Lecture 16-17: Sorting Basics

- Introduction
- Selection sort
- Insertion sort

Some slides adopted from Algorithms 4th Edition or COS226
Why study sorting?

- Analyzing sorting algorithms is a good example of how to compare the performance of different algorithms for the same problem.
- Many of the techniques used here can be found in different problems.
- Sorting your input will often be a good starting point when solving other problems.
Definitions

- **Sorting**: the process of arranging $n$ items of a collection in some logical order, typically numerically or alphabetically.
  - Examples: sorting students by names, purchases by price, neighborhoods by zipcode, flights by departure time, etc.

- **Key**: assuming that an item (also known as record, tuple, etc) consists of multiple components, sort 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.
Total order

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

- **Total order:** a binary relation $\leq$ that satisfies:
  
  - **Totality:** for all $v$ and $w$, if both $v \leq w$ or $w \leq v$ or both.
  
  - **Transitivity:** for all $v$ and $w$, if both $v \leq w$ or $w \leq x$ then $v \leq x$.

  - **Antisymmetry:** for all $v$ and $w$, if both $v \leq w$ and $w \leq v$ then $v = w$. 
INTRODUCTION

Rules of the game

- 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.
- This is achieved by making our class $T$ implement the Comparable interface (more on this in a few lectures). We will need to $\text{compareTo}$ to satisfy a total order:

\[
\text{public int compareTo(T that)}
\]
- Implement it so that $v.\text{compareTo}(w)$:
  - 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
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

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

- **Extra memory**: often as important as running time. Sorting algorithms are divided into two categories:
  - **In place**: use constant or logarithmic extra memory.
  - **Not in place**: use linear auxiliary memory.
Lecture 16-17: Sorting Basics

- Introduction
- Selection sort
- Insertion sort
Selection sort

- First, find the smallest item in the array.
- Exchange it with the first entry.
- Then, find the next smallest item.
- Exchange it with the second entry.
- Continue until the entire array is sorted.
Selection sort

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>min</th>
<th>a[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>S O R T E X A M P L E</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>A O R T E X A M P L E</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>A E R T O X S M P L E</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>A E E T O X S M P L R</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>A E E L O X S M P T R</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>A E E L M X S O P T R</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>A E E L M O S X P T R</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>A E E L M O P X S T R</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>A E E L M O P R S T X</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>A E E L M O P R S T X</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>A E E L M O P R S T X</td>
<td></td>
</tr>
</tbody>
</table>

Trace of selection sort (array contents just after each exchange)
2.1 Selection Sort Demo
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 $O(n)$
- **Running time is quadratic**, even if input is sorted.
- **In-place**, requires almost no additional memory.
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>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 16: Sorting Basics I

- Introduction
- Selection sort
- Insertion sort
Insertion sort

- Move from left to right through the array.
- Look at one element at a time and move it before the larger items on its left.
- Everything before the current time is sorted.
- Everything after the current time has not been examined yet.
**Insertion Sort**

**Trace of insertion sort (array contents just after each insertion)**
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) {
    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 quadratic.**

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

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 16-17: Sorting Basics

- 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