Chapter 15

Recursive Algorithms and Sorting

CS 180
Sunil Prabhakar
Department of Computer Science
Purdue University
Objectives

After this week, you should be able to

- Write recursive algorithms.
- Decide when to use recursion.
- Perform linear and binary search algorithms on small arrays.
- Determine whether a linear or binary search is more effective for a given situation.
- Understand selection, bubble sort, and (recursive) quicksort algorithms.
- Understand how these algorithms compare.
Recursive Algorithms

- Within a given method, we are allowed to call other accessible methods.
- It is also possible to call the same method from within the method itself.
- This is called a recursive call.
- For certain problems a recursive solution is very natural and simple.
- It is possible to implement a recursive algorithm without using recursion, but the code can be more complex.
Example of Recursion
Example of Recursion

- The *factorial of N* is the product of the first N positive integers:

\[ N! = N \times (N - 1) \times (N - 2) \times \cdots \times 2 \times 1 \]
Example of Recursion

- The **factorial of N** is the product of the first N positive integers:
  \[ N! = N \times (N-1) \times (N-2) \times \cdots \times 2 \times 1 \]

- This is useful for many situations, e.g.
  - there are \( n! \) possible sequences of \( n \) objects
  - there are \( n!(n-k)!/k! \) unique subsets of size \( k \), from a set of size \( n \). 
Example of Recursion

- The **factorial of N** is the product of the first N positive integers:
  \[ N! = N \times (N-1) \times (N-2) \times \ldots \times 2 \times 1 \]

- This is useful for many situations, e.g.
  - there are \( n! \) possible sequences of \( n \) objects
  - there are \( \frac{n!(n-k)!}{k!} \) unique subsets of size \( k \), from a set of size \( n \).

- The factorial of N can be defined **recursively** as
Example of Recursion

- The **factorial of N** is the product of the first N positive integers:
  \[ N! = N \times (N - 1) \times (N - 2) \times \ldots \times 2 \times 1 \]

- This is useful for many situations, e.g.
  - there are \( n! \) possible sequences of \( n \) objects
  - there are \( n!(n-k)!/k! \) unique subsets of size \( k \), from a set of size \( n \).

- The factorial of N can be defined *recursively* as

\[
\text{factorial}( N ) = \begin{cases} 
1 & \text{if } N = 1 \\
N \times \text{factorial}( N-1 ) & \text{otherwise}
\end{cases}
\]
Recursive Method

- An **recursive method** is a method that contains a statement (or statements) that makes a call to itself.

- Implementing the factorial of N recursively will result in the following method.
Recursive Method

- An *recursive method* is a method that contains a statement (or statements) that makes a call to itself.
- Implementing the factorial of N recursively will result in the following method.

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
```
Recursive Method

- An *recursive method* is a method that contains a statement (or statements) that makes a call to itself.
- Implementing the factorial of \( N \) recursively will result in the following method.

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
```
An **recursive method** is a method that contains a statement (or statements) that makes a call to itself.

Implementing the factorial of N recursively will result in the following method.

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
```

End case: recursion stops.

Test to stop or continue.
A **recursive method** is a method that contains a statement (or statements) that makes a call to itself.

Implementing the factorial of N recursively will result in the following method.

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
```

**End case: recursion stops.**

**Recursive case:** recursion continues.

**Test to stop or continue.**
The details …

- As with any call, a recursive call results in the creation of temporary workspace for the called method and copying of parameters.
- Each call to a method results in the creation of a separate workspace with new copies of variables (local and formal parameters).
- When a recursive call ends, flow returns to the caller.
Example: \texttt{factorial(3);}
Example

```java
public int factorial( int N ){
    if ( N == 1 ){
        return 1;
    } else {
        return N * factorial( N-1 );
    }
}
```

`factorial( 3 );`
Example

```java
public int factorial( int N ) {
    if ( N == 1 ) {
        return 1;
    } else {
        return N * factorial( N-1 );
    }
}
```

`factorial(3);`
Example

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
```

`factorial(3);`
Example

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}

factorial(3);  // This calls the factorial function with N = 3
```
Example

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
factorial(3);
```

Example output:

```
3
```
Example

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N - 1);
    }
}

factorial(3);
```
Example

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N - 1);
    }
}
```

Call to the `factorial` method:
```
factorial(3);
```
Example

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
```

`factorial(3);`
public int factorial( int N ) {
    if ( N == 1 ) {
        return 1;
    } else {
        return N * factorial( N-1 );
    }
}

factorial( 3 );
public int factorial( int N ) {
    if ( N == 1 ) {
        return 1;
    } else {
        return N * factorial( N-1 );
    }
}
Example

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
```

```java
public int factorial( int N ) { 
   if ( N == 1 ) {
      return 1;
   } else {
      return N * factorial( N-1 );
   }
}
```

```java
public int factorial( int N ) { 
   if ( N == 1 ) {
      return 1;
   } else {
      return N * factorial( N-1 );
   }
}
```

