Searching and Sorting 1-D Arrays

- Linear Search
- Binary Search
- Selection Sort
- Insertion Sort
**Objective**

- Find Something
- Two elementary searches
  - Linear
  - Binary
- Sort Something
- Two elementary sorts
  - Selection Sort
  - Insertion Sort
Motivation

- Learning Simple Search & Sorts
  - Terminology and basic context
  - Simple if often good enough
  - Simple searches/sorts are often basis for more sophisticated ones
Searching

- Scan a collection of data looking for a particular value
  - retrieve the information
  - modify/update it
  - print it
  - delete it
  - locate related information
  - etc.
Searching: examples

- Use MIRLYN
  - Look for a book with title “For One More Day“ by Mitch Albom
  - Which library has it? more than one?
  - Are any copies available? where?
  - What is the catalog number? etc.

- On the Internet
  - Search for all sites which refer to "job hunting"

- Local phone book: look up a name
Area Codes and Corresponding Locations

- Parallel arrays of size N
  - one--area codes
  - other--corresponding locations

- find area code in 1st array
  - use the index to find location in 2nd array

- We will consider two search algorithms
  - linear search
  - binary search
Linear Search: Brute Force

- look at 1st value and test for match
  if match, stop, otherwise
- look at 2nd value and test for match
  if match, stop, otherwise
- look at 3rd value
  etc...
- also stop if hit end of array
Let $N = 8$. Search for 607 (linear)

areaCodes [0]

202
203
401
413
516
607
717
802

index == 0 == index to 1st item in the array being searched

index == 1
index == 2
index == 3
index == 4
index == 5

found with 6 comparisons
Linear Search (call)

// Call the function
linearSearch(areaCodes, numAreaCodes, codeWanted, found, foundAt);

if (found)
    cout << "Search was successful. The key is: " << key << endl
    << "The element in the list is number: " << location << endl ;
else
    cout << "No match found - search is unsuccessful." << endl << endl;
void linearSearch ( const int a[], int a_len, int query, 
    bool& found, int& foundAt )
{
    found = false;

    for ( int i = 0; i < a_len; i++ )
    {
        if ( a[i] == query )
        {
            found = true;
            foundAt = i;
            return;
        }
    }
}
void linearSearch(
{
  found = false;
  for (int i = 0; i < a_len; i++) {
    if (a[i] == query) {
      found = true;
      foundAt = i;
      return;
    }
  }
}
)
void linearSearch ( const int a[], int a_len, int query )
{
    found = false;
    for ( int i = 0; i < a_len; i++ )
    {
        if ( a[i] == query )
        {
            found = true;
            foundAt = i;
            return;
        }
    }
}
void linearSearch ( const int a[], int a_len, int query,
    bool& found, int& foundAt )
{

}
void linearSearch ( const int a[], int a_len, int query,
    bool& found, int& foundAt )
{
    found = false;
    for ( int i = 0; i < a_len; i++ )
    {
    }
}
void linearSearch ( const int a[], int a_len, int query, 
    bool& found, int& foundAt )
{

    for ( int i = 0; i < a_len; i++ )
    {
        a[i]
    }
}
void linearSearch ( const int a[], int a_len, int query, 
    bool& found, int& foundAt )
{

    for ( int i = 0; i < a_len; i++ )
    {
        if ( a[i] == query )
        {

            found = true; foundAt = i; return;
        }
    }
}
void linearSearch ( const int a[], int a_len, int query,
        bool& found, int& foundAt )
{
    found = false;

    for ( int i = 0; i < a_len; i++ )
    {
        if ( a[i] == query )
        {
            found = true;
            foundAt = i;
            return;
        }
    }
}
void linearSearch ( const int a[], int a_len, int query,
    bool& found, int& foundAt )
{
    found = false;

    for ( int i = 0; i < a_len; i++ )
    {
        if ( a[i] == query )
        {
            found = true;
            foundAt = i;
            return;
        }
    }
}
Linear Search

- **Advantage:** array elements do not have to be in any order
- **Disadvantage:** very slow if the array is large
Order of Linear Search: $O(N)$

- Let $N$ be the number of things being searched through
  - In this case, the number of area codes

- Worst case:
  - $N$ comparisons
    - value sought is last one in array
    - value is not in the array

