FISF130020: Introduction to Computer Science

# Lecture 5: Algorithm I

Hui Xu

xuh@fudan.edu.cn

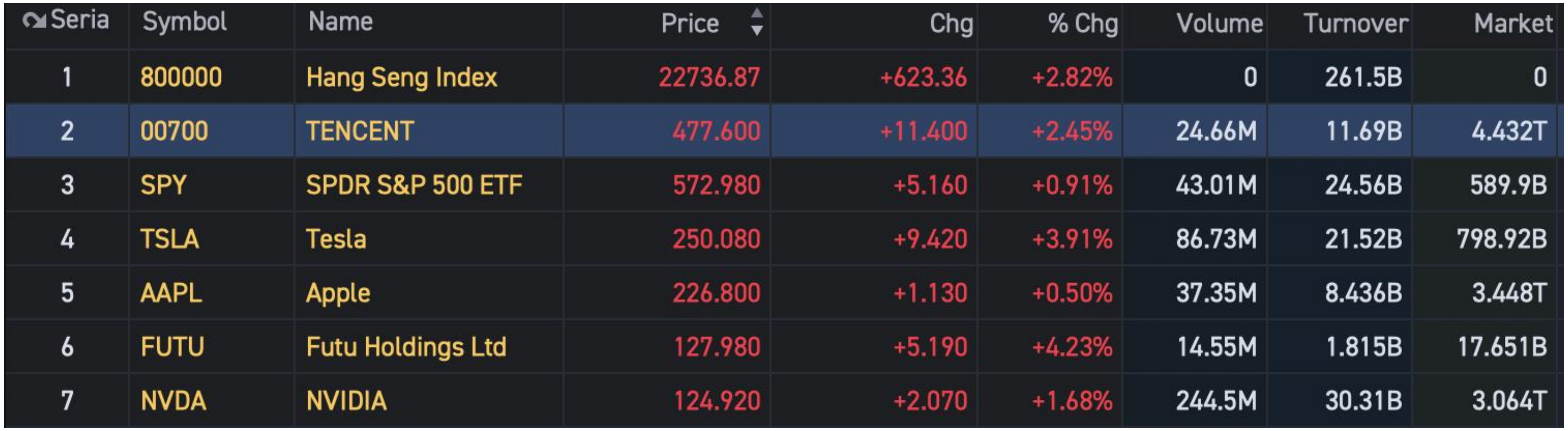
# Outline

- 1. Sorting Algorithm
- 2. String Matching
- 3. In-class Practice

# 1. Sorting Algorithm

# Scenario

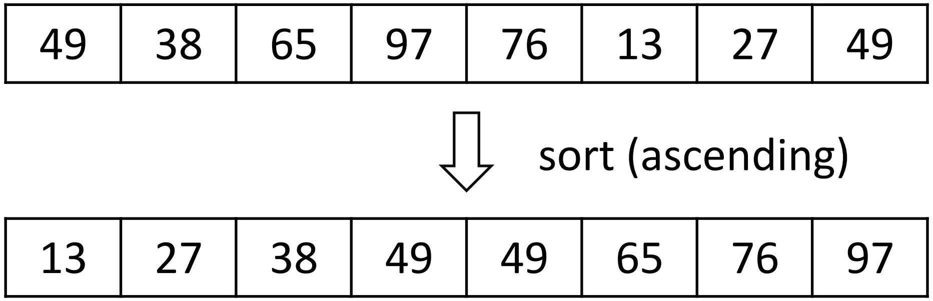
- We want to display stocks in ascending or descending order by name, price, volume, or other criteria.



| Seria | Symbol | Name | Price | Chg | % Chg | Volume | Turnover | Market |
|---|---|---|---|---|---|---|---|---|
| 1 | 800000 | Hang Seng Index | 22736.87 | +623.36 | +2.82% | 0 | 261.5B | 0 |
| 2 | 00700 | TENCENT | 477.600 | +11.400 | +2.45% | 24.66M | 11.69B | 4.432T |
| 3 | SPY | SPDR S&P 500 ETF | 572.980 | +5.160 | +0.91% | 43.01M | 24.56B | 589.9B |
| 4 | TSLA | Tesla | 250.080 | +9.420 | +3.91% | 86.73M | 21.52B | 798.92B |
| 5 | AAPL | Apple | 226.800 | +1.130 | +0.50% | 37.35M | 8.436B | 3.448T |
| 6 | FUTU | Futu Holdings Ltd | 127.980 | +5.190 | +4.23% | 14.55M | 1.815B | 17.651B |
| 7 | NVDA | NVIDIA | 124.920 | +2.070 | +1.68% | 244.5M | 30.31B | 3.064T |

# The Sorting Problem in General

- Given an array of elements, output a new array sorted in either ascending or descending order.
- Classic solutions:
  - Selection sort
  - Bubble sort
  - Quick sort
  - Radix sort



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 49 | 38 | 65 | 97 | 76 | 13 | 27 | 49 |

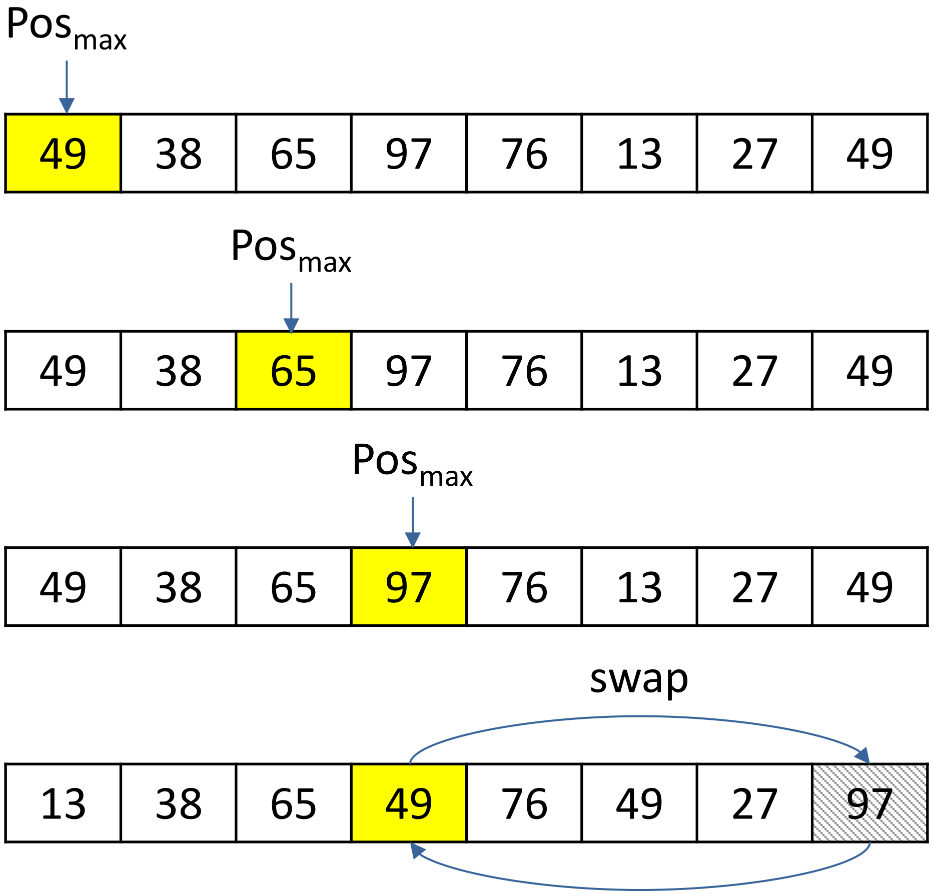
sort (ascending)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 13 | 27 | 38 | 49 | 49 | 65 | 76 | 97 |

