Searching And Sorting

- In this section of notes you will learn about different search and sort algorithms

How To Succeed In This Course

- Practice things yourself.
  - Write programs (attempt all assignments)
  - Trace lots of code

Leonardo da Vinci

J.R.R. Tolkien

Bruce Lee

Amadeus Mozart

Wayne Gretzky

Read Up The Material Ahead Of Time

- Read through the relevant parts of the book
- Trace through code examples

Review: References Vs. Objects

- Java: Only references to objects are declared:

```
class JavaClass
{
    private int num;
    public JavaClass () { num = 0; }
    public int get () { return num; }
    public void set (int newNum) { num = newNum; }
}

class Driver
{
    public static void main (String [] args)
    {
        JavaClass aReference = new JavaClass ();
    }
}
```

Review: References Vs. Objects (2)

- C++: Instances of objects can be directly declared without references or dynamic memory allocation.

```
class CPPClass
{
  private:
     int num;
  public:
     CPPClass () { num = 0; }
     int get () { return num; }
     void set (int newNum) { num = newNum; }
};

int
main ()
{
  CPPClass anObject;
}
```

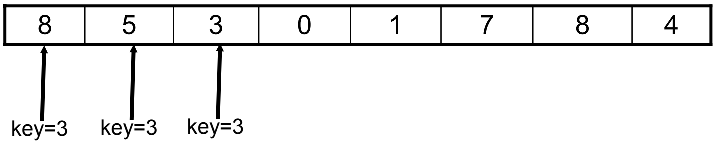
Typical Searching And Sorting Operations

- Swaps
- Moves
- Compares

Search Algorithms

1. Sequential search
2. Binary search
3. Interpolation search

Sequential Search



- Very simple to implement
- Not the most efficient algorithm
- Also know as a linear search

An Example Sequential Search: The Driver Class

- The full example can be found in the directory:

  /home/331/tamj/examples/searchingSorting/searching/sequential

```
class Driver
{
    public static void main (String [] args)
    {
        Search s1;
        int key;
        int size;
        int index;
```

The Driver Class (2)

```
        if (args.length == 1)
        {
            if (args[0].equals("-on"))
            {
                Debug.on = true;
            }
            else if (args[0].equals("-off"))
            {
                Debug.on = false;
            }
            else
            {
                System.out.println("Usage:  java Driver [-on/-
                   off]");
                System.exit(-1);
            }
        }
        else if (args.length > 1)
        {
            System.out.println("Usage:  java Driver [-on/-off]");
            System.exit(-1);
        }
```

The Driver Class (3)

```
        System.out.print("Enter size of list: ");
        size = Console.in.readInt();
        Console.in.readLine();
        if (size < 2)
             s1 = new Search ();
        else
             s1 = new Search(size);

        System.out.print("Enter value to search list for: ");
        key = Console.in.readInt();
        Console.in.readLine();
        System.out.println();

        index = s1.sequential(key);
```

Driver (4)

```
        if (index == -1)
            System.out.println("Element not found in list");
        else
            System.out.println("Index of first match: " + index);

        if (Debug.on == true)
            s1.display();
    }
}
```

Class Search

```
public class Search
{
    private int [] list;
    private Random generator;
```

Class Search (2)

```
    public Search ()
    {
        int i;
        int size = 100;
        list = new int[size];
        generator = new Random ();

        for (i = 0; i < list.length; i++)
        {
           list[i] = generator.nextInt(size+1);
        }
    }
```

Class Search (3)

```
public Search (int aSize)
{
    int i;
    char answer;
    list = new int[aSize];
    System.out.print("Enter in values manually (y/n)? ");
    answer = (char) Console.in.readChar();
    Console.in.readLine();
    if ((answer == 'Y') || (answer == 'y'))
    {
        for (i = 0; i < aSize; i++)
        {
            System.out.print("Enter value for element: ");
            list[i] = Console.in.readInt();
            Console.in.readLine();
        }
    }
```

Class Search (4)

```
    else
    {
        for (i = 0; i < list.length; i++)
        {
            list[i] = i;
        }
    }
}
```

Class Search (5)

```
    public void display ()
    {
        int i;
        for (i = 0; i < list.length; i++)
        {
            System.out.println("Element["+ i + "]=" + list[i]);
            if (((i % 20) == 0) && (i != 0))
            {
                System.out.println("Hit return to continue");
                Console.in.readLine();
            }
        }
    }
```