```java
public int factorial( int N ) { 
   if ( N == 1 ) {
      return 1;
   } else {
      return N * factorial( N-1 );
   }
}
```

`factorial(3);`
Example

public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
Example

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
```

```
factorial(3);
```
Example

```java
public int factorial( int N ) {
    if ( N == 1 ) {
        return 1;
    } else {
        return N * factorial( N-1 );
    }
}
```

```java
public int factorial( int N ) { 
   if ( N == 1 ) {
      return 1;
   } else {
      return N * factorial( N-1 );
   }
}
```

```java
public int factorial( int N ) {
    if ( N == 1 ) {
        return 1;
    } else {
        return N * factorial( N-1 );
    }
}
```
Example

public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}

factorial(3);
Example

```java
public int factorial( int N ) {
    if ( N == 1 ) {
        return 1;
    } else {
        return N * factorial( N-1 );
    }
}
```
Example

```java
public int factorial(int N) {
    if (N == 1) {
        return 1;
    } else {
        return N * factorial(N-1);
    }
}
```

Example using the function:

```
factorial(3);
```
Example

$$\text{factorial}(3);$$

6
Directory Listing

List the names of all files in a given directory and its subdirectories.

```java
public void directoryListing(File dir) {
    //assumption: dir represents a directory
    String[] fileList = dir.list(); //get the contents
    String dirPath = dir.getAbsolutePath();

    for (int i = 0; i < fileList.length; i++) {
        File file = new File(dirPath + "/" + fileList[i]);

        if (file.isFile()) { //it's a file
            System.out.println(file.getName());
        } else { //it's a directory
            directoryListing(file); //so make a recursive call
        }
    }
}
```
List the names of all files in a given directory and its subdirectories.

```java
public void directoryListing(File dir) {
    //assumption: dir represents a directory
    String[] fileList = dir.list(); //get the contents
    String dirPath = dir.getAbsolutePath();

    for (int i = 0; i < fileList.length; i++) {
        File file = new File(dirPath + "/" + fileList[i]);
        if (file.isFile()) { //it's a file
            System.out.println(file.getName());
        } else { //it's a directory
            directoryListing(file); //so make a recursive call
        }
    }
}
```
Directory Listing

- List the names of all files in a given directory and its subdirectories.

```java
public void directoryListing(File dir) {
    //assumption: dir represents a directory
    String[] fileList = dir.list(); //get the contents
    String dirPath = dir.getAbsolutePath();

    for (int i = 0; i < fileList.length; i++) {
        File file = new File(dirPath + "/" + fileList[i]);
        if (file.isFile()) {
            System.out.println(file.getName());
        } else {
            directoryListing(file); //it's a directory
            //so make a recursive call
        }
    }
}
```
Directory Listing

- List the names of all files in a given directory and its subdirectories.

```java
public void directoryListing(File dir) {
    // assumption: dir represents a directory
    String[] fileList = dir.list(); // get the contents
    String dirPath = dir.getAbsolutePath();

    for (int i = 0; i < fileList.length; i++) {
        File file = new File(dirPath + "\" + fileList[i]);

        if (file.isFile()) {
            // it's a file
            System.out.println(file.getName());
        } else {
            // it's a directory
            directoryListing(file); // so make a directory
            // recursive call
        }
    }
}
```
Anagram

List all anagrams of a given word.

Word → C A T
Anagram

List all anagrams of a given word.

Word

C A T

C T A
A T C
A C T
T C A
T A C

Anagrams
Anagram Solution

- The basic idea is to make recursive calls on a sub-word after every rotation. Here’s how:
Anagram Solution

- The basic idea is to make recursive calls on a sub-word after every rotation. Here’s how:

  C  A  T
Anagram Solution

- The basic idea is to make recursive calls on a sub-word after every rotation. Here’s how:

```
C
A
T
```

Recursion

```
C A T
C T A
```
Anagram Solution

- The basic idea is to make recursive calls on a sub-word after every rotation. Here’s how:

```
C A T
C T A
```

Rotate Left

Recursion

```
A T C
```

C A T
C T A
Anagram Solution

- The basic idea is to make recursive calls on a sub-word after every rotation. Here’s how:

```
C A T
C T A
A T C
A C T

Recursion

Rotate Left
```
The basic idea is to make recursive calls on a sub-word after every rotation. Here’s how:

- **CAT**
- **CTA**
- **ATC**
- **ACT**

Rotate Left

Recursion

Rotate Left

Recursion
Anagram Solution

- The basic idea is to make recursive calls on a sub-word after every rotation. Here’s how:

```plaintext
  C  A  T
  A  T  C
  T  C  A

  Recursion
  Rotate Left
  Recursion
  Rotate Left
  Recursion
```

- C A T
- C T A
- A T C
- A C T
- T C A
- T A C

Purdue University
public void anagram(String prefix, String suffix) {
    String newPrefix, newSuffix;
    int numOfChars = suffix.length();

    if (numOfChars == 1) {
        // End case: print out one anagram
        System.out.println(prefix + suffix);
    } else {
        for (int i = 1; i <= numOfChars; i++) {
            newSuffix = suffix.substring(1, numOfChars);
            newPrefix = prefix + suffix.charAt(0);
            anagram(newPrefix, newSuffix);
            // recursive call
            // rotate left to create a rearranged suffix
            suffix = newSuffix + suffix.charAt(0);
        }
    }
}
Anagram Method

```java
public void anagram( String prefix, String suffix ) {
    String newPrefix, newSuffix;
    int numOfChars = suffix.length();

    if (numOfChars == 1) {
        //End case: print out one anagram
        System.out.println( prefix + suffix );
    } else {
        for (int i = 1; i <= numOfChars; i++) {
            newSuffix = suffix.substring(1, numOfChars);
            newPrefix = prefix + suffix.charAt(0);
            anagram( newPrefix, newSuffix );
            //recursive call
            //rotate left to create a rearranged suffix
            suffix = newSuffix + suffix.charAt(0);
        }
    }
}
```

