12: Sorting Fundamentals
Lecture 12: Sorting Fundamentals

- Introduction
- Selection sort
- Insertion sort
Why study sorting?

- It’s more common than you think: e.g., sorting flights by price, contacts by last name, files by size, emails by day sent, neighborhoods by zipcode, etc.

- Good example of how to compare the performance of different algorithms for the same problem.

- Some sorting algorithms relate to data structures.

- Sorting your data will often be a good starting point when solving other problems (keep that in mind for interviews).
Definitions

- **Sorting**: the process of arranging \( n \) items of a collection in non-decreasing order (e.g., numerically or alphabetically).

- **Key**: assuming that an item consists of multiple components, the key is the property based on which we sort items.

- Examples: items could be books and potential keys are the title or the author which can be sorted alphabetically or the ISBN which can be sorted numerically.
Total order

- Sorting is well defined if and only if there is total order.

- **Total order:** a binary relation $\leq$ on a set $C$ that satisfies the following statements for all $v$, $w$, and $x$ in $C$:
  
  - **Connexity:** $v \leq w$ or $w \leq v$.
  
  - **Transitivity:** for all $v$, $w$, $x$, if $v \leq w$ and $w \leq x$ then $v \leq x$.
  
  - **Antisymmetry:** if both $v \leq w$ and $w \leq v$, then $v = w$. 
How many different algorithms for sorting can there be?

- Adaptive heapsort
- Bitonic sorter
- Block sort
- Bubble sort
- Bucket sort
- Cascade mergesort
- Cocktail sort
- Comb sort
- Flashsort
- Gnome sort
- Heapsort
- Insertion sort
- Mergesort
- Odd-even sort
- Pancake sort
- Quicksort
- Radixsort
- Selection sort
- Shell sort
- Spaghetti sort
- Treesort
- ...

INTRODUCTION
Rules of the game - Comparing

- We will be sorting arrays of $n$ items, where each item contains a key. In Java, objects are responsible in telling us how to naturally compare their keys.

- Let’s say we want to sort an array of objects of type T.

- Our class T should implement the Comparable interface (more on this in a few lectures). We will need to implement the compareTo method to satisfy a total order.

[https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html)
Rules of the game - Comparing

- `public int compareTo(T that)`
- Implement it so that `v.compareTo(w)`:  
  - Returns `>0` if `v` is greater than `w`.
  - Returns `<0` if `v` is smaller than `w`.
  - Returns `0` if `v` is equal to `w`.
- Java classes such as `Integer`, `Double`, `String`, `File` all implement `Comparable`.

[https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html)
Two useful abstractions

- **We will refer to data only through comparisons and exchanges.**

  ```java
  private static boolean less(Comparable v, Comparable w) {
    return v.compareTo(w) < 0;
  }
  ```

- **Less**: Is v less than w?

  ```java
  private static void exch(Comparable[] a, int i, int j) {
    Comparable swap = a[i];
    a[i]=a[j];
    a[j]=swap;
  }
  ```

- **Exchange**: swap item in array a[] at index i with the one at index j.
Rules of the game - Cost model

- **Sorting cost model**: we count *comparisons* and *exchanges*. If a sorting algorithm does not use exchanges, we count *array accesses*.

- There are other types of sorting algorithms where they are not based on comparisons (e.g., radixsort). We will not see these in CS62 but stay tuned for CS140.
Rules of the game - Memory usage

- Extra memory: often as important as running time. Sorting algorithms are divided into two categories:
  - *In place*: use constant or logarithmic extra memory, beyond the memory needed to store the items to be sorted.
  - *Not in place*: use linear auxiliary memory.
Rules of the game - Stability

- **Stable**: sorting algorithms that sort repeated elements in the same order that they appear in the input.

---

Lecture 12: Sorting Fundamentals

- Introduction
- Selection sort
- Insertion sort
Selection sort

1. Divide the array in two parts: a **sorted subarray** on the left and an **unsorted** on the right.

2. Repeat:
   - Find the smallest element in the unsorted subarray.
   - Exchange it with the leftmost unsorted element.
   - Move subarray boundaries one element to the right.
Selection sort

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.
Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.
Selection sort

Repeat:
- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.
Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.
Selection sort

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.
Selection sort

Repeat:

- Find the smallest element in the unsorted subarray.
- Exchange it with the leftmost unsorted element.
- Move subarray boundaries one element to the right.
Selection sort

1 3 5 26 36 38 44 47

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.
Selection sort

| 1 | 3 | 5 | 26 | 36 | 38 | 44 | 47 |

- Repeat:
  - Find the smallest element in the unsorted subarray.
  - Exchange it with the leftmost unsorted element.
  - Move subarray boundaries one element to the right.
Selection sort

public static void sort(Comparable[] a) {

}
Selection sort

```java
public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++) {
        int min = i;
        for (int j = i+1; j < n; j++) {
            if (less(a[j], a[min]))
                min = j;
        }
        exch(a, i, min);
    }
}
```