Class Search (6)

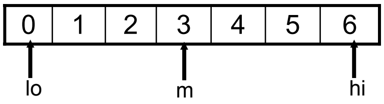
```
    public int sequential (int key)
    {
        int i;
        for (i = 0; i < length; i++)
        {
            if (key == list[i])
            {
                System.out.println("Time: " + (i+1));
                return i;
            }
        }
        System.out.println("Time: " + i);
        return -1;
    }
}
```

Analysis Of The Sequential Search

- Best case time: *O*(1)
- Worse case time: *O*(n)
- Average time: n/2 which is *O*(n)

Binary Search

Key = 0

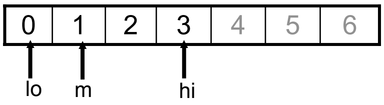


0	1	2	3	4	5	6

0 > 3?

Binary Search

Key = 0



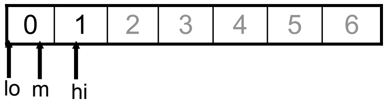
0	1	2	3	4	5	6

lo m hi

0 > 1?

Binary Search

Key = 0



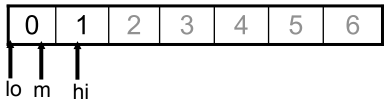
0	1	2	3	4	5	6

lo m hi

0 > 0?

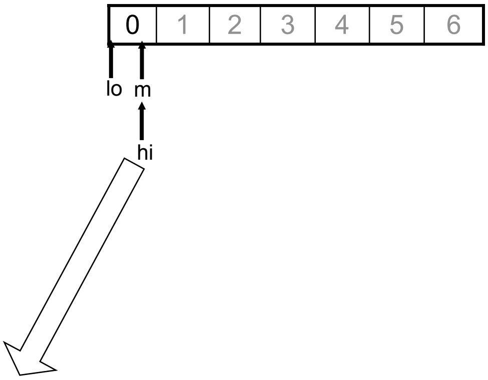
Binary Search

Key = 0



Binary Search

Key = 0



Return index of match (0)

Binary Search

- A little tougher to implement than a sequential search
- More efficient than a sequential search but the data must first be sorted.

An Example Binary Search: The Driver Class

- The full example can be found in the directory:

  /home/331/tamj/examples/searchingSorting/searching/binary

```
class Driver
{
    public static void main (String [] args)
    {
        Search s1;
        int key;
        int size;
        int index;
```

An Example Binary Search: The Driver Class (2)

```
        System.out.print("Enter size of list: ");
        size = Console.in.readInt();
        Console.in.readLine();
        if (size < 2)
             s1 = new Search ();
        else
             s1 = new Search(size);
        System.out.print("Enter value to search list for: ");
        key = Console.in.readInt();
        index = s1.binary(key);
        if (index == -1)
            System.out.println("Element not found in list");
        else
            System.out.println("Index of first match: " + index);
     }
}
```

An Example Binary Search: The Search Class

```
public class Search
{
    private int [] list;
    private Random generator;
    public Search ()
    {
        int i;
        int size = 100;
        list = new int[size];
        for (i = 0; i < list.length; i++)
        {
            list[i] = i;
        }
    }
```

An Example Binary Search: The Search Class (3)

```
public Search (int aSize)
{
    int i;
    char answer;
    list = new int[aSize];
    System.out.print("Enter values manually (y/n)? ");
    answer = (char) Console.in.readChar();
    Console.in.readLine();

    if ((answer == 'Y') || (answer == 'y'))
    {
        for (i = 0; i < aSize; i++)
        {
            System.out.print("Enter value: ");
            list[i] = Console.in.readInt();
            Console.in.readLine();
        }
    }
```

An Example Binary Search: The Search Class (4)

```
    else
    {
        for (i = 0; i < list.length; i++)
        {
            list[i] = i;
        }
    }
}
```

An Example Binary Search: The Search Class (5)

```
    public int binary (int key)
    {
        int high = list.length-1;
        int low = 0;
        int middle = 0;
        int time = 0;
        while (high > low)
        {
            time = time + 1;
            middle = (high + low) / 2;
            if (key > list[middle])
            {
                 low = middle + 1;
            }
            else
            {
                 high = middle;
            }
        }
```

An Example Binary Search: The Search Class (6)

```
        if (key == list[high])
            return high;
        else
            return -1;
    }
}
```

Analysis Of The Binary Search

- Best case time: $O(\log_2 n)$
- Worse case time: $O(\log_2 n)$
- Average time: which is $O(\log_2 n)$

- However a binary search requires that the list is already sorted.