- Best case:
  - 1 comparison
    - value sought is 1st in array
On Average

- Average number of comparisons:
  - \((N + 1) / 2\)

- Actual processing time proportional to \(N\)
  - we say the algorithm is \(O(N)\)
    - "big O" notation -- the order of

- Want faster?
  - Must sort array first
Binary Search

• **Requirement:** array must be sorted

• **Principle:** check an element
  – either get a match
  – or eliminate half the elements
  – or run out of elements to search
Remember when you were a kid…

- I’m thinking of a number 1 – 100
- Number picked?
- Response?
Binary Search Algorithm

if (value sought == middle item)
    found a match
else if value sought < middle item)
    next search 1st half of list
else
    next search 2nd half of list
repeat entire process using new "half"
Let $N = 8$. Search for 607 (binary)

First = 0 = index to 1st item in part of array being searched

Mid = (first+last) / 2 = (0+7) / 2 = 3

Last = 7 = index to last item in part of array being searched
Reset first to mid + 1, then reset mid

first = 4
mid = (4 + 7) / 2 = 5
last = 7

found with 2 comparisons
binarySearch(areaCodes, numAreaCodes, codeWanted, found, foundAt);

if (found)
    cout << "Location is "
    << locations [foundAt];
else
    cout << "Area code not found."
;
void binarySearch ( const int a[], int a_len, int query,
    bool& found, int& foundAt )
{
    int first = 0, last = a_len - 1, middle;

    found = false;
    while ( first <= last ) {
        middle = ( first + last ) / 2;
        if ( query < a[middle] )
            last = middle - 1;
        else if ( query > a[middle] )
            first = middle + 1;
        else {
            found = true;
            foundAt = middle;
            return;
        }
    }
}
void binarySearch (const int a[], int a_len, int query, bool& found, int& foundAt)
{
    int first = 0, last = a_len – 1, middle; found = false; while (first <= last) {
        middle = (first + last) / 2;
        if (query < a[middle])
            last = middle - 1;
        else if (query > a[middle])
            first = middle + 1;
        else {
            found = true; foundAt = middle; return;
        }
    }
}
void binarySearch ( const int a[], int a_len, int query, 

) 

{ 

int first = 0, last = a_len - 1, middle; found = false; while ( first <= last ) { 

middle = ( first + last ) / 2; if ( query < a[middle] ) 

last = middle - 1; 

else if ( query > a[middle] ) 

first = middle + 1; 

else { 

found = true; foundAt = middle; return; 

} 

} 

}
void binarySearch ( const int a[], int a_len, int query, bool& found, int& foundAt )
{
  int first = 0, last = a_len - 1, middle; found = false;
  while ( first <= last )
  {
    middle = ( first + last ) / 2;
    if ( query < a[middle] )
      last = middle - 1;
    else if ( query > a[middle] )
      first = middle + 1;
    else
    {
      found = true; foundAt = middle;
      return;
    }
  }
}
void binarySearch ( const int a[], int a_len, int query, 
    bool& found, int& foundAt ) 
{
    int first = 0, last = a_len - 1, middle;
void binarySearch ( const int a[], int a_len, int query, 
  bool& found, int& foundAt )
{
  int first = 0, last = a_len - 1, middle;

  while ( first <= last ) {
  }
}
void binarySearch ( const int a[], int a_len, int query,
    bool& found, int& foundAt )
{
    int first = 0, last = a_len - 1, middle;

    while ( first <= last ) {
        middle = ( first + last ) / 2;
        if ( query < a[middle] )
            last = middle - 1;
        else if ( query > a[middle] )
            first = middle + 1;
        else {
            found = true; foundAt = middle; return;
        }
    }
}
void binarySearch ( const int a[], int a_len, int query,  
        bool& found, int& foundAt )
{
    int first = 0, last = a_len - 1, middle;

    while ( first <= last ) {
        middle = ( first + last ) / 2;
        if ( query < a[middle] )
            else if ( query > a[middle] )
        else
void binarySearch ( const int a[], int a_len, int query, 
              bool& found, int& foundAt )
{
    int first = 0, last = a_len - 1, middle;

    while ( first <= last ) {
        middle = ( first + last ) / 2;
        if ( query < a[middle] )
            last = middle - 1;
        else if ( query > a[middle] )
            else

void binarySearch ( const int a[], int a_len, int query,
   bool& found, int& foundAt )
{
   int first = 0, last = a_len - 1, middle;

   while ( first <= last ) {
      middle = ( first + last ) / 2;
      if ( query < a[middle] )
         last = middle - 1;
      else if ( query > a[middle] )
         first = middle + 1;
      else

