Textbook implementation of queues

- **ResizingArrayQueue.java**: for implementation of queues with ArrayLists.
- **LinkedQueue.java**: for implementation of queues with singly linked lists.
Stacks, Queues, and Iterators

- Stacks
- Queues
- Applications
- Java Collections
- Iterators
APPLICATIONS

Stack applications

‣ Java Virtual Machine.
‣ Basic mechanisms in compilers, interpreters (see CS101).
‣ Back button in browser.
‣ Undo in word processor.
‣ Infix expression evaluation (Dijkstra’s algorithm with two stacks).
‣ Postfix expression evaluation.
1.3 DIJKSTRA'S 2-STACK DEMO
Postfix expression evaluation example (Calculator Assignment)

Example: \((52 - ((5 + 7) \times 4)) \Rightarrow 52 \ 5 \ 7 \ + \ 4 \ * \ -\)

```
Example: (52 - ((5 + 7) * 4) \Rightarrow 52 \ 5 \ 7 \ + \ 4 \ * \ -

\[
\begin{align*}
52 & \quad \text{push}(52) \\
52 & \quad \text{push}(5) \\
52 & \quad \text{push}(7) \\
12 & \quad \text{v1}=\text{pop}()=7 \\
52 & \quad \text{push}(v2+v1)=\text{push}(12) \\
52 & \quad \text{push}(4) \\
48 & \quad \text{v2}=\text{pop}()=5 \\
52 & \quad \text{push}(v2+v1)=\text{push}(48) \\
4 & \quad \text{v1}=\text{pop}()=4 \\
52 & \quad \text{push}(v2*v1)=48 \\
\end{align*}
\]
```

\[
\begin{align*}
52 & \quad \text{push}(v2*v1)=48 \\
4 & \quad \text{v2}=\text{pop}()=52 \\
4 & \quad \text{push}(v2-v1)=4 \\
\end{align*}
\]
Queue applications

- Spotify playlist.
- Data buffers (netflix, Hulu, etc.).
- Asynchronous data transfer (file I/O, sockets).
- Requests in shared resources (printers).
- Traffic analysis.
- Waiting times at calling center.
Lecture 9: Stacks, Queues, and Iterators

- Stacks
- Queues
- Applications
- Java Collections
- Iterators
The Java Collections Framework

Collections

Deque in Java Collections

- Do not use Stack.
- Queue is an interface...
- It’s recommended to use Deque instead.
- Double-ended queue (can add and remove from either end).

```java
java.util.Deque;
```

**public interface Deque<E> extends Queue<E>**

- You can choose between LinkedList and ArrayDeque implementations.

  ```java
  Deque deque = new ArrayDeque(); //preferable
  ```

https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html
Lecture 9: Stacks, Queues, and Iterators

- Stacks
- Queues
- Applications
- Java Collections
- Iterators
**Iterator Interface**

- Interface that allows us to traverse a collection one element at a time.

```java
public interface Iterator<E> {
    // returns true if the iteration has more elements
    // that is if next() would return an element instead of throwing an exception
    boolean hasNext();

    // returns the next element in the iteration
    // post: advances the iterator to the next value
    E next();

    // removes the last element that was returned by next
    default void remove(); // optional, better avoid it altogether
}
```

https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html
**Iterator Example**

```java
List<String> myList = new ArrayList<String>();
//... operations on myList

Iterator listIterator = myList.iterator();

while(listIterator.hasNext()){
    String elt = listIterator.next();
    System.out.println(elt);
}
```
Iterables Interface

- Interface that allows an object to be the target of a for-each loop:

```java
for(String elt: myList){
    System.out.println(elt);
}
```

```java
interface Iterable<E>{
    //returns an iterator over elements of type E
    Iterator<E> iterator();
}
```
How to make your data structures iterable?