A More Efficient Binary Search?

```
        while (high > low)
        {
            time = time + 1;
            middle = (high + low) / 2;
            if (key == list[middle])
                return middle;
            else if (key > list[middle])
                low = middle + 1;
            else
                high = middle;
         }
         if (key == list[high])
            return high;
        else
            return -1;
    }
}
```

Recap Of The Search Algorithms

Sequential search

Binary search

Discard half

I don't need this half of the book. I'll just throw it away.

Example: Looking for "Television repairs"

Another Search Algorithm

- Premise: If you know that the data is sorted you may attempt a better "guess" as to what portion of the list should be discarded.
- The guess is based on how closely the key lies within the range of the largest vs. smallest element.
- Example: Looking for "Television repairs"

"If Ya Gotta Go... Go With Us!"
D B PORT-A-PODI TOILET RENTALS
SMITHRITE
ALLIED TOOLS LTD
243-6000

An Example Interpolation Search: The Driver Class

- The full example can be found in the directory:

  /home/331/tamj/examples/searchingSorting/searching/interpolation

```
class Driver
{
    public static void main (String [] args)
    {
        System.out.print("Enter size of list: ");
        size = Console.in.readInt();
        Console.in.readLine();
        if (size < 2)
            s1 = new Search ();
        else
            s1 = new Search(size);
```

The Driver Class (2)

```
     System.out.print("Enter value to search list for: ");
     key = Console.in.readInt();
     Console.in.readLine();
     System.out.println();

     index = s1.interpolation(key);
     if (index == -1)
        System.out.println("Element not found in list");
     else
        System.out.println("Index of first match: " +
           index);
     if (Debug.on == true)
        s1.display();
    }
}
```

The Search Class

```
public class Search
{
    private int [] list;
    private int size;
    private Random generator;
```

The Search Class (2)

```
    public int interpolation (int key)
    {
        int high = list.length-1;
        int low = 0;
        int middle = 0;
        float weight;
        int time = 0;
        while ((list[high] >= key) && (key > list[low]))
        {
          weight = (float)(key - list[low])/(float)(list[high] -
             list[low]);
           middle = Math.round(weight * (high - low)) + low;
           if (key > list[middle])
               low = middle + 1;
           else if (key < list[middle])
              high = middle - 1;
           else
                low = middle;
        }
```

The Search Class (3)

```
   if (key == list[low])
            return low;
        else
            return -1;
    }
}
```

Analysis Of The Interpolation Search

- Best case time: $O(1)$
- Worse case time: $O(n)$
- Average time: ~$O(\log_2(\log_2 n))$

Sorting Algorithms

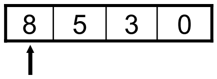
- Simple sorting algorithms
  1) Bubble sort (also known as an exchange sort)
  2) Selection sort
  3) Insertion sort
- Complex sorting algorithms
  1) Merge sort
  2) Quick sort
  3) Shell sort

Technical Constraints

- Size of the data set
  - Internal sorts
  - External sorts
- Comparisons and swaps
  - Is the data a simple type (small in size) or is it composite (larger)

Bubble Sort Algorithm

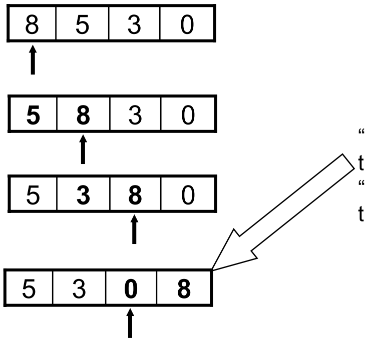
Flag = true



8	5	3	0

Bubble Sort Algorithm

Flag = false



8	5	3	0
5	**8**	3	0
5	**3**	**8**	0
5	3	**0**	**8**

“Bubble sort” because the largest items will “bubble” over towards the end.