   }
}
void binarySearch ( const int a[], int a_len, int query, 
    bool& found, int& foundAt )
{
    int first = 0, last = a_len - 1, middle;

    found = false;
    while ( first <= last ) {
        middle = ( first + last ) / 2;
        if ( query < a[middle] )
            last = middle - 1;
        else if ( query > a[middle] )
            first = middle + 1;
        else
        {
            found = true;
            foundAt = middle;
            return;
        }
    }
}
void binarySearch ( const int a[], int a_len, int query,
    bool& found, int& foundAt )
{
    int first = 0, last = a_len - 1, middle;

    found = false;
    while ( first <= last ) {
        middle = ( first + last ) / 2;
        if ( query < a[middle] )
            last = middle - 1;
        else if ( query > a[middle] )
            first = middle + 1;
        else
        {
            found = true;
            foundAt = middle;
            return;
        }
    }
}
void binarySearch ( const int a[], int a_len, int query,  
    bool& found, int& foundAt ) 
{
    int first = 0, last = a_len - 1, middle;

    found = false;
    while ( first <= last ) {
        middle = ( first + last ) / 2;
        if ( query < a[middle] )
            last = middle - 1;
        else if ( query > a[middle] )
            first = middle + 1;
        else {
            found = true;
            foundAt = middle;
            return;
        }
    }
}
About the Binary Search

• Why compare first <= last?
  – when first == last,
    • have reached final comparison;
    • subarray being searched reaches a size of 1
  – when first > last
    • they have "crossed"
    • no more array elements left to consider

• efficiency: why order tests this way?
  – equals case is last, is least likely in a long list
Efficiency of Binary Search

- Processing time:
  - # of comparisons is proportional to $\log_2(N)$
- Say algorithm is $O(\log_2 N)$
- e.g., if $N = 1024$
  - linear search: average # comparisons is
    - 512
  - binary search: average # comparisons is
    - $10 \rightarrow 2^{10} = 1024$
Comparison of Sequential and Binary Searches

<table>
<thead>
<tr>
<th>Length</th>
<th>Sequential Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5.5</td>
<td>2.9</td>
</tr>
<tr>
<td>100</td>
<td>50.5</td>
<td>5.8</td>
</tr>
<tr>
<td>1,000</td>
<td>500.5</td>
<td>9.0</td>
</tr>
<tr>
<td>10,000</td>
<td>5000.5</td>
<td>12.4</td>
</tr>
</tbody>
</table>
Order of Magnitude of a Function

The order of magnitude of an expression describes the complexity of an algorithm according to the highest order of $N$ that appears in its complexity expression.

If the number of operations is:

$$N^2 + 6N + 8$$

Its order of magnitude is:

$$O(N^2)$$
Names of Orders of Magnitude

- $O(1)$: constant time
- $O(\log_2 N)$: logarithmic time
- $O(N)$: linear time
- $O(N^2)$: quadratic time
- $O(N^3)$: cubic time
<table>
<thead>
<tr>
<th>N</th>
<th>$\log_2 N$</th>
<th>$N \cdot \log_2 N$</th>
<th>$N^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>24</td>
<td>64</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>160</td>
<td>1024</td>
</tr>
<tr>
<td>64</td>
<td>6</td>
<td>384</td>
<td>4096</td>
</tr>
<tr>
<td>128</td>
<td>7</td>
<td>896</td>
<td>16,384</td>
</tr>
</tbody>
</table>
## Big-O Comparison of Array Operations

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>UnsortedList</th>
<th>SortedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsPresent</td>
<td>O(N)</td>
<td>O(N) sequential search</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O(log₂N) binary search</td>
</tr>
<tr>
<td>Insert</td>
<td>O(1)</td>
<td>O(N)</td>
</tr>
<tr>
<td>Delete</td>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>SelSort</td>
<td>O(N²)</td>
<td></td>
</tr>
</tbody>
</table>
EECS 183
University of Michigan

