

# CS280-Data Structures

## Introductory Sorting Part 1

# Recap

- Algorithm complexity analysis
- Big  $O$  notation
  - Asymptotic analysis
  - Common growth rates
- Program complexity derivation
  - Sequences
  - Conditionals
  - Loops
  - Function calls

# Sorting Algorithm

- **Input:** A sequence of  $n$  numbers  $\{a_1, a_2, \dots, a_n\}$
- **Output:** A permutation  $\{a_1', a_2', \dots, a_n'\}$  of the input sequence such that  $a_1' \leq a_2' \leq \dots \leq a_n'$  or  $a_1' \geq a_2' \geq \dots \geq a_n'$ 
  - Increasing order
  - Decreasing order

# Basics of Sorting

- $A = \{a_1, a_2, \dots, a_n\}$ ,  $n \geq 0$  elements from some universal set  $U$
- **Sorting** algorithms aim to rearrange  $A$  to produce  $A'$  where each  $a_i \in A'$ ,  $i=1,2,\dots,n$  are in order.

# In Order

- What does it mean by **in order**?

- ❖ **Numbers:**

- 1, 2, 8, 20,10000

- ❖ **Strings:**

- Alvin, Edward, Michael, Prabhu, Prasanna, Zhaoyan

- ❖ **struct Student {**

- int Age;**

- long Year;**

- float GPA;**

- long ID;**

- };**

Usually the **Keys** of Items are in numerical or alphabetical order.

# Usefulness in Practice

- Google search
  - **Relevance**: Similarity between the query and the webpage
  - **Importance**: PageRank
- Text search
- Image search
  - Content-based image retrieval
  - Semantic-based image retrieval
- Recommendations
- etc.

# Usefulness in Practice

digipen



Web

Images

Maps

News

Videos

More ▾

Search tools

About 725,000 results (0.26 seconds)

## [DigiPen Singapore: A Leading Game Development Education: Home](https://singapore.digipen.edu/)

<https://singapore.digipen.edu/> ▾

Offering bachelor's degrees in computer science, game design, and animation. Professional development courses and two-week programs for teens also ...  
[3 Google reviews](#) · [Write a review](#)



10 Central Exchange Green, 138649  
6577 1900

### [DigiPen](#)

DigiPen offers degree programs in Video Game Programming ...

### [Faculty Profiles](#)

Faculty Profiles. Expand/collapse left column. About · History ...

### [Admissions](#)

The Ministry of Education announced the formation of ...

[More results from digipen.edu »](#)

### [Continuing Education](#)

DigiPen Institute of Technology Singapore is a Continuing ...

### [About DigiPen](#)

DigiPen Institute of Technology is a dedicated, world-renowned ...

### [Ubisoft® Campus Program](#)

Singapore is home to DigiPen's first international campus. The ...

## [DigiPen Institute of Technology - Wikipedia, the free encyclopedia](https://en.wikipedia.org/wiki/DigiPen_Institute_of_Technology)

[en.wikipedia.org/wiki/DigiPen\\_Institute\\_of\\_Technology](https://en.wikipedia.org/wiki/DigiPen_Institute_of_Technology) ▾

DigiPen Institute of Technology is a private university located in Redmond, Washington. DigiPen has campuses in Redmond, Washington, in Singapore, and ...



[See photos](#)



## Digipen Institute of Technology Singapore

[Directions](#)

[Write a review](#)

**Address:** 10 Central Exchange Green, 138649

**Phone:** 6577 1900

### Reviews

[3 Google reviews](#)

**More reviews:** [learnpipe.com](https://learnpipe.com)

# Usefulness in Practice

- Google search
  - **Relevance**: Similarity between the query and the webpage
  - **Importance**: PageRank
- Text search
- Image search
  - Content-based image retrieval
  - Semantic-based image retrieval
- Recommendations
- etc.