public void anagram(String prefix, String suffix) {
    String newPrefix, newSuffix;
    int numOfChars = suffix.length();
    if (numOfChars == 1) {
        // End case: print out one anagram
        System.out.println(prefix + suffix);
    } else {
        for (int i = 1; i <= numOfChars; i++) {
            newSuffix = suffix.substring(1, numOfChars);
            newPrefix = prefix + suffix.charAt(0);
            anagram(newPrefix, newSuffix);
            // recursive call
            // rotate left to create a rearranged suffix
            suffix = newSuffix + suffix.charAt(0);
        }
    }
}
public void anagram(String prefix, String suffix) {
    String newPrefix, newSuffix;
    int numOfChars = suffix.length();
    if (numOfChars == 1) {
        System.out.println(prefix + suffix);
    } else {
        for (int i = 1; i <= numOfChars; i++) {
            newSuffix = suffix.substring(1, numOfChars);
            newPrefix = prefix + suffix.charAt(0);
            anagram(newPrefix, newSuffix);
        }
    }
}
The goal of the Towers of Hanoi puzzle is to move $N$ disks from Peg 1 to Peg 3:
The goal of the Towers of Hanoi puzzle is to move N disks from Peg 1 to Peg 3:

- Only one disk can be moved at a time.
- A larger disk cannot be placed on top of a smaller disk.
Towers of Hanoi Solution
Towers of Hanoi Solution
Towers of Hanoi Solution
Towers of Hanoi Solution
Towers of Hanoi Solution
Towers of Hanoi Solution

Peg1  Peg2  Peg3

Peg1  Peg2  Peg3

Peg1  Peg2  Peg3

Peg1  Peg2  Peg3

Peg1  Peg2  Peg3
Towers of Hanoi Solution
Towers of Hanoi Solution
public void towersOfHanoi(int N, int from, int to, int spare) {
    if (N == 1) {
        moveOne(from, to);
    } else {
        towersOfHanoi(N-1, from, spare, to);
        moveOne(from, to);
        towersOfHanoi(N-1, spare, to, from);
    }
}

private void moveOne(int from, int to) {
    System.out.println(from + " ---> " + to);
}
### towersOfHanoi Method

```java
public void towersOfHanoi(int N, //number of disks
         int from,  //origin peg
         int to,    //destination peg
         int spare ){//"middle" peg

    if ( N == 1 ) {
        moveOne( from, to );
    } else {
        towersOfHanoi( N-1, from, spare, to );
        moveOne( from, to );
        towersOfHanoi( N-1, spare, to, from );
    }
}

private void moveOne( int from, int to ) {
    System.out.println( from + " ---> " + to );
}
```

public void towersOfHanoi(int N, int from, int to, int spare) {
    if (N == 1) {
        moveOne(from, to);
    } else {
        towersOfHanoi(N-1, from, spare, to);
        moveOne(from, to);
        towersOfHanoi(N-1, spare, to, from);
    }
}

private void moveOne(int from, int to) {
    System.out.println(from + " ---> " + to);
}
public void towersOfHanoi(int N,  //number of disks  
                    int from,  //origin peg  
                    int to,   //destination peg  
                    int spare ){  //"middle" peg
    if ( N == 1 ) {
        moveOne( from, to );
    } else {
        towersOfHanoi( N-1, from, spare, to );
        moveOne( from, to );
        moveOne( from, to );
        towersOfHanoi( N-1, spare, to, from );
    }
}

private void moveOne( int from, int to ) {
    System.out.println( from + " ---> " + to );
}
Searching

- When we maintain a collection of data, one of the operations we need is a search routine to locate desired data quickly.

- Here’s the problem statement:
  
  Given a value $X$, return the index of $X$ in the array, if such $X$ exists. Otherwise, return NOT_FOUND (-1). We assume there are no duplicate entries in the array.

- We will count the number of comparisons the algorithms make to analyze their performance.
  
  - The ideal searching algorithm will make the least possible number of comparisons to locate the desired data.
  - Two separate performance analyses are normally done, one for successful search and another for unsuccessful search.
Search Result

Unsuccessful Search:

Successful Search:

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>17</td>
<td>5</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
</tbody>
</table>
Search Result

Unsuccessful Search: \texttt{search( 45 )} \rightarrow \texttt{NOT_FOUND}

Successful Search:

\begin{center}
\begin{tabular}{c|c|c|c|c|c|c|c|c|c|c}
\hline
\text{number} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
23 & 17 & 5 & 90 & 12 & 44 & 38 & 84 & 77 \\
\hline
\end{tabular}
\end{center}
Unsuccessful Search:  \( \text{search}(45) \rightarrow \text{NOT\_FOUND} \)

Successful Search:  \( \text{search}(12) \rightarrow 4 \)

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
23 & 17 & 5 & 90 & 12 & 44 & 38 & 84 & 77 \\
\end{array}
\]
Linear Search

- Search the array from the first to the last position in linear progression.

```java
public int linearSearch ( int[] number, int searchValue ) {
    int loc = 0;
    while (loc < number.length && number[loc] != searchValue) {
        loc++;
    }
    if (loc == number.length) { //Not found
        return NOT_FOUND;
    } else {
        return loc;  //Found, return the position
    }
}
```
Linear Search Performance

- We analyze the successful and unsuccessful searches separately.
- We count how many times the search value is compared against the array elements.