Bubble Sort: Description Of The Algorithm

- Start by assuming the list is sorted.
- Starting with the first element compare each element with its right hand neighbor.
- If elements are out of order swap them and set the flag to false.
- Continue comparisons until the second last element is reached.
- If the flag is true then return the beginning of the list, set the flag to false (assume that this time its sorted until proven otherwise) and check the order again otherwise you are done.
- While the largest item moves to the end, each of the smaller elements will have moved one step closer to the front of the list.

An Example Of A Bubble Sort: Class Driver

- The full example can be found in the directory:

/home/331/tamj/examples/searchingSorting/sorting/bubble

```
class Driver
{
    public static void main (String [] args)
    {
        Sort s1 = null;
        if (args.length == 0)
        {
            int size;
            System.out.print("Enter size of list: ");
            size = Console.in.readInt();
            Console.in.readLine();
```

An Example Of A Bubble Sort: Class Driver (2)

```
        if (size < 2)
        {
            s1 = new Sort ();
            s1.randomValues();
        }
        else
        {
            s1 = new Sort(size);
            s1.randomValues ();
        }
    }
```

An Example Of A Bubble Sort: Class Driver (3)

```
    else if (args.length == 2)
    {
         if (args[0].equals("-on"))
            Debug.on = true;
        else if (args[0].equals("-off"))
            Debug.on = false;
        else
            System.exit(-1);
        if (args[1].equals("-b"))
        {
            s1 = new Sort(8);
            s1.bestValues(8);
        }
        else if (args[1].equals("-w"))
        {
            s1 = new Sort(8);
            s1.worstValues(8);
        }
```

An Example Of A Bubble Sort: Class Driver (4)

```
            else if (args[1].equals("-a"))
            {
                s1 = new Sort (8);
                s1.averageValues(8);
            }
            else
            {
                System.out.print("Second argument must be '-b',
                   'w'");
                System.out.println("or '-a'");
                System.exit(-1);
            }
        }
         else
              System.exit(-1);
        s1.bubble();
        s1.display();
    }
}
```

An Example Of A Bubble Sort: Class Debug

```
public class Debug
{
    public static boolean on = false;
}
```

An Example Of A Bubble Sort: Class Sort

```
public class Sort
{
    private int [] list;
    public static final int MAX = 100;

    public Sort ()
    {
        int size = 10;
        list = new int [size];
    }

    public Sort (int newSize)
    {
        list = new int [newSize];
    }
```

An Example Of A Bubble Sort: Class Sort (2)

```
    public void randomValues ()
    {
        Random generator;
        int i;
        generator = new Random ();
        for (i = 0; i < list.length; i++)
             list[i] = generator.nextInt(Sort.MAX)+1;
    }

    public void bestValues (int newSize)
    {
        int i;
        for (i = 0; i < newSize; i++)
        {
            list[i] = i;
        }
    }
```

An Example Of A Bubble Sort: Class Sort (3)

```
public void worstValues (int newSize)
{
    int i;
    for (i = 0; i < newSize; i++)
    {
        list[i] = newSize - i;
    }
}

public void averageValues (int newSize)
{
    int i;
    for (i = 0; i < (newSize/2); i++)
        list[i] = newSize - i;
    for (; i < newSize; i++)
        list[i] = i;
}
```

An Example Of A Bubble Sort: Class Sort (4)

```
public void inputValues ()
{
    int i;
    for (i =0; i < list.length; i++)
    {
        System.out.print("Enter value for list element: ");
        list[i] = Console.in.readInt();
        Console.in.readLine();
    }
}
```

An Example Of A Bubble Sort: Class Sort (5)

```
public void bubble ()
{
    boolean isSorted;
    int i;
    int j;
    int temp;
    j = list.length-1;
     isSorted = false;
```

An Example Of A Bubble Sort: Class Sort (6)

```
    while ((isSorted == false) && (j > 0))
    {
        if (Debug.on == true)
            displayDebug();
       isSorted = true;
       for (i = 0; i < j; i++)
       {
           if (list[i] > list[i+1])
           {
               temp = list[i];
               list[i] = list[i+1];
               list[i+1] = temp;
               isSorted = false;
           }
       }
       j--;
    }
}
```

An Example Of A Bubble Sort: Class Sort (7)

```
public void displayDebug ()
{
    int i;
    for (i = 0; i < list.length; i++)
        System.out.print(list[i] + " ");
    System.out.println();
}
```