1. Implement `Iterable` interface.

2. Make a private class that implements the `Iterator` interface.

3. Override `iterator()` method to return an instance of the private class.
Example: making ArrayList iterable

```java
public class ArrayList<Item> implements Iterable<Item> {
    //...
    public Iterator<Item> iterator() {
        return new ArrayListIterator();
    }

    private class ArrayListIterator implements Iterator<Item> {
        private int i = 0;

        public boolean hasNext() {
            return i < n;
        }

        public Item next() {
            return a[i++];
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```
Traversing ArrayList

All valid ways to traverse ArrayList and print its elements one by one.

```java
for (String elt: a1) {
    System.out.println(elt);
}

a1.forEach(System.out::println);
a1.forEach(elt->{System.out.println(elt);});

a1.iterator().forEachRemaining(System.out::println);
a1.iterator().forEachRemaining(elt->{System.out.println(elt);});
```
Lecture 9: Stacks, Queues, and Iterators

- Stacks
- Queues
- Applications
- Java Collections
- Iterators
Readings:

- Oracle’s guides:
  - Collections: [https://docs.oracle.com/javase/tutorial/collections/intro/index.html](https://docs.oracle.com/javase/tutorial/collections/intro/index.html)
  - Deque: [https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html)
  - Iterator: [https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html](https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html)
  - Iterable: [https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html)

- Textbook:
  - Chapter 1.3 (Page 126–157)

- Website:
  - Stacks and Queues: [https://algs4.cs.princeton.edu/13stacks/](https://algs4.cs.princeton.edu/13stacks/)

Practice Problems:

- 1.3.2–1.3.8, 1.3.32–1.3.33
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 $v, w, \text{ and } x$ 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
- Cascade mergesort
- Cocktail sort
- Comb sort
- Flashsort
- Gnome sort
- Heapsort
- Insertion sort
- Library sort
- Mergesort
- Odd-even sort
- Pancake sort
- Quicksort
- Radixsort
- Selection sort
- Shell sort
- Spaghetti sort
- Treesort
- ...
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
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.**
- **Less:** Is $v$ less than $w$?
  ```java
  private static boolean less(Comparable v, Comparable w) {
      return v.compareTo(w) < 0;
  }
  ```
- **Exchange:** swap item in array $a[]$ at index $i$ with the one at index $j$.
  ```java
  private static void exch(Comparable[] a, int i, int j) {
      Comparable swap = a[i];
      a[i]=a[j];
      a[j]=swap;
  }
  ```
Rules of the game - Cost model

- **Sorting cost model**: we count *comparissons* 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

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

### Selection sort

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

- **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.
Selection sort

```java
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**: $1 + 2 + \ldots + (n - 2) + (n - 1) \sim \frac{n^2}{2}$, 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**, think of the array [5_a, 3, 5_b, 1] which will end up as [1, 3, 5_b, 5_a].
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

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

Insertion sort

3  44  38  5  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.
- **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

Insertion sort

| 3 | 44 | 38 | 5 | 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.
**INSERTION SORT**

Insertion sort

```
3 44 38 5 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.
**Insertion Sort**

Insertion sort

| 3 | 38 | 44 | 5 | 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.
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.
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**

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.
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.
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.
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.
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.
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

- 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.
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.
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.
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.
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.
2.1 Insertion Sort Demo
In case you didn’t get this...

- [https://www.youtube.com/watch?v=ROalU379l3U](https://www.youtube.com/watch?v=ROalU379l3U)
Insertion sort

```java
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;
            }
        }
    }
}

▷ 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]
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: \(0 + 1 + 2 + \ldots + (n - 2) + (n - 1)\sim n^2/2\), that is \(O(n^2)\).

‣ Exchanges: \(0 + 1 + 2 + \ldots + (n - 2) + (n - 1)\sim n^2/2\), that is \(O(n^2)\).

‣ Worst-case running time is \textit{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])) {  // if a[j] is less than a[j-1]
                exch(a, j, j-1);  // exchange a[j] and a[j-1]
            } else {  // if a[j] is not less than a[j-1]
                break;  // break the inner loop
            }
        }
    }
}
```

- **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

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