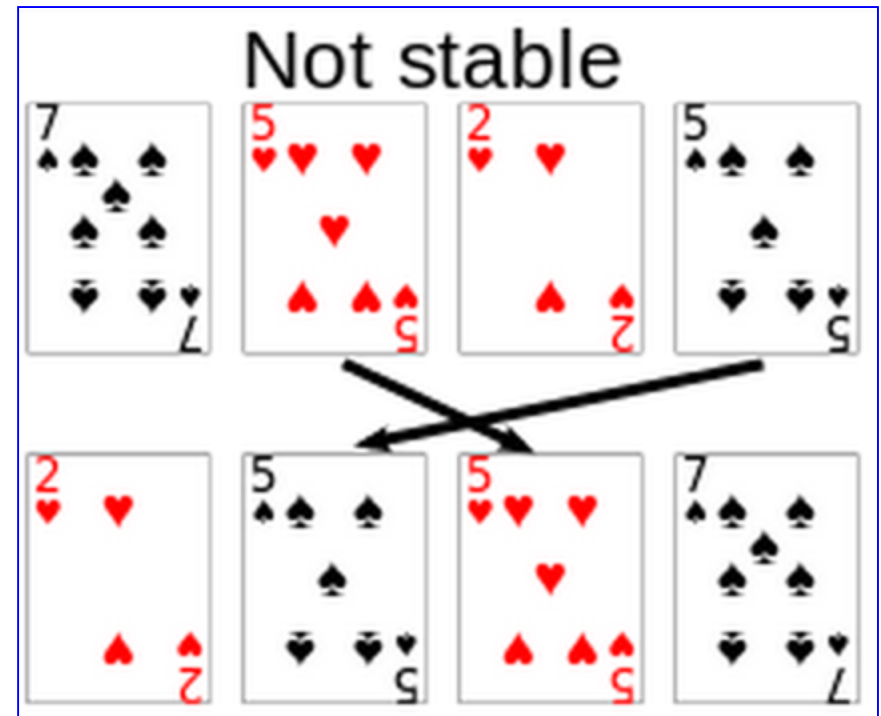
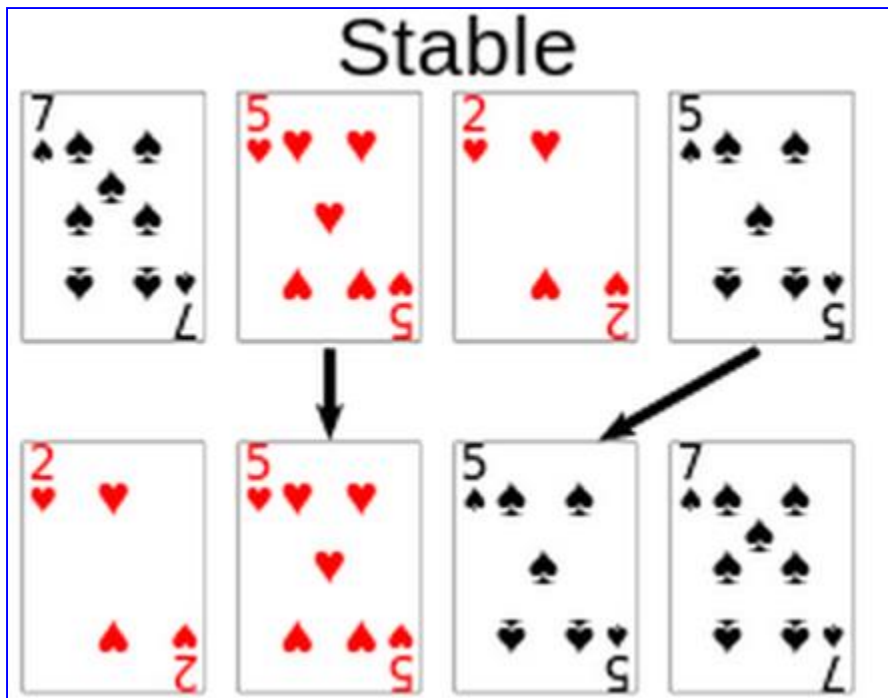


# Attributes of Sorting Algorithms

- Stable/unstable sorting algorithms
- Loop invariant
- In-place/Out-of-place algorithms
- Adaptive

# Stable Sorting Algorithm

- A sorting method is said to be **stable** if it preserves the relative order of items with equal values.



# Loop Invariant

- **Loop invariant** is a property that holds before (and after) each repetition

# Loop Invariant

- **Loop invariant** is a property that holds before (and after) each repetition

```
1. int max(int n, const int a[n]) {
2.     int m = a[0];
3.     // m equals the maximum value in a[0...0]
4.     int i = 1;
5.     while (i != n) {
6.         // m equals the maximum value in a[0...i-1]
7.         if (m < a[i])
8.             m = a[i];
9.         // m equals the maximum value in a[0...i]
10.        ++i;
11.        // m equals the maximum value in a[0...i-1]
12.    }
13.    // m equals the maximum value in a[0...i-1], and i==n
14.    return m;
15.}
```