# Selection Sort

- In each round, find the largest element.
- Swap it with the last unsorted element.

Round 1:



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 49 | 38 | 65 | 97 | 76 | 13 | 27 | 49 |

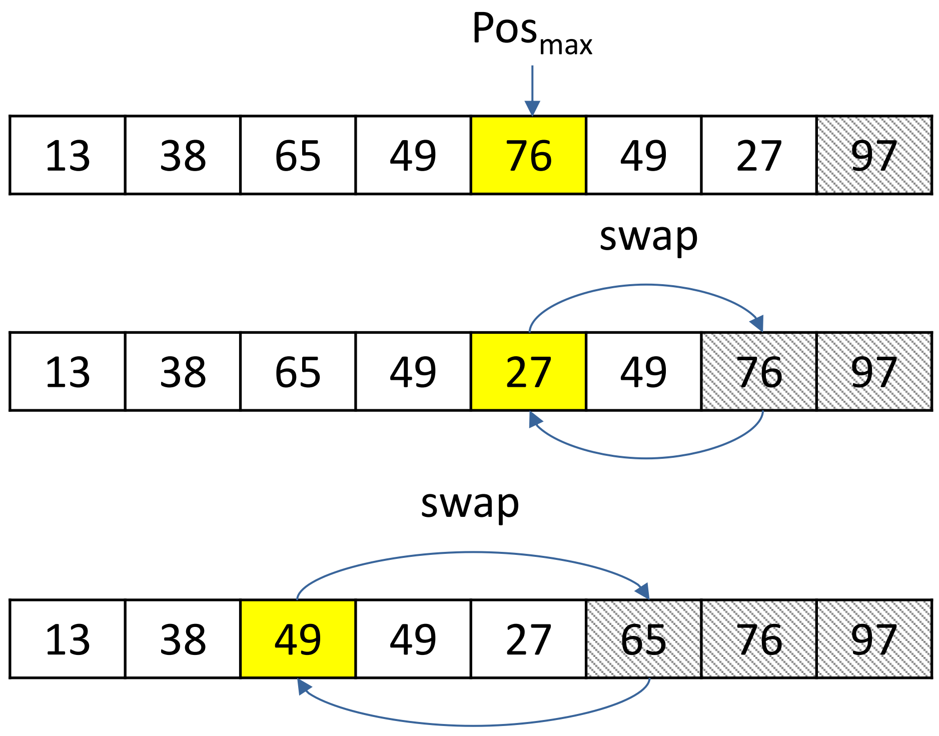
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 49 | 38 | 65 | 97 | 76 | 13 | 27 | 49 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 49 | 38 | 65 | 97 | 76 | 13 | 27 | 49 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 13 | 38 | 65 | 49 | 76 | 49 | 27 | 97 |

# Selection Sort

- Repeat the selection and swap operations iteratively.



Round 2:

$\text{Pos}_{\text{max}}$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 13 | 38 | 65 | 49 | 76 | 49 | 27 | 97 |

swap

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 13 | 38 | 65 | 49 | 27 | 49 | 76 | 97 |

swap

Round 3:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 13 | 38 | 49 | 49 | 27 | 65 | 76 | 97 |

…

# Selection Sort Algorithm

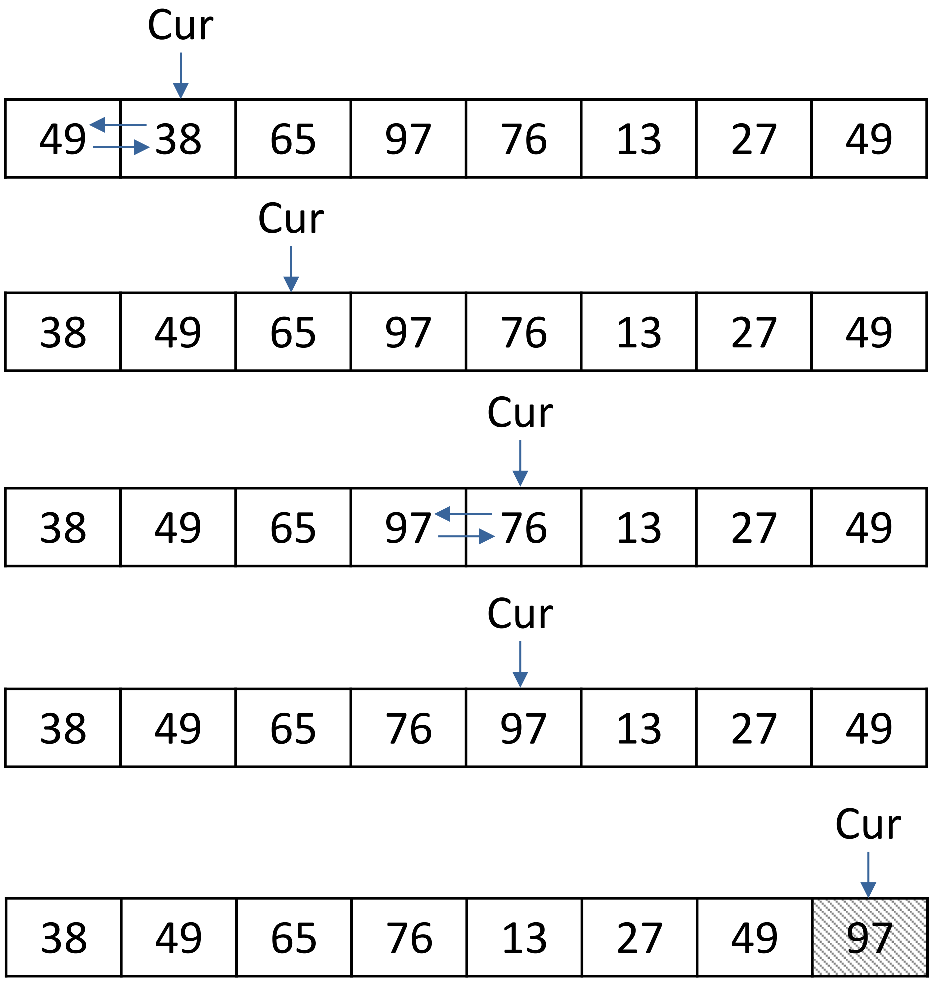
```
fn selection_sort(arr: &mut [i32]) {
    let n = arr.len();
    for i in 0..n {
        let mut max_idx = i;
        for j in (i + 1)..n {
            if arr[j] > arr[max_idx] {
                max_idx = j;
            }
        }
        arr.swap(i, max_idx);
    }
}
```