Lecture 14 – Sorting and Searching

Running Times

- Linear: $N$\
- Quadratic: $N^2$\
- Cubic: $N^3$\
- $\log_2(N)$\
- $N\log_2(N)$
Sorting

- How do we put a list of items in order (ascending or descending)?
- Example: sort an array A with N integer elements into ascending order.

\[
A[0] \ 32 \quad -16 \\
A[1] \ 115 \quad 32 \\
A[2] \ 56 \quad 43 \\
A[3] \ -16 \quad 56 \\
A[4] \ 43 \quad 115
\]
Selection Sort

• Basic Algorithm
  – perform N-1 passes
  – on each pass,
    • find smallest element
    • exchange the first element in the unsorted part of the array with this smallest element
    • at the end of each pass, one element is thus "sorted" into the correct position
  – repeat, reducing the unsorted portion to the remaining unsorted elements

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A[0]</td>
<td>32</td>
</tr>
<tr>
<td>A[1]</td>
<td>115</td>
</tr>
</tbody>
</table>

Want to SWAP A[0] and A[3]
1st look at Exchange – “swap”

A[0] 32  
A[1] 115  

1) copy A[0] to temp
**1st look at Exchange – “swap”**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-16</td>
<td>115</td>
<td>56</td>
<td>-16</td>
<td>43</td>
</tr>
</tbody>
</table>

1) copy A[0] to temp

1) 
2) 
3) temp

```
32
```
1st look at Exchange – “swap”

A[0] -16
A[1] 115

1) copy A[0] to temp
3) copy temp to A[3]
First Pass

A[0] 32
A[1] 115

sorted

-16
115
56
32
43

unsorted
Second Pass

A[0]
-16
A[1]
115
A[2]
56
A[3]
32
43

sorted
-16
32
56
115
43

unsorted
Third Pass

A[0] -16 32
A[1] 56 115 43

unsorted
### Fourth Pass

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-16</td>
<td>32</td>
<td>43</td>
<td>115</td>
<td>56</td>
</tr>
</tbody>
</table>

Sorted list:

-16  32  43  115  56

(sorted)
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp    = a[i];
            a[i]        = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort (int a[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        int smallest = i;
        for (int j = i + 1; j < n; j++)
        {
            if (a[j] < a[smallest])
            {
                smallest = j;
            }
        }
        if (smallest != i)
        {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}

void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ ) {
        int smallest = i; for ( int j = i + 1; j < n; j++ ) {
            if ( a[j] < a[smallest] ) {
                smallest = j;
            }
        }
        if ( smallest != i ) {
            int temp = a[i]; a[i] = a[smallest]; a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i; for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }
        if ( smallest != i )
        {
            int temp = a[i]; a[i] = a[smallest]; a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i; for ( int j = i + 1; j < n; j++ )
            if ( a[j] < a[smallest] )
                smallest = j;
        if ( smallest != i )
        {
            int temp = a[i]; a[i] = a[smallest]; a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i; for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }
        if ( smallest != i )
        {
            int temp = a[i]; a[i] = a[smallest]; a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {

        for ( int j = ; j < ; j++)
        {
            a[j]
        }

    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
        
        }

    }

}

void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {

        }
    }
}

void selectionSort ( int a[], int n ) {
    for ( int i = 0; i < n - 1; i++ ) {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ ) {
            if ( a[j] < a[smallest] ) {
                smallest = j;
            }
        }
        if ( smallest != i ) {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }
        if ( smallest != i )
        {
            int temp    = a[i];
            a[i]        = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp    = a[i];
            a[i]        = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp    = a[i];
            a[i]        = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}

<table>
<thead>
<tr>
<th>n</th>
<th>i</th>
<th>smallest</th>
<th>j</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>115</td>
</tr>
<tr>
<td>2</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>-16</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
</tr>
</tbody>
</table>
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp    = a[i];
            a[i]        = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp   = a[i];
            a[i]       = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n ) {
    for ( int i = 0; i < n - 1; i++ ) {
        int smallest = i;
        for ( int j = i + 1; j < n; j++ ) {
            if ( a[j] < a[smallest] ) {
                smallest = j;
            }
        }
        if ( smallest != i ) {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}