**Successful Search**
- Best Case – 1 comparison
- Worst Case – N comparisons (N – array size)

**Unsuccessful Search**
- Best Case = Worst Case = N comparisons
Binary Search

5 12 17 23 38 44 77 84 90
**Binary Search**

- **If the array is sorted**, then we can apply the binary search technique.

- The basic idea is straightforward. First search the value in the middle position. If X is less than this value, then search the middle of the left half next. If X is greater than this value, then search the middle of the right half next. Continue in this manner.
Sequence of Successful Search - 1

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>17</td>
</tr>
<tr>
<td>23</td>
<td>38</td>
<td>44</td>
</tr>
<tr>
<td>77</td>
<td>84</td>
<td>90</td>
</tr>
</tbody>
</table>

search( 44 )
### Sequence of Successful Search - 1

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

#### #1

```
low  high  mid
0    8
```

```python
search(44)
```

```
0   1   2   3   4   5   6   7   8
5   12  17  23  38  44  77  84  90
```

- **low**: 0
- **high**: 8
- **search(44)**

---

**PURDUE UNIVERSITY**
Sequence of Successful Search - 1

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

search( 44 )

\[
mid = \left\lfloor \frac{low + high}{2} \right\rfloor
\]
Sequence of Successful Search - 1

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

search(44)

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$
**Sequence of Successful Search - 1**

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

### search(44)

\[
mid = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>17</td>
<td>23</td>
<td>38</td>
<td>44</td>
<td>77</td>
<td>84</td>
<td>90</td>
</tr>
</tbody>
</table>

\[38 < 44\]
Sequence of Successful Search - 1

<table>
<thead>
<tr>
<th>#1</th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

search( 44 )

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>17</td>
<td>23</td>
<td>38</td>
<td>44</td>
<td>77</td>
<td>84</td>
<td>90</td>
</tr>
</tbody>
</table>

38 < 44 \rightarrow \text{low} = \text{mid} + 1 = 5
Sequence of Successful Search - 2

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
\text{search}(44)
\]

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

- **low**: 0
- **high**: 8
- **mid**: 4

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>17</td>
<td>23</td>
<td>38</td>
<td>44</td>
<td>77</td>
<td>84</td>
<td>90</td>
</tr>
</tbody>
</table>
## Sequence of Successful Search - 2

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

**Search(44)**

\[
mid = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

- #1: low = 0, high = 8, mid = 4
- #2: low = 5, high = 8

---

![Binary Search Diagram](image)

- low = 0
- high = 8

- Search(44) with low = 0 and high = 8
- mid = 4
- Move to the right

- Search(77) with low = 4 and high = 8
- mid = 6
- Move to the right

- Search(84) with low = 6 and high = 8
- mid = 7
- Move to the right

- Search(90) with low = 7 and high = 8
- mid = 8
- Move to the left

---

**Note:**
- The binary search is conducted by comparing the target value with the middle value of the current search range.
- If the target is less than the middle value, the search continues in the lower half of the range.
- If the target is greater than the middle value, the search continues in the upper half of the range.
- The process repeats until the target is found or the range is exhausted.
### Sequence of Successful Search - 2

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

**search(44)**

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>44</td>
<td>77</td>
<td>84</td>
<td>90</td>
</tr>
</tbody>
</table>
Sequence of Successful Search - 2

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

search( 44 )

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

0 1 2 3 4 5 6 7 8

44 77 84 90

\[44 < 77\]
# Sequence of Successful Search - 2

<table>
<thead>
<tr>
<th>#1</th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

```
search( 44 )
```

```
mid = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
```

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>44</td>
<td>77</td>
<td>84</td>
<td>90</td>
</tr>
</tbody>
</table>

```
high = mid-1=5  ←  44 < 77
```
Sequence of Successful Search - 3

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

search(44)

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

<table>
<thead>
<tr>
<th></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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>44</td>
<td>77</td>
<td>84</td>
<td>90</td>
<td></td>
</tr>
</tbody>
</table>
Sequence of Successful Search - 3

<table>
<thead>
<tr>
<th>#</th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

```
search( 44 )
```

```
0  1  2  3  4  5  6  7  8
```

44
### Sequence of Successful Search - 3

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Search(44)

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]
Sequence of Successful Search - 3

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

search( 44 )

\[ \text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor \]

Successful Search!!

