# Lecture 10: Elementary Sorts

## BT 3051 – Data Structures and Algorithms for Biology

Karthik Raman
Department of Biotechnology
Bhupat and Jyoti Mehta School of Biosciences
Indian Institute of Technology Madras

# Introduction

# Introduction

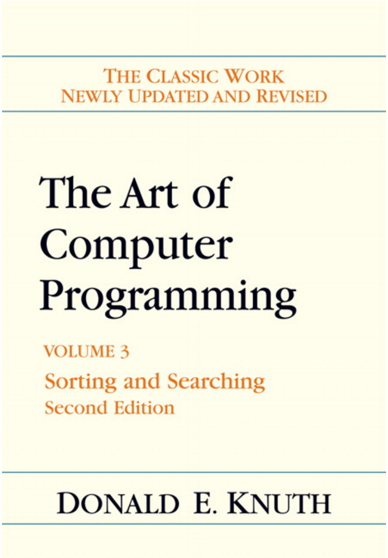
- Sorting is a very critical operation
- Perhaps the first step to searching

# Introduction

- Sorting is a very critical operation
- Perhaps the first step to searching

# Introduction

- Sorting is a very critical operation
- Perhaps the first step to searching



# Comparison-based Sorting Algorithms

http://www.sorting-algorithms.com/

The ideal sorting algorithm would have the following properties:

- Stable: Equal keys are not reordered
- Operates in place, requiring $O(1)$ extra space
- Worst-case $O(n \lg n)$ key comparisons
- Worst-case $O(n)$ swaps
- Adaptive: Speeds up to $O(n)$ when data is nearly sorted or when there are few unique keys

# Comparison-based Sorting Algorithms

## http://www.sorting-algorithms.com/

The ideal sorting algorithm would have the following properties:

- Stable: Equal keys are not reordered
- Operates in place, requiring $O(1)$ extra space
- Worst-case $O(n \lg n)$ key comparisons
- Worst-case $O(n)$ swaps
- Adaptive: Speeds up to $O(n)$ when data is nearly sorted or when there are few unique keys

# Comparison-based Sorting Algorithms

http://www.sorting-algorithms.com/

The ideal sorting algorithm would have the following properties:

- Stable: Equal keys are not reordered
- Operates in place, requiring $O(1)$ extra space
- Worst-case $O(n\ lg\ n)$ key comparisons
- Worst-case $O(n)$ swaps
- Adaptive: Speeds up to $O(n)$ when data is nearly sorted or when there are few unique keys

# Comparison-based Sorting Algorithms

http://www.sorting-algorithms.com/

The ideal sorting algorithm would have the following properties:

- Stable: Equal keys are not reordered
- Operates in place, requiring $O(1)$ extra space
- Worst-case $O(n \lg n)$ key comparisons
- Worst-case $O(n)$ swaps
- Adaptive: Speeds up to $O(n)$ when data is nearly sorted or when there are few unique keys

# Comparison-based Sorting Algorithms

http://www.sorting-algorithms.com/

The ideal sorting algorithm would have the following properties:

- Stable: Equal keys are not reordered
- Operates in place, requiring $O(1)$ extra space
- Worst-case $O(n \lg n)$ key comparisons
- Worst-case $O(n)$ swaps
- Adaptive: Speeds up to $O(n)$ when data is nearly sorted or when there are few unique keys

# Comparison-based Sorting Algorithms

http://www.sorting-algorithms.com/

The ideal sorting algorithm would have the following properties:

- Stable: Equal keys are not reordered
- Operates in place, requiring $O(1)$ extra space
- Worst-case $O(n \lg n)$ key comparisons
- Worst-case $O(n)$ swaps
- Adaptive: Speeds up to $O(n)$ when data is nearly sorted or when there are few unique keys

There is no algorithm that has all of these properties!

