Sorting Algorithms
Selection, Bubble, Insertion and Radix Sort

Worst-case vs. Average-case Analyses

• An algorithm can require different times to solve different problems of the same size.

• Worst-case analysis (find the maximum number of operations an algorithm can execute in all situations)
  - is easier to calculate and is more common

• Average-case (enumerate all possible situations, find the time of each of the m possible cases, total and dive by m)
  - is harder to compute but yields a more realistic expected behavior

Selection Sort

values [ 0 ]
[ 1 ]
[ 2 ]
[ 3 ]
[ 4 ]

<table>
<thead>
<tr>
<th></th>
<th>36</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Divides the array into two parts: already sorted, and not yet sorted.

On each pass, finds the smallest of the unsorted elements, and swap it into its correct place, thereby increasing the number of sorted elements by one.
Selection Sort: Pass One
values
[0] 36
[1] 24
[2] 10
[3] 6
[4] 12

Selection Sort: End Pass One
values
[0] 6
[1] 24
[2] 10
[3] 36
[4] 12

Selection Sort: Pass Two
values
[0] 6
[1] 24
[2] 10
[3] 36
[4] 12

SORTED
UNSORTED
Selection Sort: End Pass Two

<table>
<thead>
<tr>
<th>values</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>10</td>
<td>24</td>
<td>36</td>
<td>12</td>
</tr>
</tbody>
</table>

Selection Sort: Pass Three

<table>
<thead>
<tr>
<th>values</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>10</td>
<td>24</td>
<td>36</td>
<td>12</td>
</tr>
</tbody>
</table>

Selection Sort: End Pass Three

<table>
<thead>
<tr>
<th>values</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>36</td>
<td>24</td>
</tr>
</tbody>
</table>
Selection Sort: Pass Four

values [0] 6
values [1] 10
values [2] 12
values [3] 36
values [4] 24

Selection Sort: End Pass Four

values [0] 6
values [1] 10
values [2] 12
values [3] 24
values [4] 36

Selection Sort: How many comparisons?

values [0] 6
values [1] 10
values [2] 12
values [3] 24
values [4] 36

4 compares for values[0]
3 compares for values[1]
2 compares for values[2]
1 compare for values[3]

= 4 + 3 + 2 + 1
For selection sort in general

- The number of comparisons when the array contains N elements is
  \[ \text{Sum} = (N-1) + (N-2) + \ldots + 2 + 1 \]

Notice that . . .

\[
\begin{align*}
\text{Sum} &= (N-1) + (N-2) + \ldots + 2 + 1 \\
&= 1 + 2 + \ldots + (N-2) + (N-1) \\
2 \times \text{Sum} &= N + N + \ldots + N + N \\
2 \times \text{Sum} &= N \times (N-1) \\
\text{Sum} &= N \times (N-1)/2 = \frac{1}{2} N^2 - \frac{1}{2} N
\end{align*}
\]

- How many number of swaps?
  - \( O(N) \)
  - Therefore, the complexity is \( O(N^2) \)

Code for Selection Sort

```c
void SelectionSort (int values[], int numValues)
// Post: Sorts array values[0 . . numValues-1] into ascending order by key
{
    int endIndex = numValues - 1;
    for (int current=0;current<endIndex;current++)
        Swap (values, current, MinIndex(values, current, endIndex));
}
```
Selection Sort code (contd)

```c
int MinIndex(int values[], int start, int end)
// Post: Function value = index of the smallest value
// in values [start] . . values [end].
{
    int indexOfMin = start ;
    for(int index = start + 1 ; index <= end ; index++)
        if  (values[index] < values[indexOfMin])
            indexOfMin = index ;
    return indexOfMin;
}
```

Sorting Algorithms
Bubble, Insertion and Radix Sort

Bubble Sort

Values: [36, 24, 10, 6, 12]

Compares neighboring pairs of array elements, starting with the last array element, and swaps neighbors whenever they are not in correct order.

On each pass, this causes the smallest element to "bubble up" to its correct place in the array.
Code for Bubble Sort

```c
void BubbleSort(int values[], int numValues)
{
    int current = 0;
    while (current < numValues - 1)
    {
        BubbleUp(values, current, numValues-1);
        current++;
    }
}
```

Bubble Sort code (contd.)

```c
void BubbleUp(int values[], int startIndex, int endIndex)
// Post: Adjacent pairs that are out of
//   order have been switched between
//   values[startIndex]..values[endIndex]
//   beginning at values[endIndex].
{
    for (int index = endIndex;
         index > startIndex; index--)
    if (values[index] < values[index-1])
        Swap(values, index, index-1);
}
```

Observations on Bubble Sort

- There can be a large number of intermediate swaps.
- Can this algorithm be improved?
  - What are the best/worst cases?
- This algorithm is O(N^2) in the worst-case