44 == 44
Sequence of Unsuccessful Search - 1

low  high  mid

search(45)
Sequence of Unsuccessful Search - 1

<table>
<thead>
<tr>
<th>#1</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

search( 45 )
Sequence of Unsuccessful Search - 1

\[
\text{search( 45 )}
\]

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

0  1  2  3  4  5  6  7  8
5  12 17 23 38 44 77 84 90

low

high
Sequence of Unsuccessful Search - 1

\[
\begin{align*}
\text{low} & \quad \text{high} & \quad \text{mid} \\
#1 & \quad 0 & \quad 8 & \quad 4
\end{align*}
\]

search( 45 )

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

0 1 2 3 4 5 6 7 8

5 12 17 23 38 44 77 84 90

low

mid

high
Sequence of Unsuccessful Search - 1

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

search( 45 )

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>17</td>
<td>23</td>
<td>38</td>
<td>44</td>
<td>77</td>
<td>84</td>
<td>90</td>
</tr>
</tbody>
</table>

38 < 45 \rightarrow \text{low} = \text{mid} + 1 = 5
## Sequence of Unsuccessful Search - 2

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

### search(45)

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>17</td>
<td>23</td>
<td>38</td>
<td>44</td>
<td>77</td>
<td>84</td>
<td>90</td>
</tr>
</tbody>
</table>
Sequence of Unsuccessful Search - 2

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

search( 45 )

\[
mid = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

 PURDUE UNIVERSITY
Sequence of Unsuccessful Search - 2

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

search( 45 )

\[
mid = \left\lfloor \frac{low + high}{2} \right\rfloor
\]
Sequence of Unsuccessful Search - 2

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

The search for 45 is unsuccessful because it is not found in the sequence.

### Calculation

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

### Diagram

- **0** 1 2 3 4 5 6 7 8
- **44** 77 84 90
- **45 < 77**
- **low** mid high
Sequence of Unsuccessful Search - 2

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

search( 45 )

\[
mid = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

0 1 2 3 4 5 6 7 8

44 77 84 90

high = mid-1 = 5

45 < 77
Sequence of Unsuccessful Search - 3

<table>
<thead>
<tr>
<th>#</th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

search(45)

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]
### Sequence of Unsuccessful Search - 3

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**search(45)**

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

---

**Diagram:**

- **low** and **high** values are shown.
- **mid** value is calculated.
- **44** is the target value.
### Sequence of Unsuccessful Search - 3

<table>
<thead>
<tr>
<th>#</th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

**search(45)**

$$ mid = \left\lfloor \frac{low + high}{2} \right\rfloor $$

0 1 2 3 4 5 6 7 8

![Diagram showing the search process](image)
Sequence of Unsuccessful Search - 3

<table>
<thead>
<tr>
<th>#</th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

$$mid = \left\lceil \frac{low + high}{2} \right\rceil$$

```
search( 45 )
```

```
0 1 2 3 4 5 6 7 8
```

44 < 45
## Sequence of Unsuccessful Search - 3

<table>
<thead>
<tr>
<th>#</th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

### search(45)

\[
mid = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

**Diagram:**

- Initial range: low = 0, high = 8
- After first iteration: low = 5, high = 8, mid = 6
- After second iteration: low = 5, high = 5, mid = 5

44 < 45

\[ \rightarrow \text{low} = \text{mid} + 1 = 6 \]
Sequence of Unsuccessful Search - 4

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

\[ mid = \left\lfloor \frac{low + high}{2} \right\rfloor \]

0 1 2 3 4 5 6 7 8

search( 45 )

44
### Sequence of Unsuccessful Search - 4

<table>
<thead>
<tr>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>#4</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

**search( 45 )**

\[
mid = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

**Diagram:**

- **low:** 0, 5, 6
- **high:** 8, 8, 5
- **mid:** 4, 6, 5
Sequence of Unsuccessful Search - 4

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>#4</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Search( 45 )

\[
\text{mid} = \left\lfloor \frac{\text{low} + \text{high}}{2} \right\rfloor
\]

low > high
Sequence of Unsuccessful Search - 4

<table>
<thead>
<tr>
<th></th>
<th>low</th>
<th>high</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>0</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>#3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>#4</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Unsuccessful Search

search( 45 )

mid = \left\lfloor \frac{low + high}{2} \right\rfloor

Unsuccessful Search

no more elements to search

low > high
public int binarySearch (int[] number, int searchValue) {
    int low = 0,
        high = number.length - 1,
        mid = (low + high) / 2;
    while (low <= high && number[mid] != searchValue) {
        if (number[mid] < searchValue) {
            low = mid + 1;
        } else { //number[mid] > searchValue
            high = mid - 1;
        }
        mid = (low + high) / 2; //integer division will truncate
    }
    if (low > high) {
        mid = NOT_FOUND;
    }
    return mid;
}
Binary Search Performance

- **Successful Search**
  - Best Case – 1 comparison
  - Worst Case – $\log_2 N$ comparisons

- **Unsuccessful Search**
  - Best Case = Worst Case – $\log_2 N$ comparisons

Since the portion of an array to search is cut into half after every comparison, we compute how many times the array can be divided into halves.

After $K$ comparisons, there will be $N/2^K$ elements in the list. We solve for $K$ when $N/2^K = 1$, deriving $K = \log_2 N$. 
## Comparing N and $\log_2 N$ Performance

<table>
<thead>
<tr>
<th>Array Size</th>
<th>Linear – N</th>
<th>Binary – $\log_2 N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>6</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>500</td>
<td>500</td>
<td>9</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>2000</td>
<td>2000</td>
<td>11</td>
</tr>
<tr>
<td>3000</td>
<td>3000</td>
<td>12</td>
</tr>
<tr>
<td>4000</td>
<td>4000</td>
<td>12</td>
</tr>
<tr>
<td>5000</td>
<td>5000</td>
<td>13</td>
</tr>
<tr>
<td>6000</td>
<td>6000</td>
<td>13</td>
</tr>
<tr>
<td>7000</td>
<td>7000</td>
<td>13</td>
</tr>
<tr>
<td>8000</td>
<td>8000</td>
<td>13</td>
</tr>
<tr>
<td>9000</td>
<td>9000</td>
<td>14</td>
</tr>
<tr>
<td>10000</td>
<td>10000</td>
<td>14</td>
</tr>
</tbody>
</table>
When we maintain a collection of data, many applications call for rearranging the data in certain order, e.g. arranging Person information in ascending order of age.