# Complexity and Big O Notation

- Complexity analysis:
  - How many rounds do we need to perform?
  - How many comparisons are needed in each round?
  - How many comparisons are needed in total?
- Order of approximation: $O(n^2)$
  - $(n-1) \times \frac{(n-1)+1}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$

# Bubble Sort

- Swap two adjacent elements if they are not in ascending order.



Cur

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 49 | 38 | 65 | 97 | 76 | 13 | 27 | 49 |

Cur

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 38 | 49 | 65 | 97 | 76 | 13 | 27 | 49 |

Cur

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 38 | 49 | 65 | 97 | 76 | 13 | 27 | 49 |

Cur

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 38 | 49 | 65 | 76 | 97 | 13 | 27 | 49 |

Cur

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 38 | 49 | 65 | 76 | 13 | 27 | 49 | 97 |

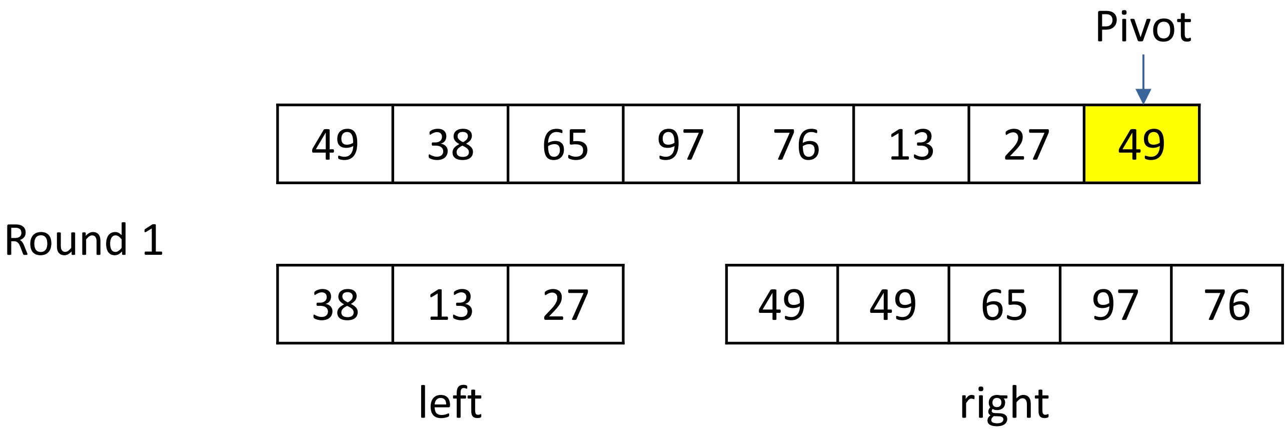
# Bubble Sort Algorithm

```
fn bubble_sort(arr: &mut [i32]) {
    let n = arr.len();
    for i in 0..n {
        let mut swapped = false;
        for j in 0..(n - i - 1) {
            if arr[j] > arr[j + 1] {
                arr.swap(j, j + 1);
                swapped = true;
            }
        }
        if !swapped {
            break;
        }
    }
}
```

Bubble sort performs better if the array is already sorted.

# Quick Sort: A Faster Algorithm

- Divide-and-conquer approach:
  - Select a pivot element from the array and partition the other elements into two sub-arrays in each round.
  - All elements in the left array are less than the pivot.
  - All elements in the right array are grater than or equal to the pivot.
  - Recursively sorted the sub-arrays.



Pivot

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 49 | 38 | 65 | 97 | 76 | 13 | 27 | 49 |