# In-place/Out-of-place algorithms

```
void f1(int a[],int n){  
    int* b = new int[n];  
    int i;  
    for(i=0;i<n;++i)  
        b[n-i-1] = a[i];  
    for(i=0;i<n;++i)  
        a[i]=b[i];  
    delete [] b;  
}
```

```
void f2(int a[], int n){  
    int i;  
    int m=n/2;  
    for(i=0;i<m;++i)  
        Swap(a[i],a[n-i-1]);  
}
```

# Adaptive

- It takes advantage of existing order in its input
- It benefits from the pre-sortedness in the input sequence
- $A_1[9] = \{9, 8, 7, 6, 5, 4, 3, 2, 1\}$
- $A_2[9] = \{2, 1, 3, 4, 5, 6, 7, 8, 9\}$

# Sorting Algorithms

- Bubble sort
- Selection sort
- Insertion sort
- Merge sort
- Quick sort
- Lower bounds for sorting
- Counting sort

# Sorting Algorithms

- Bubble sort
- Selection sort
- Insertion sort



# Sorting Algorithms

- Bubble sort
- Selection Sort
- Insertion Sort

# Bubble Sort

- Exchange Sort: switch a pair of neighboring elements
- Main idea: Keep **bubbling** the largest element to the end.
  1. Get the current  $i$  and  $i+1$  elements.
  2. Swap them into the correct order.
  3. Start from front again when reach the end.

# Example

- $A[9]=\{7, 4, 1, 3, 8, 6, 9, 5, 2\}$
- objective:  $A'[9]=\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

# Bubble Sort

```
void BubbleSort(int a[], int N){  
    for (int i = 0; i < N - 1; ++i)  
        for (int j = 0; j < N - i - 1; ++j)  
            if (a[j] > a[j + 1])  
                Swap(a[j], a[j + 1]);  
}
```

```
void Swap(int &a, int &b){  
    int temp = a;  
    a = b;  
    b = temp;  
}
```

Stable?

Loop invariant?

In-place?

Adaptive?

# Bubble Sort

```
void BubbleSort(int a[], int N){  
    for (int i = 0; i < N - 1; ++i)  
        for (int j = 0; j < N - i - 1; ++j)  
            if (a[j] > a[j + 1])  
                Swap(a[j], a[j + 1]);  
}
```

```
void Swap(int &a, int &b){  
    int temp = a;  
    a = b;  
    b = temp;  
}
```

Stable: Y

Loop invariant: Y

In-place: Y

Adaptive: N

# Bubble Sort

```
void BubbleSort(int a[], int N){  
    for (int i = 0; i < N - 1; ++i)  
        for (int j = 0; j < N - i - 1; ++j)  
            if (a[j] > a[j + 1])  
                Swap(a[j], a[j + 1]);  
}
```

```
void Swap(int &a, int &b){  
    int temp = a;  
    a = b;  
    b = temp;  
}
```

**Best** case complexity: ?

**Worst** case complexity: ?

**Question:** How can we improve the algorithm?  
(hint: consider the best case)

# Bubble Sort

```
void BubbleSort(int a[], int N){  
    for (int i = 0; i < N - 1; ++i)  
        for (int j = 0; j < N - i - 1; ++j)  
            if (a[j] > a[j + 1])  
                Swap(a[j], a[j + 1]);  
}
```

```
void Swap(int &a, int &b){  
    int temp = a;  
    a = b;  
    b = temp;  
}
```

**Best** case complexity:  $O(N)$   
**Worst** case complexity:  $O(N^2)$

**Question:** How can we improve the algorithm?  
(hint: consider the best case)

# Adaptive Bubble Sort

- If the algorithm stops when no swaps, then best case is  $O(N)$

```
void BubbleSort(int a[], int N){  
    for (int i = 0; i < N - 1; ++i){  
        bool swap = false;  
        for (int j = 0; j < N - i - 1; ++j)  
            if (a[j] > a[j + 1]){  
                swap=true;  
                Swap(a[j], a[j + 1]);  
            }  
        }  
        if(!swap)  
            return;  
    }  
}
```



# Sorting Algorithms

- Bubble sort
- Selection sort
- Insertion sort

# Selection Sort

- Main idea: Increase the sorted sequence by selecting the smallest element from the unsorted.
1. Select the smallest element from the unsorted side
  2. Append it at the end of the sorted side (or the first position of the unsorted)

Find the Smallest, then the second smallest from the rest...

# Selection Sort Example

- Input:  $A[9]=\{7, 4, 1, 3, 8, 6, 9, 5, 2\}$
- Output:  $A'[9]=\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

# Selection Sort

```
void SelectionSort(int a[], int N){  
    for (int i = 0; i < N-1; ++i){  
        int min = i;  
        int j;  
  
        for (j = i + 1; j < N; ++j)  
            if (a[j] < a[min])  
                min = j;  
  
        Swap(a[min], a[i]);  
    }  
}
```

Stable?