Here’s the problem statement:

Given an array of N integer values, arrange the values into ascending order.

We will count the number of comparisons the algorithms make to analyze their performance.

- The ideal sorting algorithm will make the least possible number of comparisons to arrange data in a designated order.

We will compare different sorting algorithms by analyzing their worst-case performance.
Selection Sort
Selection Sort

1. Find the smallest element in the list.

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>5</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
</tbody>
</table>
1. Find the smallest element in the list.
1. Find the smallest element in the list.

2. Exchange the element in the first position and the smallest element. Now the smallest element is in the first position.
### Selection Sort

1. **Find the smallest element in the list.**

2. **Exchange the element in the first position and the smallest element.** Now the smallest element is in the first position.

3. **Repeat Step 1 and 2 with the list having one less element (i.e., the smallest element is discarded from further processing).**

This is the result of one pass.
Selection Sort Passes
Selection Sort Passes

<table>
<thead>
<tr>
<th>Pass #</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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>17</td>
<td>23</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
</tbody>
</table>

Sorted

Result AFTER one pass is completed.
### Selection Sort Passes

<table>
<thead>
<tr>
<th>Pass #</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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>17</td>
<td>23</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
</tbody>
</table>

Sorted:

| 5  | 12 | 23 | 90 | 17 | 44 | 38 | 84 | 77 |

Result AFTER one pass is completed.
## Selection Sort Passes

<table>
<thead>
<tr>
<th>Pass #</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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>17</td>
<td>23</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sorted

<table>
<thead>
<tr>
<th>Pass #</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>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>12</td>
<td>23</td>
<td>90</td>
<td>17</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Result AFTER one pass is completed.

<table>
<thead>
<tr>
<th>Pass #</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>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>12</td>
<td>17</td>
<td>90</td>
<td>23</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Selection Sort Passes

<table>
<thead>
<tr>
<th>Pass #</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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>17</td>
<td>23</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>12</td>
<td>23</td>
<td>90</td>
<td>17</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>12</td>
<td>17</td>
<td>90</td>
<td>23</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>12</td>
<td>17</td>
<td>23</td>
<td>38</td>
<td>44</td>
<td>77</td>
<td>84</td>
<td>90</td>
</tr>
</tbody>
</table>

Result AFTER one pass is completed.
Selection Sort Passes

<table>
<thead>
<tr>
<th>Pass #</th>
<th>Pass 0</th>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
<th>Pass 4</th>
<th>Pass 5</th>
<th>Pass 6</th>
<th>Pass 7</th>
<th>Pass 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 17 23 90 12 44 38 84 77</td>
<td>5 17 23 90 12 44 38 84 77</td>
<td>5 12 23 90 17 44 38 84 77</td>
<td>5 12 17 90 23 44 38 84 77</td>
<td>5 12 17 23 38 44 77 84 90</td>
<td>5 12 17 23 38 44 77 84 90</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Result AFTER one pass is completed.

Sorted list:

- Pass 1: 5 17 23 90 12 44 38 84 77
- Pass 2: 5 12 23 90 17 44 38 84 77
- Pass 3: 5 12 17 90 23 44 38 84 77
- Pass 4: 5 12 17 23 38 44 77 84 90
- Pass 5: 5 12 17 23 38 44 77 84 90
- Pass 6: 5 12 17 23 38 44 77 84 90
- Pass 7: 5 12 17 23 38 44 77 84 90
- Pass 8: 5 12 17 23 38 44 77 84 90
Selection Sort Routine

```java
public void selectionSort(int[] number) {
    int startIndex, minIndex, length, temp;
    length = number.length;

    for (startIndex = 0; startIndex <= length-2; startIndex++) {
        // each iteration of the for loop is one pass
        minIndex = startIndex;

        // find the smallest in this pass at position minIndex
        for (i = startIndex+1; i <= length-1; i++) {
            if (number[i] < number[minIndex]) minIndex = i;
        }

        // exchange number[startIndex] and number[minIndex]
        temp = number[startIndex];
        number[startIndex] = number[minIndex];
        number[minIndex] = temp;
    }
}
```
Selection Sort Performance

- We derive the total number of comparisons by counting the number of times the inner loop is executed.
- For each execution of the outer loop, the inner loop is executed \textit{length} – \textit{start} times.
Selection Sort Performance

- We derive the total number of comparisons by counting the number of times the inner loop is executed.
- For each execution of the outer loop, the inner loop is executed $\text{length} - \text{start}$ times.

<table>
<thead>
<tr>
<th>Start</th>
<th>Number of Comparisons ($\text{Length} - \text{Start}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\text{length}$</td>
</tr>
<tr>
<td>1</td>
<td>$\text{length} - 1$</td>
</tr>
<tr>
<td>2</td>
<td>$\text{length} - 2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\text{length} - 2$</td>
<td>2</td>
</tr>
</tbody>
</table>
We derive the total number of comparisons by counting the number of times the inner loop is executed.