- **Invariants**: At the end of each iteration $i$:
  - the array $a$ is sorted in ascending order for the first $i+1$ elements $a[0...i]$
  - no entry in $a[i+1...n-1]$ is smaller than any entry in $a[0...i]$
Selection sort: mathematical analysis for worst-case

```java
public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++) {
        int min = i;
        for (int j = i+1; j < n; j++) {
            if (less(a[j], a[min]))
                min = j;
        }
        exch(a, i, min);
    }
}
```

- **Comparisons**: $1 + 2 + \ldots + (n - 2) + (n - 1)\sim n^2/2$, that is $O(n^2)$.
- **Exchanges**: $n$ or $O(n)$, making it useful when exchanges are expensive.
- **Running time** is quadratic, even if input is sorted.
- **In-place**, requires almost no additional memory.
- **Not stable**
Practice Time

- Using selection sort, sort the array with elements \([12, 10, 16, 11, 9, 7]\).
- Visualize your work for every iteration of the algorithm.
## Selection Sort

### Answer

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
<th>6th</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>10</td>
<td>16</td>
<td>11</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>10</td>
<td>16</td>
<td>11</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>16</td>
<td>11</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>

Lecture 12: Sorting Fundamentals

- Introduction
- Selection sort
- Insertion sort
Insertion sort

Keep a partially sorted subarray on the left and an unsorted subarray on the right.

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

 Repeat:

  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Insertion sort

- **Repeat:**
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
**INSERTION SORT**

Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
INSERTION SORT

Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

| 3 | 5 | 38 | 44 | 47 | 1 | 36 | 26 |

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:
- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Insertion sort

```
3 1 5 38 44 47 36 26
```

- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Insertion sort

- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Insertion sort

Repeat:

▸ Examine the next element in the unsorted subarray.

▸ Find the location it belongs within the sorted subarray and insert it there.

▸ Move subarray boundaries one element to the right.
**INSERTION SORT**

Insertion sort

![Sorted Array]

- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
**INSERTION SORT**

Insertion sort

| 1 | 3 | 5 | 38 | 36 | 44 | 47 | 26 |

- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

![Sorted array example](image)

- **Repeat:**
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Insertion sort

Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
**INSERTION SORT**

Insertion sort

1  3  5  36  38  26  44  47

- **Repeat:**
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
**INSERTION SORT**

**Insertion sort**

- **Repeat:**
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Repeat:

- Examine the next element in the unsorted subarray.
- Find the location it belongs within the sorted subarray and insert it there.
- Move subarray boundaries one element to the right.
Insertion sort

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>26</th>
<th>36</th>
<th>38</th>
<th>44</th>
<th>47</th>
</tr>
</thead>
</table>

- Repeat:
  - Examine the next element in the unsorted subarray.
  - Find the location it belongs within the sorted subarray and insert it there.
  - Move subarray boundaries one element to the right.
2.1 Insertion Sort Demo
INSERTION SORT

In case you didn’t get this...

- https://www.youtube.com/watch?v=ROalU379l3U
Insertion sort

public static void sort(Comparable[] a) {
}

public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++) {
        for (int j = i; j > 0; j--) {
            if(less(a[j], a[j-1]))
                exch(a, j, j-1);
            else
                break;
        }
    }
}

Invariant: At the end of each iteration i:

- the array a is sorted in ascending order for the first i+1 elements a[0...i]
INSERTION SORT

Insertion sort: mathematical analysis for worst-case

```java
public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++) {
        for (int j = i; j > 0; j--) {
            if (less(a[j], a[j-1])) {
                exch(a, j, j-1);
            } else {
                break;
            }
        }
    }
}
```

- **Comparisons**: $0 + 1 + 2 + \ldots + (n-2) + (n-1)\approx n^2/2$, that is $O(n^2)$.

- **Exchanges**: $0 + 1 + 2 + \ldots + (n-2) + (n-1)\approx n^2/2$, that is $O(n^2)$.

- **Worst-case running time** is **quadratic**.

- **In-place**, requires almost no additional memory.

- **Stable**
### Insertion sort: average and best case

```java
public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++) {
        for (int j = i; j > 0; j--) {
            if (less(a[j], a[j-1])) {
                exch(a, j, j-1);
            } else {
                break;
            }
        }
    }
}
```

- **Average case**: quadratic for both comparisons and exchanges \(\sim n^2/4\) when sorting a randomly ordered array.

- **Best case**: \(n - 1\) comparisons and 0 exchanges for an already sorted array.

Practice Time

- Using insertion sort, sort the array with elements [12, 10, 16, 11, 9, 7].
- Visualize your work for every iteration of the algorithm.
Answer

INSERTION SORT

Insertion Sort

1st
12 10 16 11 9 7

2nd
10 12 16 11 9 7

3rd
10 12 16 11 9 7

4th
10 11 12 16 9 7

5th
9 10 11 12 16 7

last
7 9 10 11 12 16

Lecture 12: Sorting Fundamentals

- Introduction
- Selection sort
- Insertion sort
Readings:

- Textbook:
  - Chapter 2.1 (pages 244-262)
- Website:
  - Elementary sorts: [https://algs4.cs.princeton.edu/21elementary/](https://algs4.cs.princeton.edu/21elementary/)

Practice Problems:

- 2.1.1-2.1.8