Round 1

| | | |
|---|---|---|
| 38 | 13 | 27 |

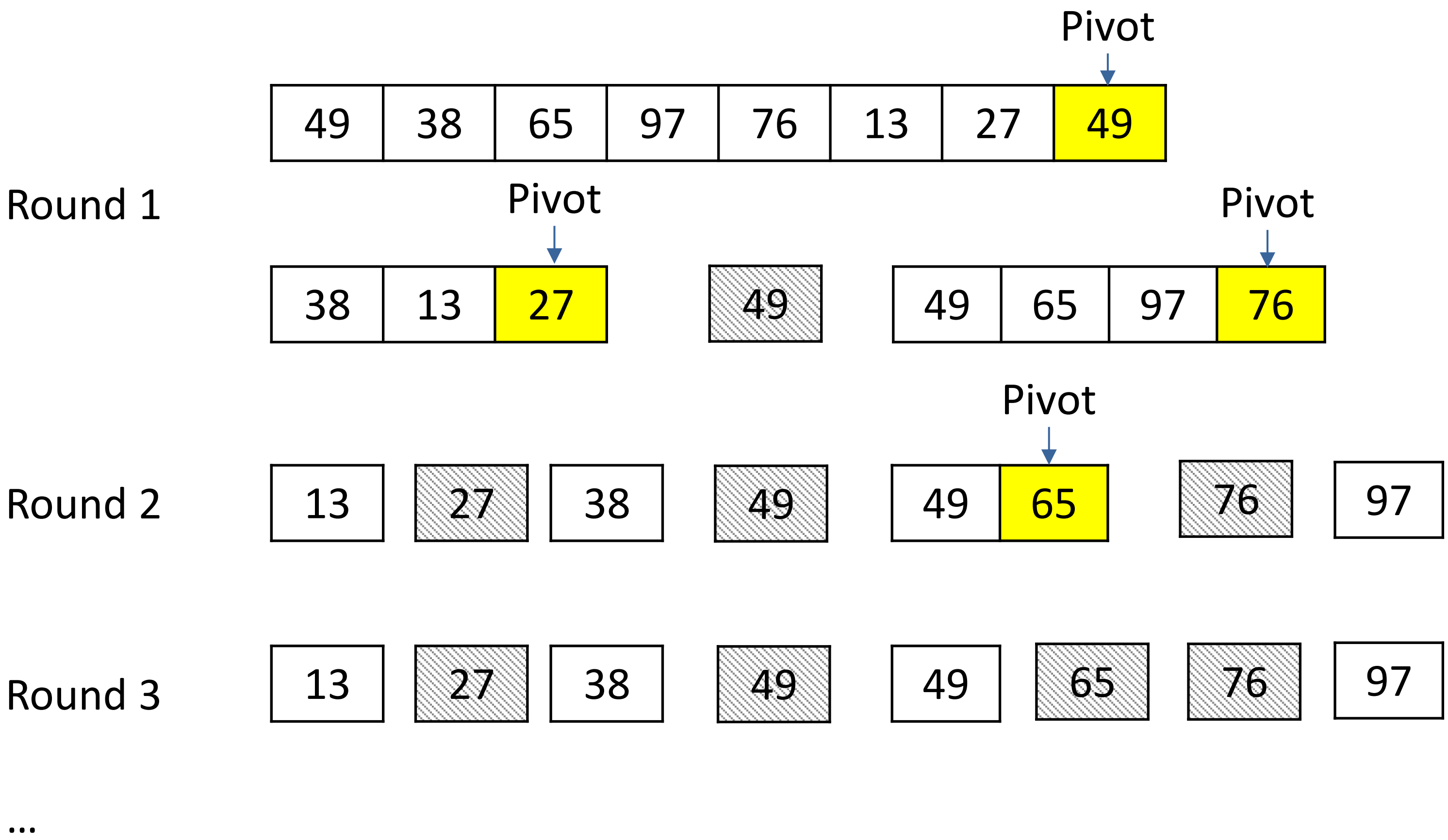
left

| | | | | |
|---|---|---|---|---|
| 49 | 49 | 65 | 97 | 76 |

right

# Quick Sort

- Average complexity: $O(nlogn)$
- Worst-case complexity: $O(n^2)$



Pivot

49 38 65 97 76 13 27 49

Round 1

Pivot Pivot

38 13 27 49 49 65 97 76

Pivot

Round 2

13 27 38 49 49 65 76 97

Round 3

13 27 38 49 49 65 76 97

…

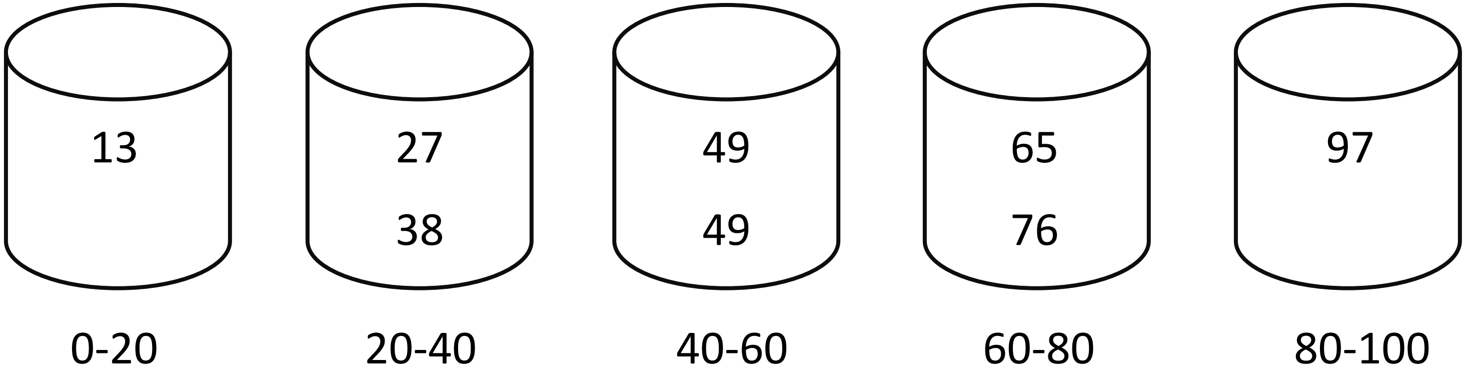
# Quick Sort Algorithm

```
fn quick_sort(arr: &mut [i32]) {
    if arr.len() <= 1 { return; }
    let pivot_index = partition(arr);
    let (left, right) = arr.split_at_mut(pivot_index);
    quick_sort(left);
    quick_sort(&mut right[1..]);
}
fn partition(arr: &mut [i32]) -> usize {
    let pivot_index = arr.len() - 1;
    let pivot = arr[pivot_index];
    let mut i = 0;
    for j in 0..pivot_index {
        if arr[j] < pivot {
            arr.swap(i, j);
            i += 1;
        }
    }
    arr.swap(i, pivot_index);
    i
}
```

# Quicker: Bucket Sort

- Instead of dividing the elements into two subset, we distribute them into multiple subsets or buckets.

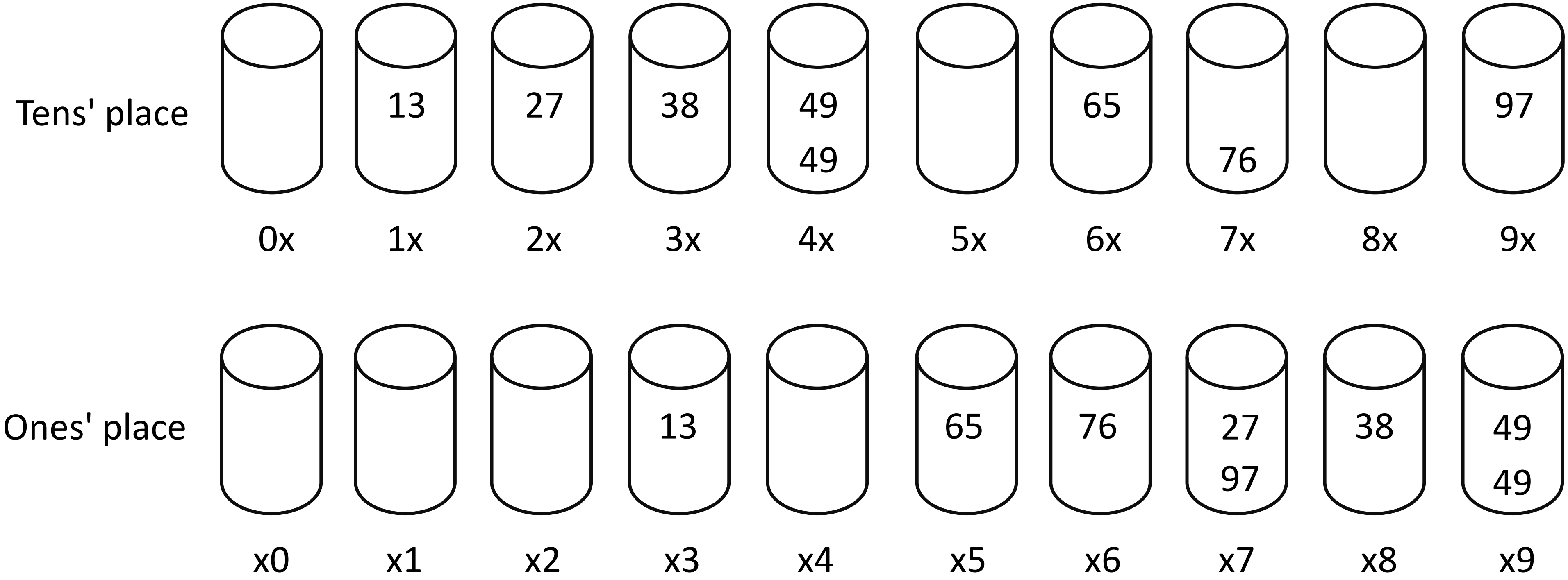
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 49 | 38 | 65 | 97 | 76 | 13 | 27 | 49 |