Loop invariant?

In-place?

Adaptive?

# Selection Sort

```
void SelectionSort(int a[], int N){  
    for (int i = 0; i < N-1; ++i){  
        int min = i;  
        int j;  
  
        for (j = i + 1; j < N; ++j)  
            if (a[j] < a[min])  
                min = j;  
  
        Swap(a[min], a[i]);  
    }  
}
```

Stable: N

Loop invariant: Y

In-place: Y

Adaptive: N

# Selection Sort

```
void SelectionSort(int a[], int N){
```

```
    for (int i = 0; i < N-1; ++i){
```

```
        int min = i;
```

**Best** case complexity:?

```
        int j;
```

**Worst** case complexity: ?

```
        for (j = i + 1; j < N; ++j)
```

```
            if (a[j] < a[min])
```

```
                min = j;
```

```
        Swap(a[min], a[i]);
```

```
    }
```

```
}
```

**Question:** How could you modify this algorithm to sort the array in half the time? (Not really half the time, but half the number of passes.)

# Selection Sort

```
void SelectionSort(int a[], int N){
```

```
    for (int i = 0; i < N-1; ++i){
```

```
        int min = i;
```

**Best** case complexity:  $O(N^2)$

```
        int j;
```

**Worst** case complexity:  $O(N^2)$

```
        for (j = i + 1; j < N; ++j)
```

```
            if (a[j] < a[min])
```

```
                min = j;
```

```
        Swap(a[min], a[i]);
```

```
    }
```

```
}
```

**Question:** How could you modify this algorithm to sort the array in half the time? (Not really half the time, but half the number of passes.)

# Improved Selection Sort

- Find the minimum and maximum at the same time!

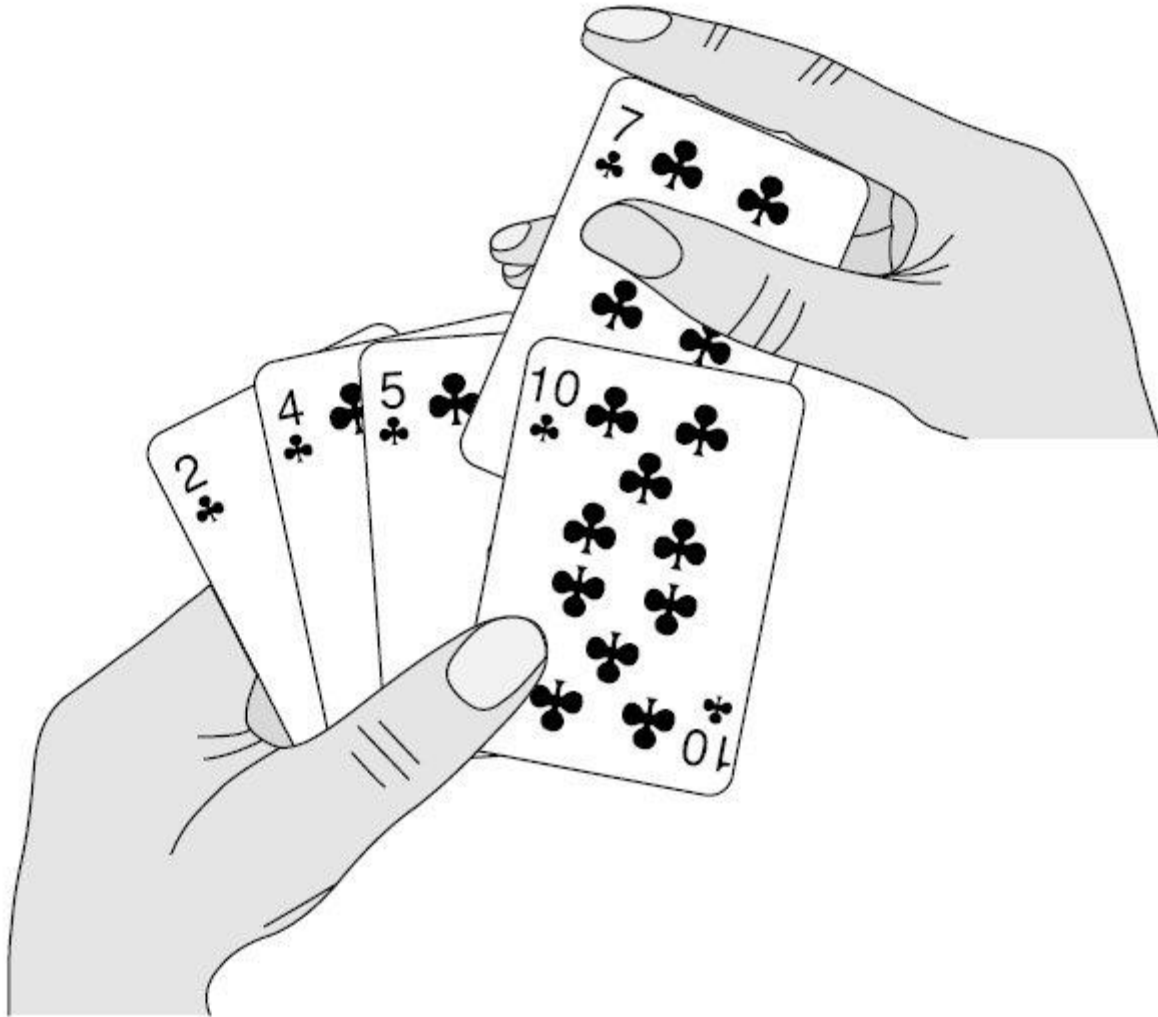
```
void BiSelectionSort(int a[], int N){
    for (int i = 0; i < N/2; ++i){
        int min = i;
        int max = N-i-1;
        int j;
        for (j = i; j < N-i; ++j){
            if (a[j] < a[min])
                min = j;
            if (a[j] > a[max])
                max = j;
        }
        Swap(a[min], a[i]);
        if (i==max)
            Swap(a[min], a[N-i-1]);
        else
            Swap(a[max], a[N-i-1]);
    }
}
```



