A$, k)
    //Input: an array $A[0..n-1]$
    //Input: a key element $k$
    //Output: The index of the first occurrence of $k$ in $A$
    $i \leftarrow 0$
    while $i < n$ and $\boxed{\phantom{xxxx}}$ do
        $i \leftarrow i + 1$
    if $i < n$ then
        return $i$
    else
        return $\boxed{\phantom{xxxx}}$
```

For an n-element input array, exactly how many *key comparisons* does the algorithm make in the worst case? (A key comparison is any comparison made between the key k and an element of the array. The number of key comparisons often determines the asymptotic running time of searching algorithms, which is why we count them.)

The best-case efficiency of an algorithm is not nearly as important as the worst-case efficiency. But it is not completely useless either. For example, some sorting algorithms perform best on nearly sorted data, which is a common situation in practice. For an n element array, exactly how many key comparisons does sequential search make in the best case?

A simple trick is often used in implementing sequential search: if the key k is added to the end of the array (assume there is space for it), then the search will find the key there (for an unsuccessful search), if not sooner (for a successful search). The advantage of this is that we can eliminate the check $i < n$ in each iteration of the **while** loop. Write pseudocode for this version of sequential search below.

```
ALGORITHM SequentialSearch($A$, k)
    //Input: an array $A[0..n-1]$ and extra slot $A[n]$ to store $k$
    //Input: a key element $k$
    //Output: The index of the first occurrence of $k$ in $A$
```

```
ALGORITHM BruteForceStringMatch($T$, $P$)
    //Input: a text string $T[0..n-1]$
    //Input: a pattern string $P[0..m-1]$
    for $i \leftarrow 0..n-m$ do
        $j \leftarrow 0$
        while $j < m$ and $P[j] = T[i+j]$ do
            $j \leftarrow j+1$
        if $j = m$ then
            return $i$
    return $-1$
```

Size metrics:

Basic operation:

Best/average/worst cases:

Levitin Exercise 3.1.4, p. 102