For each execution of the outer loop, the inner loop is executed $\text{length} - \text{start}$ times.

The variable $\text{length}$ is the size of the array. Replacing $\text{length}$ with $N$, the array size, the sum is derived as...

<table>
<thead>
<tr>
<th>Start</th>
<th>Number of Comparisons (Length − Start)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\text{length}$</td>
</tr>
<tr>
<td>1</td>
<td>$\text{length} - 1$</td>
</tr>
<tr>
<td>2</td>
<td>$\text{length} - 2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(\text{length} - 2)</td>
<td>2</td>
</tr>
</tbody>
</table>
We derive the total number of comparisons by counting the number of times the inner loop is executed.

For each execution of the outer loop, the inner loop is executed \( \text{length} - \text{start} \) times.

The variable \textbf{length} is the size of the array. Replacing \textbf{length} with \( N \), the array size, the sum is derived as...

\[
N + (N - 1) + (N - 2) + \cdots + 2
\]

\[
= \sum_{i=2}^{N} i = \left( \sum_{i=1}^{N} i \right) - 1 = \frac{N(N+1)}{2} - 1
\]

\[
= \frac{N^2 + N - 2}{2} \approx N^2
\]
Bubble Sort

- With selection sort, we make one exchange at the end of one pass.
- Bubble sort improves the performance by making more than one exchange during each pass.
- By making multiple exchanges, we will be able to move more elements toward their correct positions using the same number of comparisons as the selection sort makes.
- The key idea of the bubble sort is to make pairwise comparisons and exchange the positions of the pair if they are out of order.
One Pass of Bubble Sort

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>17</td>
<td>5</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
</tbody>
</table>
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8

23 17 5 90 12 44 38 84 77
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8

23 17 5 90 12 44 38 84 77

exchange

17 23 5 90 12 44 38 84 77
One Pass of Bubble Sort

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>17</td>
<td>5</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
</tbody>
</table>

exchange

| 17| 23| 5 | 90| 12| 44| 38| 84| 77|

exchange
One Pass of Bubble Sort

```
0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77
   ^  ^
   exchange

17 23 5 90 12 44 38 84 77
   ^  ^
   exchange

17 5 23 90 12 44 38 84 77
```
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77

exchange

17 23 5 90 12 44 38 84 77

exchange

17 5 23 90 12 44 38 84 77
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77

↑↑ exchange

17 23 5 90 12 44 38 84 77

↑↑ exchange

17 5 23 90 12 44 38 84 77

↑↑ ok
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77

exchange

17 23 5 90 12 44 38 84 77

exchange

17 5 23 90 12 44 38 84 77

ok
One Pass of Bubble Sort

```
0  1  2  3  4  5  6  7  8
23 17  5 90 12 44 38 84 77
```

```
17 23  5 90 12 44 38 84 77
```

```
17  5 23 90 12 44 38 84 77
```

```
17  5 23 12 90 44 38 84 77
```

exchange
exchange
ok
exchange
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77

1 7 5 23 12 44 90 38 84 77

exchange

17 23 5 90 12 44 38 84 77

exchange

17 5 23 90 12 44 38 84 77

exchange

17 5 23 12 90 44 38 84 77

ok
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77

exchange

17 23 5 90 12 44 38 84 77

exchange

17 5 23 90 12 44 38 84 77

ok exchange

17 5 23 12 90 44 38 84 77

exchange

17 5 23 12 44 90 38 84 77

17 5 23 12 44 38 90 84 77

17 5 23 12 44 38 90 84 77
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77

17 23 5 90 12 44 38 84 77

17 23 5 90 12 44 38 84 77

17 5 23 90 12 44 38 84 77

17 5 23 90 12 44 38 84 77

17 5 23 12 44 38 90 84 77

17 5 23 12 44 38 90 84 77

exchange

exchange

exchange

exchange

ok
One Pass of Bubble Sort

<table>
<thead>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>17</td>
<td>5</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td>exchange</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>23</td>
<td>5</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td>exchange</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>23</td>
<td>90</td>
<td>12</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td>exchange</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>23</td>
<td>12</td>
<td>90</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>23</td>
<td>12</td>
<td>90</td>
<td>44</td>
<td>38</td>
<td>84</td>
<td>77</td>
</tr>
</tbody>
</table>
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77

exchange

17 23 5 90 12 44 38 84 77

exchange

17 5 23 90 12 44 38 84 77

ok exchange

17 5 23 12 90 44 38 84 77
One Pass of Bubble Sort

0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77

exchange

17 23 5 90 12 44 38 84 77

exchange

17 5 23 90 12 44 38 84 77

exchange

17 5 23 12 44 38 90 84 77

exchange

17 5 23 12 44 38 90 84 77

exchange

17 5 23 12 44 38 90 84 77

exchange

17 5 23 12 44 38 90 84 77

exchange

17 5 23 12 44 38 90 84 77

exchange
One Pass of Bubble Sort

The largest value 90 is at the end of the list.
One Pass of Bubble Sort

The largest value 90 is at the end of the list.
public void bubbleSort(int[] number) {
    int temp, bottom, i;
    boolean exchanged = true;
    bottom = number.length - 2;
    while (exchanged) {
        exchanged = false;
        for (i = 0; i <= bottom; i++) {
            if (number[i] > number[i+1]) {
                temp = number[i]; //exchange
                number[i] = number[i+1];
                number[i+1] = temp;
                exchanged = true; //exchange is made
            }
        }
        bottom--;
    }
}

