Last Time

• What can complicate our runtime analysis?
  – Large constants, non-dominant loops, instruction time, system considerations, too close to call, dependence on inputs, multiple problem parameters

• Memory
  – Integers, doubles, objects, arrays, Strings
Why Does Sorting Matter?

• “About a quarter of all computer cycles are spent sorting.” – Donald Knuth, 1973
  – Bank account transactions
  – Search engine results
  – Scientific computations – astrophysics, molecular dynamics, weather prediction, linguistics

• Suitable “prototype problem” – easily modeled and has good mathematical properties.

• The first step towards organizing and evaluating data is often to sort it.
Sorting Goals

• Sort an array of items, where each item contains a key. (what’s that?)
• After sorting, each item in the array should be arranged so that its key is ordered by some well-defined ordering rule (usually numerical or alphabetical order).
• Access and manipulate the data through two functions: less() and exch().
  – When analyzing these sorting algorithms, we will count the number of these calls separately.
• We will also look at memory usage by each algorithm:
  – Either in-place or require extra memory.
public class Example {
    public static void sort(Comparable[] a) {
        /* Sorting algorithms go here */ } //sort

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

    private static void exch(Comparable[] a, int i, int j) {
        Comparable t=a[i]; a[i]=a[j]; a[j]=t; } //exch

    private static void show(Comparable[] a) {
        /* Prints the current state */ } //show

    public static boolean isSorted(Comparable[] a) {
        /* Checks to see if a is sorted */ } //isSorted

    public static void main(String[] args) {
        /* Read input, sort, check if sorted, print */ } //main
} //Example (class)
Comparable Interface

• Implemented by all Java datatypes that can be sorted: Integer, Double, String, File, etc.
  – That means that the sort algorithms provided by the book can sort all of these datatypes!
  – That also means we can sort any class we create, just by implementing Comparable and adding a compareTo() method!
compareTo() Function

• v.compareTo(w) will
  – Return -1 if v<w
  – Return 0 if v=w
  – Return 1 if v>w

• Must implement a total order:
  – Reflexive – ∀v: v = v
  – Antisymmetric – ∀v, w: if v < w and v > w, then v = w
  – Transitive – ∀v, w, x: if v ≤ w and w ≤ x, then v ≤ x
Selection Sort

• Find the smallest item in the array; put it first.
• Find the next smallest item; put it second.
• Repeat until you’ve reached the last item in the input array.

```c
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);
} //for
```
### Selection Sort Visual

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

*entries in black are examined to find the minimum*

*entries in red are \( a[\text{min}] \)*

*entries in gray are in final position*

---

Trace of selection sort (array contents just after each exchange)
Selection Sort Evaluation

• `less()` compares
  – When $i = 1$, we compare it against the other $(n - 1)$ entries.
  – When $i = 2$, we compare it against the remaining $(n - 2)$ entries.

\[ (n - 1) + (n - 2) + \cdots + 2 + 1 = \sim \frac{n^2}{2} \text{ compares} \]

• `exch()` exchanges
  – For each $i$ value, we do one exchange, swapping $a[i]$ with $a[min]$.

\[ 1 + 1 + 1 + \cdots + 1 = n \text{ exchanges} \]
Selection Sort Evaluation

Trace of selection sort (array contents just after each exchange)

entries in black are examined to find the minimum

entries in red are a[min]

entries in gray are in final position
Selection Sort Evaluation

• Run time is insensitive to input.
  – Finding the smallest item on iteration $i$ does not give any information about the location of the smallest item in iteration $(i + 1)$.
  – Therefore, worst case = average case.

• Data movement is minimal.
  – Number of exchanges is linear w.r.t. array size.
  – No other sorting algorithm that we will consider has this property.
Any Questions?