# Radix Sort:

- Distribute the elements based on the digits at each position. Then, select the elements in ascending order.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 49 | 38 | 65 | 97 | 76 | 13 | 27 | 49 |



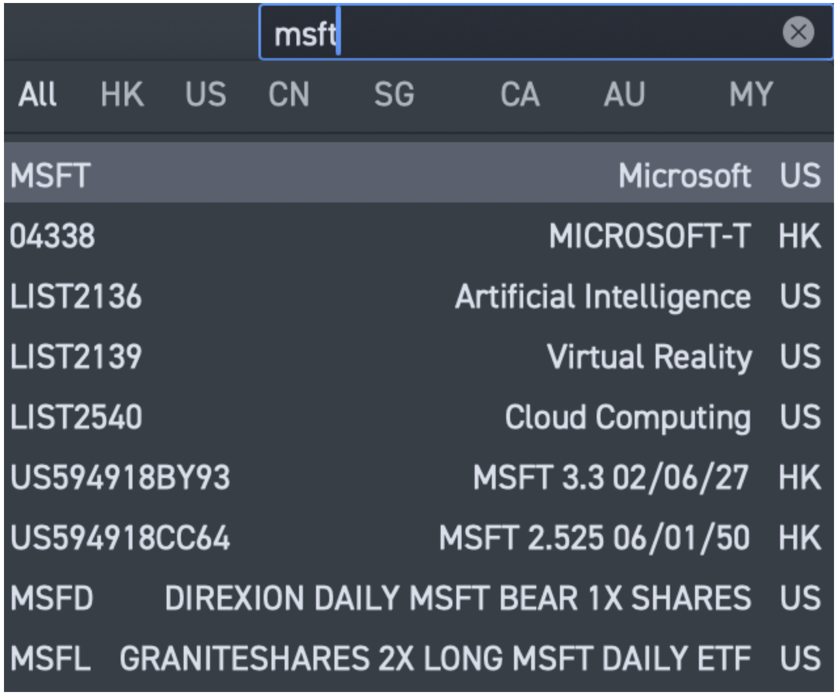
# Complexity of Radix Sort

- Suppose the array length is n, and each element has at most $w$ digits.
- We need to distribute the elements $w \times n$ times.
- Cost: Additional space is required to keep track of the distributions

# 2. String Matching

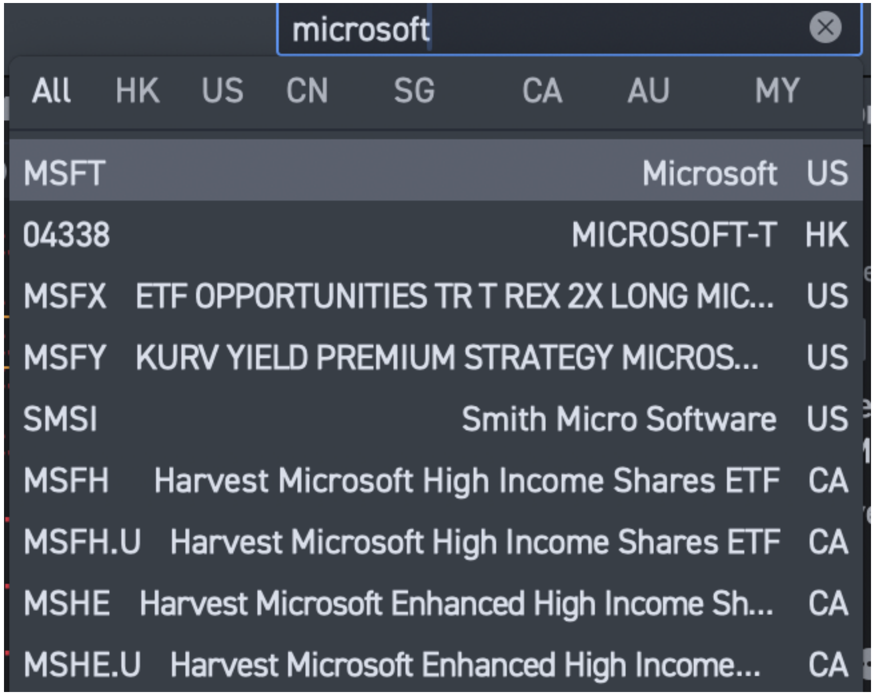
# Scenario

- We want to search for a stock by its ticker symbol or company name.



msft

| | | |
|---|---|---|
| MSFT | Microsoft | US |
| 04338 | MICROSOFT-T | HK |
| LIST2136 | Artificial Intelligence | US |
| LIST2139 | Virtual Reality | US |
| LIST2540 | Cloud Computing | US |
| US594918BY93 | MSFT 3.3 02/06/27 | HK |
| US594918CC64 | MSFT 2.525 06/01/50 | HK |
| MSFD | DIREXION DAILY MSFT BEAR 1X SHARES | US |
| MSFL | GRANITESHARES 2X LONG MSFT DAILY ETF | US |