# Comparison-based Sorting Algorithms

http://www.sorting-algorithms.com/

The ideal sorting algorithm would have the following properties:

- Stable: Equal keys are not reordered
- Operates in place, requiring $O(1)$ extra space
- Worst-case $O(n \lg n)$ key comparisons
- Worst-case $O(n)$ swaps
- Adaptive: Speeds up to $O(n)$ when data is nearly sorted or when there are few unique keys

There is no algorithm that has all of these properties!

Therefore, choice of sorting algorithm depends on application!

# Introduction to Sorting: Video

https://www.youtube.com/watch?v=cVMKXKoGu_Y, courtesy of CS unplugged

# Elementary Sorts

# Selection Sort

```
def SelectionSort(a):
    n = len(a)
    for i in range(n):
        k = i
        for j in range(i+1,n):
            if (a[j]<a[k]):
                k = j
        a[i],a[k] = a[k], a[i]
    return a
```

## Properties

- Not stable
- $O(1)$ extra space
- $\Theta(n^2)$ comparisons
- $\Theta(n)$ swaps
- Not adaptive

# Selection Sort

```
def SelectionSort(a):
    n = len(a)
    for i in range(n):
        k = i
        for j in range(i+1,n):
            if (a[j]<a[k]):
                k = j
        a[i],a[k] = a[k], a[i]
    return a
```

## Properties

- Not stable
- $O(1)$ extra space
- $\Theta(n^2)$ comparisons
- $\Theta(n)$ swaps
- Not adaptive

# Selection Sort

```
def SelectionSort(a):
    n = len(a)
    for i in range(n):
        k = i
        for j in range(i+1,n):
            if (a[j]<a[k]):
                k = j
        a[i],a[k] = a[k], a[i]
    return a
```

## Properties

- Not stable
- $O(1)$ extra space
- $\Theta(n^2)$ comparisons
- $\Theta(n)$ swaps
- Not adaptive

# Selection Sort

```python
def SelectionSort(a):
    n = len(a)
    for i in range(n):
        k = i
        for j in range(i+1,n):
            if (a[j]<a[k]):
                k = j
        a[i],a[k] = a[k], a[i]
    return a
```

## Properties

- Not stable
- $O(1)$ extra space
- $\Theta(n^2)$ comparisons
- $\Theta(n)$ swaps
- Not adaptive

# Selection Sort

```
def SelectionSort(a):
    n = len(a)
    for i in range(n):
        k = i
        for j in range(i+1,n):
            if (a[j]<a[k]):
                k = j
        a[i],a[k] = a[k], a[i]
    return a
```

## Properties

- Not stable
- $O(1)$ extra space
- $\Theta(n^2)$ comparisons
- $\Theta(n)$ swaps
- Not adaptive

# Selection Sort

- Selection sort has the property of *minimising the number of swaps*
- May be useful in applications where swap cost is high!

# Selection Sort

- Selection sort has the property of *minimising the number of swaps*
- May be useful in applications where swap cost is high!

# Bubble Sort

```
def BubbleSort(a):
    n = len(a)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
    return a
```

## Properties

- Stable
- $O(1)$ extra space
- $O(n^2)$ comparisons and swaps
- Is it adaptive?

# Bubble Sort

```
def BubbleSort(a):
    n = len(a)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
    return a
```

## Properties

- Stable
- $O(1)$ extra space
- $O(n^2)$ comparisons and swaps
- Is it adaptive?

# Bubble Sort

```
def BubbleSort(a):
    n = len(a)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
    return a
```

## Properties

- Stable
- $O(1)$ extra space
- $O(n^2)$ comparisons and swaps
- Is it adaptive?

# Bubble Sort

```python
def BubbleSort(a):
    n = len(a)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
    return a
```

## Properties

- Stable
- $O(1)$ extra space
- $O(n^2)$ comparisons and swaps
- Is it adaptive?

# Bubble Sort: Making it Adaptive

```
def BubbleSort(a):
    n = len(a)
    for i in range(n - 1):
        swapped = False
        for j in range(n - 1 - i):
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
                swapped = True
        if not swapped:
            break
    return a
```

How did we make it adaptive?