<table>
<thead>
<tr>
<th>n</th>
<th>i</th>
<th>smallest</th>
<th>j</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
<td>A[smallest]</td>
</tr>
<tr>
<td>1</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>56</td>
<td>A[j]</td>
</tr>
<tr>
<td>3</td>
<td>-16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>43</td>
<td></td>
</tr>
</tbody>
</table>
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp    = a[i];
            a[i]        = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp    = a[i];
            a[i]        = a[smallest];
            a[smallest] = temp;
        }
    }
}
```c
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;
        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }
        if ( smallest != i )
        {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}
```

```plaintext
<table>
<thead>
<tr>
<th>n</th>
<th>i</th>
<th>smallest</th>
<th>j</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
<td>A[smallest]</td>
</tr>
<tr>
<td>1</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-16</td>
<td>A[j]</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
<td></td>
</tr>
</tbody>
</table>
```
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp    = a[i];
            a[i]        = a[smallest];
            a[smallest] = temp;
        }
    }
}

<table>
<thead>
<tr>
<th>n</th>
<th>i</th>
<th>smallest</th>
<th>j</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>115</td>
</tr>
<tr>
<td>2</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>-16</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
</tr>
</tbody>
</table>
void selectionSort ( int a[], int n )
{
    int smallest = i;
    for ( int j = i + 1; j < n; j++ )
    {
        if ( a[j] < a[smallest] )
        {
            smallest = j;
        }
    }
    if ( smallest != i )
    {
        int temp = a[i];
        a[i] = a[smallest];
        a[smallest] = temp;
    }
}

index | value
-------|------
   0   |  32
   1   | 115
   2   |  56
   3   | -16  A[smallest]
   4   |  43  A[j]
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}

<table>
<thead>
<tr>
<th>n</th>
<th>i</th>
<th>smallest</th>
<th>j</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
<td>A[i]</td>
</tr>
<tr>
<td>1</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-16</td>
<td>A[smallest]</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
<td></td>
</tr>
</tbody>
</table>
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}

<table>
<thead>
<tr>
<th>n</th>
<th>i</th>
<th>smallest</th>
<th>j</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-16</td>
</tr>
<tr>
<td>1</td>
<td>115</td>
</tr>
<tr>
<td>2</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>-16</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
</tr>
</tbody>
</table>
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp    = a[i];
            a[i]        = a[smallest];
            a[smallest] = temp;
        }
    }
}
void selectionSort ( int a[], int n )
{
    for ( int i = 0; i < n - 1; i++ )
    {
        int smallest = i;

        for ( int j = i + 1; j < n; j++ )
        {
            if ( a[j] < a[smallest] )
            {
                smallest = j;
            }
        }

        if ( smallest != i )
        {
            int temp = a[i];
            a[i] = a[smallest];
            a[smallest] = temp;
        }
    }
}
Efficiency: Selection Sort

- maximum number of comparisons:
  - \(N \times (N - 1) / 2\)
- maximum number of exchanges:
  - \(N - 1\)
- processing time is proportional to \(N^2\)
  - \(O(N^2)\)
- this is a quadratic sort (fairly slow)
  about the best we can do without using different structuring
Selection Sort - Advantages

• minimal motion of items (minimal number of swaps)
Selection Sort - Disadvantages

- running time only slightly dependent upon pre-order
  - min key on one pass tells nothing about min key on subsequent passes
  - runs about the same on
    - already sorted array
    - array with all keys equal
    - randomly arranged array
Descending Sort (1)

How would we change this selection sort function so that it sorted the array elements into descending order instead of ascending?
Descending Sort (1a)

How would we change this selection sort function so that it sorted the array elements into descending order instead of ascending?

- change the `<` operator in the comparison to `>`
- change the name of the variable `smallest` to `largest`

```
```
A Note on sorting parallel arrays

- example: sort the area codes and locations arrays we discussed earlier into ascending numeric order, by area code
- the area code is called the sort key
- comparison: use one array
  \[
  \text{if (area_codes}[j] < \text{area_codes [small]})
  \]
- exchange: perform in both arrays
  area_codes[i] with area_codes[small]
  locations[i] with locations[small]
**Insertion Sort**

- Consider elements one at a time
- Insert each into its proper place among those already considered
Insertion Sort

one item is automatically in order
**Insertion Sort**

3
16
12
1
2

- Compare this element against the one above it.
- If it is greater than it, so it is in order.
Insertion Sort