microsoft

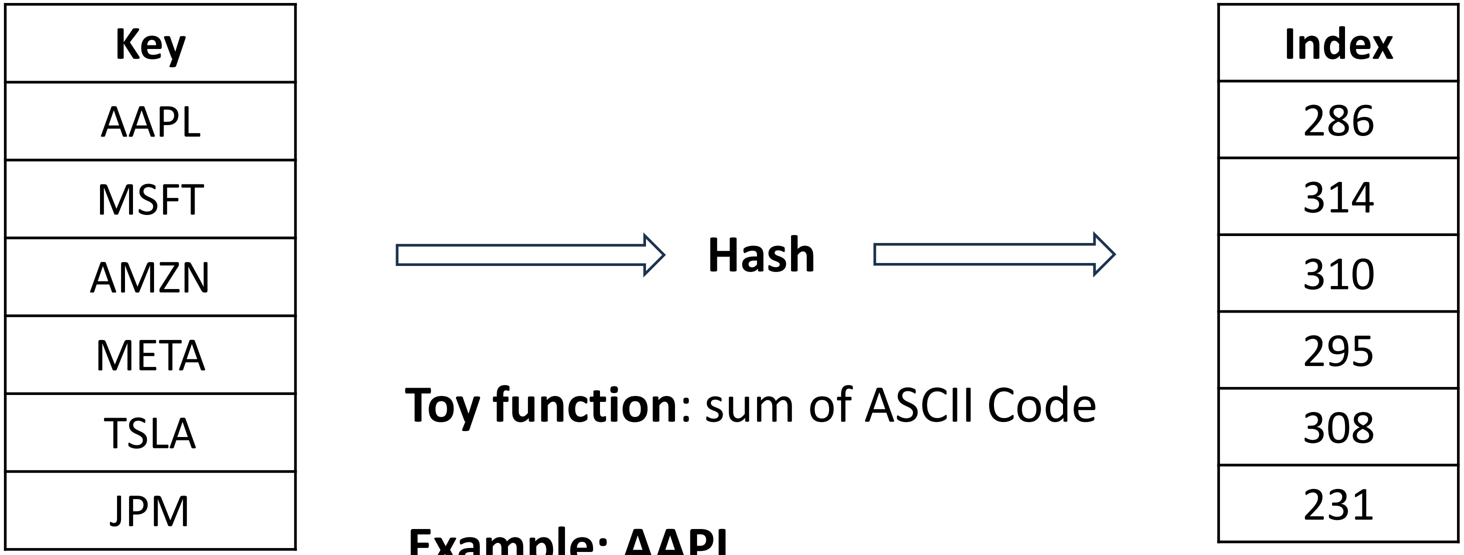
| | | |
|---|---|---|
| MSFT | Microsoft | US |
| 04338 | MICROSOFT-T | HK |
| MSFX | ETF OPPORTUNITIES TR T REX 2X LONG MIC... | US |
| MSFY | KURV YIELD PREMIUM STRATEGY MICROS... | US |
| SMSI | Smith Micro Software | US |
| MSFH | Harvest Microsoft High Income Shares ETF | CA |
| MSFH.U | Harvest Microsoft High Income Shares ETF | CA |
| MSHE | Harvest Microsoft Enhanced High Income Sh... | CA |
| MSHE.U | Harvest Microsoft Enhanced High Income... | CA |

# Hash (Lookup) Table

- Map each string (key) to a number using a hash function.
- The search time is constant.
- A tradeoff between space and time.

| Index | Key | Value |
|---|---|---|
| … | | |
| 231 | JPM | |
| … | | |
| 286 | AAPL | |
| … | | |
| 295 | META | |
| … | | |
| … | | |
| 308 | TSLA | |
| … | | |
| 310 | AMZN | |
| … | | |
| 314 | MSFT | |

# Hash (Lookup) Table



| Key |
|---|
| AAPL |
| MSFT |
| AMZN |
| META |
| TSLA |
| JPM |

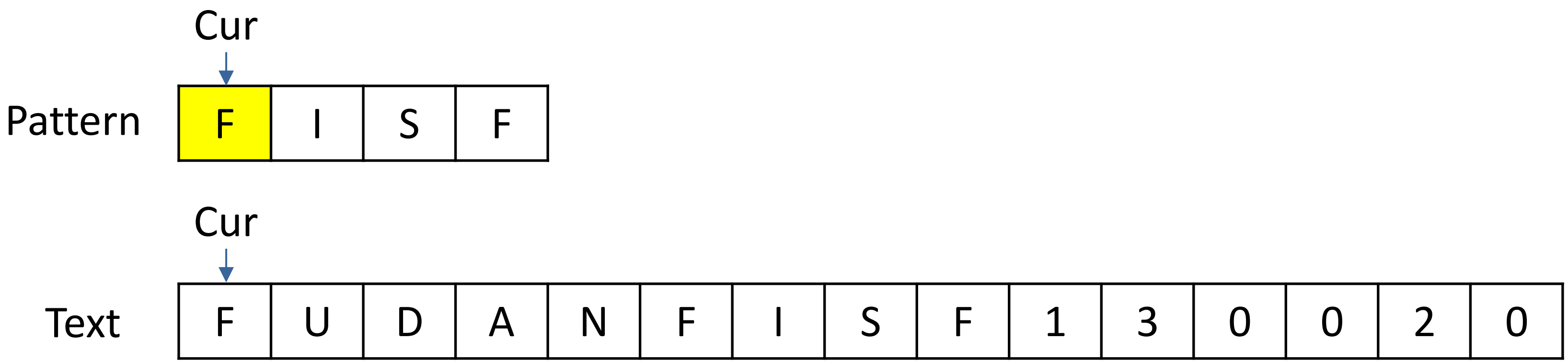
| Index |
|---|
| 286 |
| 314 |
| 310 |
| 295 |
| 308 |
| 231 |

**Toy function**: sum of ASCII Code

**Example: AAPL**
'A' = 65
'A' = 65
'P' = 80
'L' = 76
**Sum**: 65+65+80+76 = 286

# The General String Matching Problem

- How to find the place that a string pattern appears in a text?
  - Naive approach
  - KMP (Knuth-Morris-Pratt) algorithm