# Sorting Algorithms

- Bubble Sort
- Selection Sort
- Insertion Sort

# Insertion Sort



# Insertion Sort

- Main idea: Keep inserting next element into the sorted sequence.
  1. Get the 1<sup>st</sup> element from the unsorted side
  2. Find the correct position in the sorted side
  3. Shift elements to make room

# Insertion Sort Example

- $A[9] = \{7, 4, 1, 3, 8, 6, 9, 5, 2\}$
- Objective:  $A'[9] = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

# Insertion Sort

```
void InsertionSort(int a[], int N){  
    for (int i = 1; i < N; ++i){  
        int j = i;  
        int current = a[i];  
  
        while ((j > 0) && (a[j-1] > current)){  
            a[j] = a[j-1];  
            --j;  
        }  
        a[j] = current;  
    }  
}
```

Stable?

Loop invariant?

In-place?

Adaptive?

# Insertion Sort

```
void InsertionSort(int a[], int N){  
    for (int i = 1; i < N; ++i){  
        int j = i;  
        int current = a[i]; // the item to be inserted  
  
        while ((j > 0) && (a[j-1] > current)){  
            a[j] = a[j-1];  
            --j;  
        } // find the position for insertion: j  
        a[j] = current;  
    }  
}
```

Stable: Y

Loop invariant: Y

In-place: Y

Adaptive: Y

# Insertion Sort

```
void InsertionSort(int a[], int N){  
    for (int i = 1; i < N; ++i){  
        int j = i;  
        int current = a[i];  
  
        while ((j > 0) && (a[j-1] > current)){  
            a[j] = a[j-1];  
            --j;  
        }  
        a[j] = current;  
    }  
}
```

Best case complexity?

Worst case complexity?

# Insertion Sort

```
void InsertionSort(int a[], int N){  
    for (int i = 1; i < N; ++i){  
        int j = i;  
        int current = a[i];  
  
        while ((j > 0) && (a[j-1] > current)){  
            a[j] = a[j-1];  
            --j;  
        }  
        a[j] = current;  
    }  
}
```

**Best** case complexity:  $O(N)$

**Worst** case complexity:  $O(N^2)$



# Summary

- Elementary Sorting Methods
  - Bubble sort
  - Selection sort
  - Insertion sort

# Comparison

- Bubble sort and variants
  - rarely used in practice
  - commonly found in teaching and theoretical discussions.
- Insertion sort
  - generally faster than selection sort in practice, due to fewer comparisons and good performance on almost-sorted data
  - preferred in practice.
  - is relatively efficient for small lists and mostly sorted lists.
- Selection sort
  - Does no more than  $n$  swaps, and thus is useful where swapping is very expensive.
  - uses fewer writes, and thus is used when write performance is a limiting factor.