Bubble Sort Performance

- In the worst case, the outer while loop is executed $N-1$ times for carrying out $N-1$ passes.
- For each execution of the outer loop, the inner loop is executed $\text{bottom}+1$ times. The number of comparisons in each successive pass is $N-1$, $N-2$, $\ldots$, $1$. Summing these will result in the total number of comparisons.
Bubble Sort Performance

- In the worst case, the outer while loop is executed N-1 times for carrying out N-1 passes.
- For each execution of the outer loop, the inner loop is executed \( \text{bottom} + 1 \) times. The number of comparisons in each successive pass is N-1, N-2, \( \ldots \), 1. Summing these will result in the total number of comparisons.

\[
(N - 1) + (N - 2) + \cdots + 1 = \sum_{i=1}^{N-1} i = \frac{N(N - 1)}{2} = \frac{N^2 - N}{2} \approx N^2
\]
Bubble Sort Performance

- In the worst case, the outer while loop is executed N-1 times for carrying out N-1 passes.
- For each execution of the outer loop, the inner loop is executed \( \text{bottom} + 1 \) times. The number of comparisons in each successive pass is N-1, N-2, \( \ldots \), 1. Summing these will result in the total number of comparisons.

- So the performances of the bubble sort and the selection sort are approximately equivalent. However, on the average, the bubble sort performs much better than the selection sort because it can finish the sorting without doing all N-1 passes.
Quicksort

To sort an array from index low to high, we first select a pivot element $p$.
- Any element may be used for the pivot, but for this example we will use number[low].

Move all elements less than the pivot to the first half of an array and all elements larger than the pivot to the second half. Put the pivot in the middle.

Recursively apply quicksort on the two halves.
Any element can be used as a pivot. For simplicity, we use number[low] as pivot.
public void quickSort(int[] number, int low, int high) {
    if (low < high) {
        int mid = partition(number, low, high);
        quickSort(number, low, mid - 1);
        quickSort(number, mid + 1, high);
    }
}
Quicksort Performance
Quicksort Performance

- In the worst case, quicksort executes roughly the same number of comparisons as the selection sort and bubble sort.
Quicksort Performance

- In the worst case, quicksort executes roughly the same number of comparisons as the selection sort and bubble sort.
- On average, we can expect a partition process to split the array into two roughly equal subarrays.
QuickSort Performance

- In the worst case, quicksort executes roughly the same number of comparisons as the selection sort and bubble sort.
- On average, we can expect a partition process to split the array into two roughly equal subarrays.

<table>
<thead>
<tr>
<th>Level No.</th>
<th>Number of subarrays at Level i</th>
<th>Size of each subarray at Level i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$1 = 2^0$</td>
<td>$N = N/2^0$</td>
</tr>
<tr>
<td>1</td>
<td>$2 = 2^1$</td>
<td>$N/2 = N/2^1$</td>
</tr>
<tr>
<td>2</td>
<td>$4 = 2^4$</td>
<td>$N/4 = N/2^2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$K$</td>
<td>$N$</td>
<td>$1 = N/2^K$</td>
</tr>
</tbody>
</table>
When recursive algorithms are designed carelessly, it can lead to very inefficient and unacceptable solutions.

For example, consider the following:

```java
public int fibonacci(int N) {
    if (N == 0 || N == 1) {
        return 1;
    } else {
        return fibonacci(N-1) + fibonacci(N-2);
    }
}
```
Excessive Repetition

- Recursive Fibonacci ends up repeating the same computation numerous times.

\[
\text{fibonacci}(5)
\]
Excessive Repetition

Recursive Fibonacci ends up repeating the same computation numerous times.

```
fibonacci(5)  \rightarrow  fibonacci(4)+fibonacci(3)
```
Excessive Repetition

- Recursive Fibonacci ends up repeating the same computation numerous times.

```
fibonacci(5)
fibonacci(4)+fibonacci(3)
fibonacci(2)+fibonacci(1)
```
Excessive Repetition

Recursive Fibonacci ends up repeating the same computation numerous times.

\[
\text{fibonacci(5)} = \text{fibonacci(4)} + \text{fibonacci(3)} = \text{fibonacci(2)} + \text{fibonacci(1)} = \text{fibonacci(1)} + \text{fibonacci(0)}
\]
Excessive Repetition

Recursive Fibonacci ends up repeating the same computation numerous times.
public int fibonacci( int N ) {
    int fibN, fibN1, fibN2, cnt;
    if (N == 0 || N == 1) {
        return 1;
    } else {
        fibN1 = fibN2 = 1;
        cnt = 2;
        while (cnt <= N) {
            fibN = fibN1 + fibN2; //get the next fib no.
            fibN1 = fibN2;
            fibN2 = fibN;
            cnt ++;
        }
        return fibN;
    }
}

Overhead of recursion

- Remember that each recursive call is expensive
  - create local variables for each call
  - copy arguments into local variables
  - track the execution point for each call
  - returned values are copied back
  - local space is reclaimed
When Not to Use Recursion

In general, use recursion if

- A recursive solution is natural and easy to understand.
- A recursive solution does not result in excessive duplicate computation.
- The equivalent iterative solution is too complex.