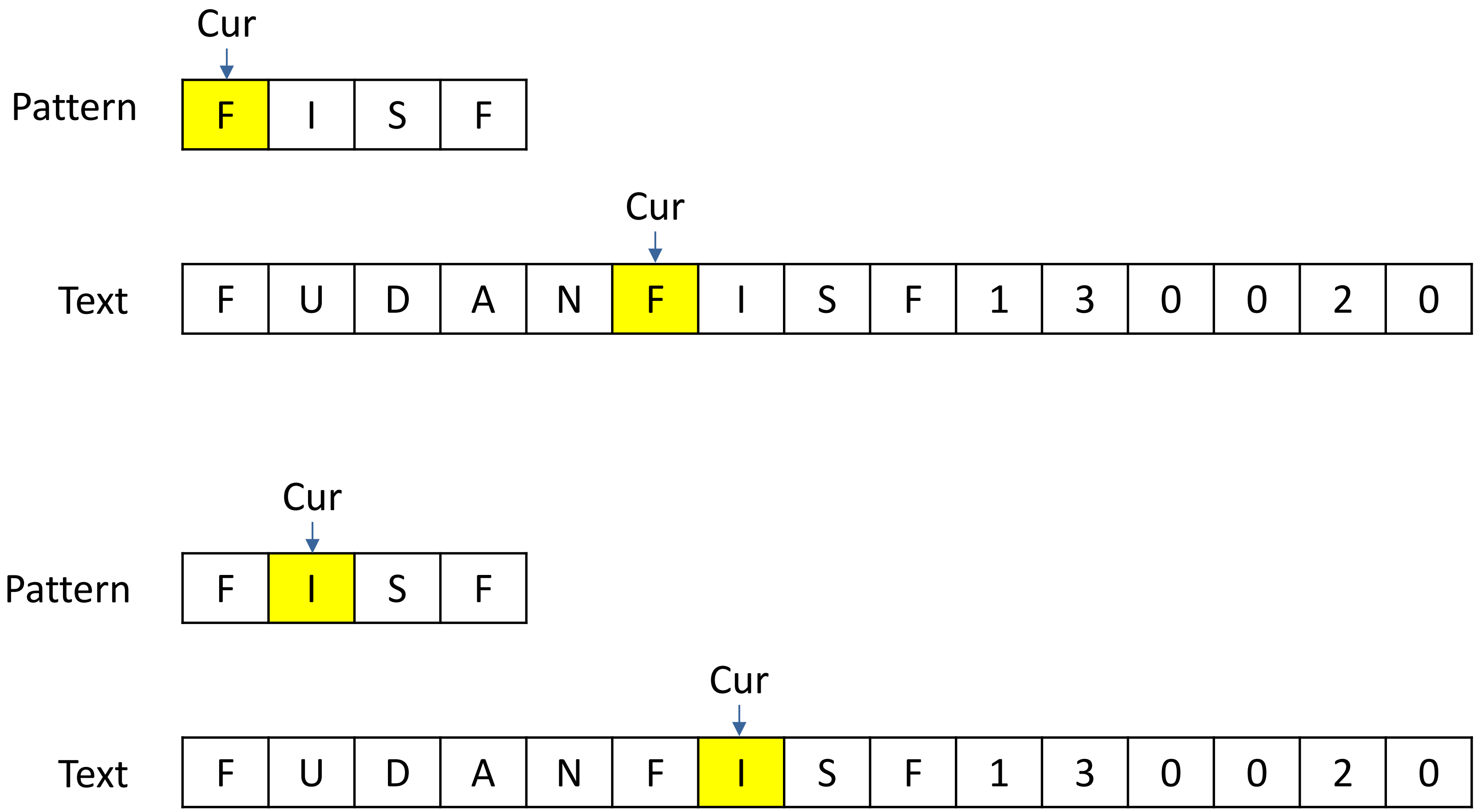
Cur

| Pattern | | | | |
|---|---|---|---|---|
| | F | I | S | F |

Cur

| Text | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | F | U | D | A | N | F | I | S | F | 1 | 3 | 0 | 0 | 2 | 0 |

# Naive Approach



| Pattern | F | I | S | F |
|---|---|---|---|---|

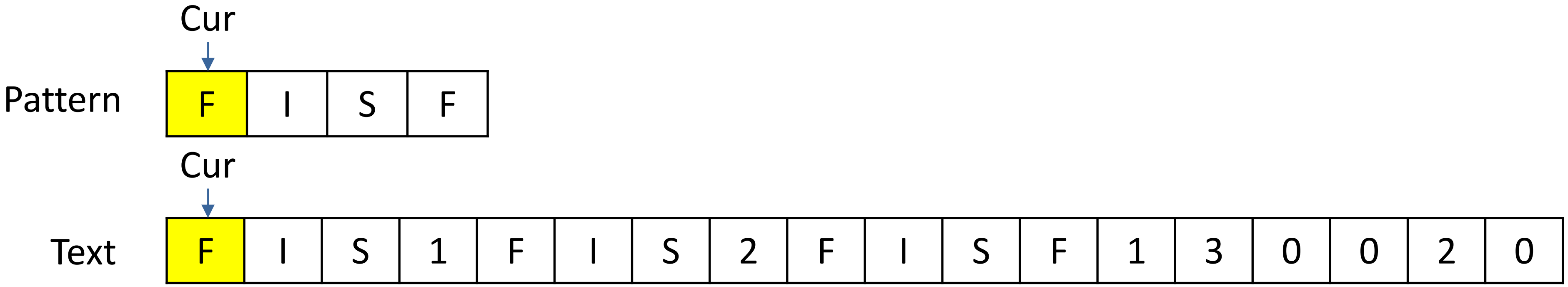
| Text | F | U | D | A | N | F | I | S | F | 1 | 3 | 0 | 0 | 2 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Pattern | F | I | S | F |
|---|---|---|---|---|

| Text | F | U | D | A | N | F | I | S | F | 1 | 3 | 0 | 0 | 2 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Complexity: $O(l_1 * l_2)$

# Worst-case of The Naive Approach



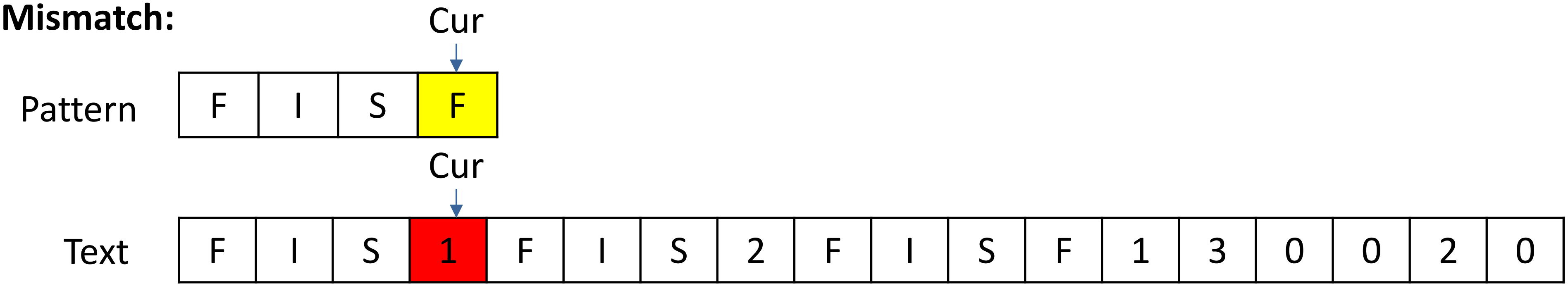
Cur

| Pattern | F | I | S | F |
|---|---|---|---|---|

Cur

| Text | F | I | S | 1 | F | I | S | 2 | F | I | S | F | 1 | 3 | 0 | 0 | 2 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Mismatch:**