3 16
12
1
2

compare this element against the one above it

It is < it, so swap

3 12 16
1
2

compare: OK
Insertion Sort

3
12
16
1
2

compare
it is < swap

3
12
16
1
2

compare swap

3
12
16
2

3
1
12
16
2

Lecture 14 – Sorting and Searching
Insertion Sort

3
1
12
16
2

1
3
12
16
2

compare

it is < swap

compare
Insertion Sort

1 3 12 16

1 3 12

1 3

1

2 12 16

2 12

2

1

1 2 3 12 16

1 2

1 3

1 16
void insertionSort( int a[], int n ) {
    for ( int i = 1; i < n; i++ ) {
        for ( int j = i; j > 0 && a[j] < a[j-1]; j-- ) {
            int temp = a[j];
            a[j] = a[j-1];
            a[j-1] = temp;
        }
    }
}
void insertionSort(int a[], int n) {
    for (int i = 1; i < n-1; i++) {
        for (int j = i; j > 0 && a[j] < a[j-1]; j--) {
            int temp = a[j];
            a[j] = a[j-1];
            a[j-1] = temp;
        }
    }
}
Insertion Sort

```c
void insertionSort( int a[], int n )
{
   for ( int i = 1; i < n-1; i++) {
      for ( int j = i; j > 0 && a[j] < a[j-1]; j-- ) {
         int temp = a[j]; a[j] = a[j-1]; a[j-1] = temp;
      }
   }
}
```
void insertionSort( int a[], int n )
{
    for ( int i = ; i < ; i++)
    {
        for ( int j = i; j > 0 && a[j] < a[j-1]; j-- ) {
            int temp = a[j]; a[j] = a[j-1]; a[j-1] = temp;
        }
    }
}
Insertion Sort

```c
void insertionSort( int a[], int n )
{
    for ( int i = 1; i < n; i++ )
    {
        for ( int j = i; j > 0 && a[j] < a[j-1]; j-- )
        {
            int temp = a[j];
            a[j] = a[j-1];
            a[j-1] = temp;
        }
    }
}
```
Insertion Sort

```c
void insertionSort( int a[], int n )
{
    for ( int i = 1; i < n; i++)
    {
        for ( int j = ; j > ; j-- )
        {

        }
    }
}
```
void insertionSort( int a[], int n )
{
    for ( int i = 1; i < n; i++)
    {
        for ( int j = i; j > 0; j-- )
        {
            int temp = a[j]; a[j] = a[j-1]; a[j-1] = temp;
        }
    }
}
void insertionSort( int a[], int n )
{
    for ( int i = 1; i < n; i++)
    {
        for ( int j = i; j > 0 && a[j] < a[j-1]; j-- )
        {
        
        }
    }
}
Insertion Sort

```c
void insertionSort( int a[], int n )
{
    for ( int i = 1; i < n; i++ )
    {
        for ( int j = i; j > 0 && a[j] < a[j-1]; j-- )
        {
            int temp = a[j];
            a[j] = a[j-1];
            a[j-1] = temp;
        }
    }
}
```
Insertion Sort

```c
void insertionSort( int a[], int n )
{
    for ( int i = 1; i < n; i++ )
    {
        for ( int j = i; j > 0 && a[j] < a[j-1]; j-- )
        {
            int temp = a[j];
            a[j] = a[j-1];
            a[j-1] = temp;
        }
    }
}
```
Insertion Sort

- Great for reading in values and inserting them into an array in order.
Insertion Sort

• Analysis
  – $n^2/2$ comparisons worst case
  – $n^2/2$ swaps worst case
  – $n^2/4$ comparisons average case
  – $n^2/4$ swaps average case
  – $n$ comparisons best case
  – $0$ swaps best case
Insertion Sort - Advantages

- Running time depends upon initial order of keys
- Algorithm is “tune-able”
- With ‘cleverness, can be made $n^2/4$ for compares and swaps
Insertion Sort - Disadvantages

- still $O(n^2)$
Summary: Elementary Sorts

• Several simple sorts
  – Selection Sort
  – Insertion Sort

• All these simple sorts are $O(n^2)$

• Some sorts execute more quickly given special cases