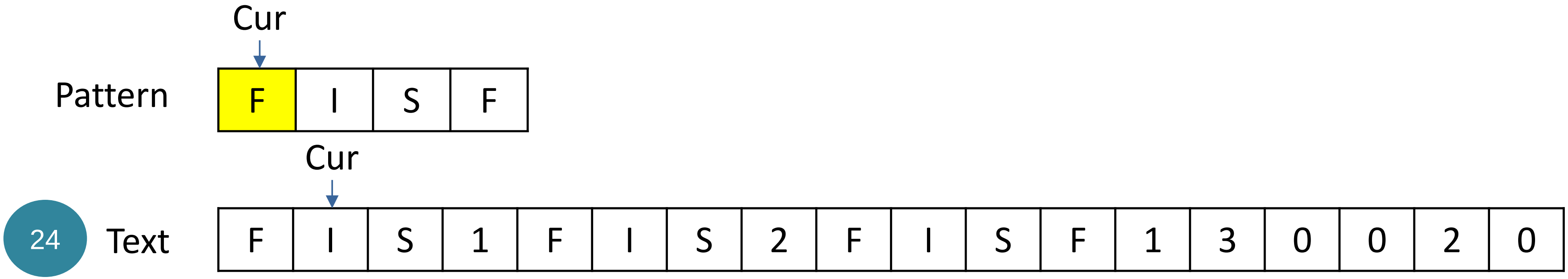
Cur

| Pattern | F | I | S | F |
|---|---|---|---|---|

Cur

| Text | F | I | S | 1 | F | I | S | 2 | F | I | S | F | 1 | 3 | 0 | 0 | 2 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Restart from the next position of the text**



Cur

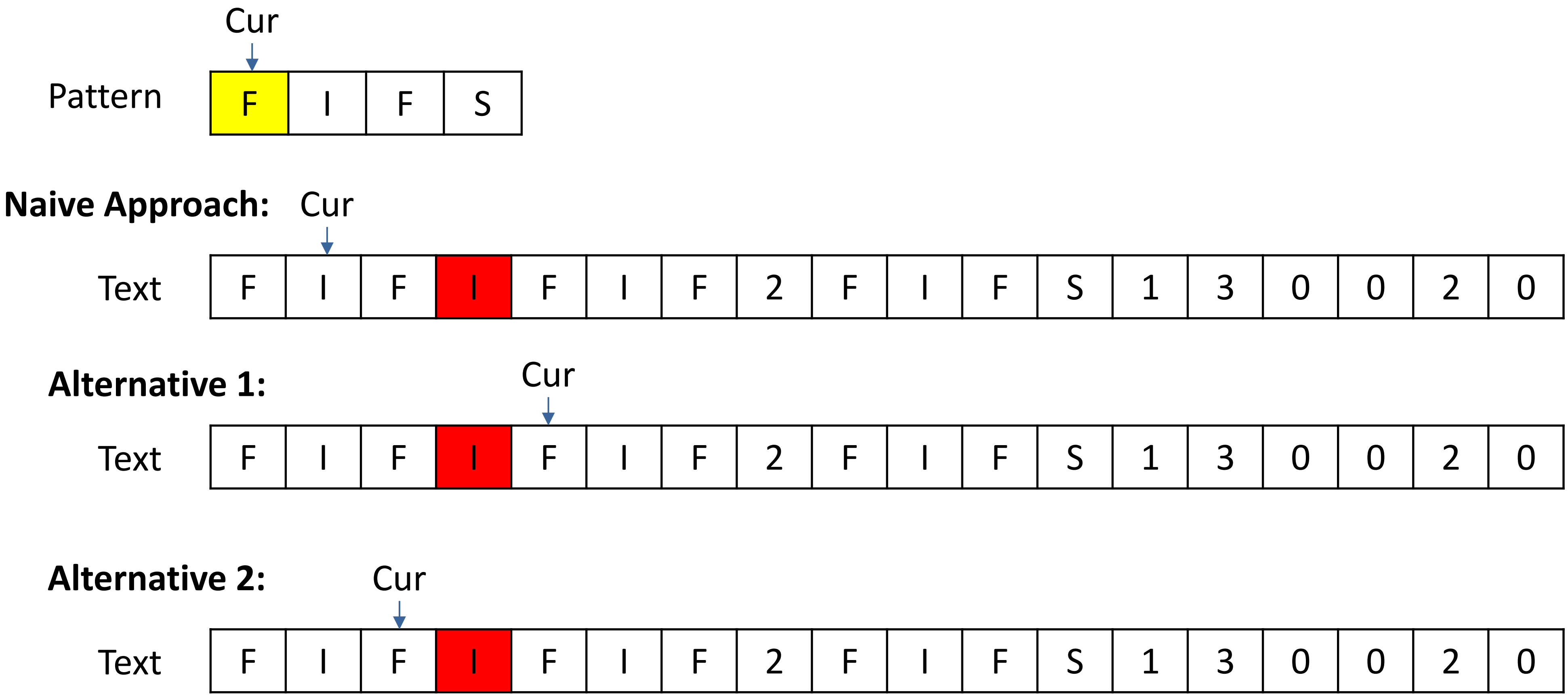
| Pattern | F | I | S | F |
|---|---|---|---|---|

Cur

| Text | F | I | S | 1 | F | I | S | 2 | F | I | S | F | 1 | 3 | 0 | 0 | 2 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# KMP Algorithm

- Keep records of the prefix appeared in the already matched substring.
- Continue from the next position if such prefix does not exist.



|  |  |  |  |  |
|---|---|---|---|---|
|  | Cur |  |  |  |
| Pattern | F | I | F | S |

**Naive Approach:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | Cur |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Text | F | I | F | I | F | I | F | 2 | F | I | F | S | 1 | 3 | 0 | 0 | 2 | 0 |

**Alternative 1:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  | Cur |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Text | F | I | F | I | F | I | F | 2 | F | I | F | S | 1 | 3 | 0 | 0 | 2 | 0 |

**Alternative 2:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  | Cur |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Text | F | I | F | I | F | I | F | 2 | F | I | F | S | 1 | 3 | 0 | 0 | 2 | 0 |

# KMP via LPS (Longest Prefix Suffix)

- The LPS of a pattern at position $i$ indicates the length of the longest proper prefix of the pattern (up to $i$), which is also a suffix.
- Move several steps to the left based on the LPS value.

| | | | |
|---|---|---|---|
| F | I | F | S |

- LSP(0) = 0 because the “F” does not have a prefix nor suffix.
- LSP(1) = 0 because the “FI” has a prefix “F” and a suffix “I”, which do not match.
- LSP(2) = 1 because the “FIF” has a matched prefix and suffix “F”.
- LSP(3) = 0

# More Problems of String Matching

- How to find the longest common substring?
  - A substring consists of contiguous characters
  - For example: the longest common substring of "fundamental" and "fudanmental" is "mental"
- How to find the longest common subsequence?
  - A subsequence consists of noncontinuous characters
  - For example: the longest common subsequence of "fundamental" and "fudanmental" is "fudamental"

# 3. In-class Practice

# Option 1: Trading Software

- Implement the following features for your trading software.
    - Display stocks in ascending or descending order by name, price, volume, or other criteria.
    - Search for a stock by its ticker symbol or company name.
- Discuss the sorting and searching algorithms employed in your software.

# Option 2: Sorting

- Design experiments to compare the performance of selection sort, merge sort, and quick sort with 1000, and 10000 elements.