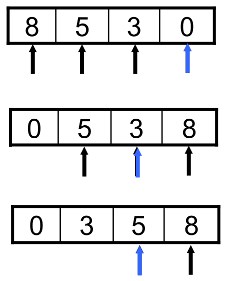
An Example Of A Bubble Sort: Class Sort (8)

```
        public void display ()
        {
            int i;
            char answer;

            for (i = 0; i < list.length; i++)
            {
                System.out.println("Element["+ i + "]=" +
                   list[i]);
                if (((i % 20) == 0) && (i != 0))
                {
                    System.out.println("Hit return to continue or
                        'q' to quit");
                    answer = (char) Console.in.readChar();
                    if ((answer == 'q') || (answer == 'Q'))
                        return;
                }
            }
        }
    }
}
```

Analysis Of The Bubble Sort

- Worse case: $O(n^2)$
- Best case: $O(n)$
- Average case: $O(n^2)$

Selection Sort Algorithm



Insert in 1st spot — 8 | 5 | 3 | 0 — Index of smallest = 3

Insert in 2nd spot — 0 | 5 | 3 | 8 — Index of smallest = 2

Insert in 3rd spot — 0 | 3 | 5 | 8 — Index of smallest = 2

Selection Sort: Description Of The Algorithm

- Start off by trying to put the smallest element in the first spot.
- Assume that the element at this spot is smallest and compare it with the second element. Track the index of the smaller of the two.
- Compare the smaller of the two with the third element and update the variable that tracks the index of the smallest element.
- Continue comparing elements until the end of the list is reached.
- If the smallest element isn't already in the first spot then swap the element which is currently at the beginning of the list with the smallest element.
- Repeat this with the second element in the array (find the second smallest element and put it here).
- Repeat this with the third element in the array (find the third smallest element and put it here).
- Continue moving elements to the appropriate place in the array until you have reached the last element.
- Stop, don't bother trying to swap anything into the last spot because at this point you would have already swapped all previous elements into the correct spot so by default the last element is already in its proper place.

An Example Of An Selection Sort: Class Driver

- The full example can be found in the directory:

/home/331/tamj/examples/searchingSorting/sorting/selection

```
class Driver
{
    public static void main (String [] args)
    {
        Sort s1 = null;
        s1.selection();
        s1.display();
    }
}
```

An Example Of An Selection Sort: Class Sort

```
public class Sort
{
    private int [] list;
    public static final int MAX = 100;

    public Sort ()
    {
        int size = 10;
        list = new int [size];
    }

    public Sort (int newSize)
    {
        list = new int [newSize];
    }
```

An Example Of An Selection Sort: Class Sort (2)

```
    public void randomValues ()
    {
        Random generator;
        int i;
        generator = new Random ();
        for (i = 0; i < list.length; i++)
              list[i] = generator.nextInt(Sort.MAX)+1;
    }

    public void bestValues (int newSize)
    {
        int i;
        for (i = 0; i < newSize; i++)
        {
            list[i] = i;
        }
    }
```

An Example Of An Selection Sort: Class Sort (3)

```
public void worstValues (int newSize)
{
    int i;
    for (i = 0; i < newSize; i++)
         list[i] = i + 1;
    list[0] = newSize;
}

public void averageValues (int newSize)
{
    int i;
    int j;
    for (i = 0; i < newSize; i++)
        list[i] = newSize - i;
}
```

An Example Of An Selection Sort: Class Sort (4)

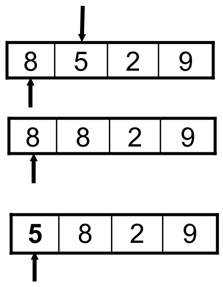
```
public void selection ()
{
    int i;
    int j;
    int minIndex;
    int temp;
    for (i = 0; i < (list.length-1); i++)
    {
        minIndex = i;
        for (j = (i+1); j < list.length; j++)
        {
             if (list[j] < list[minIndex])
                minIndex = j;
        }
        if (minIndex != i)
        {
            temp = list[i];
            list[i] = list[minIndex];
            list[minIndex] = temp;
        }
    }
}
```

Analysis Of The Selection Sort

- All cases: $O(n^2)$

The Insertion Sort Algorithm

Checking second element



5 < 8?

8	5	2	9

Item to insert = 5

Start of list?

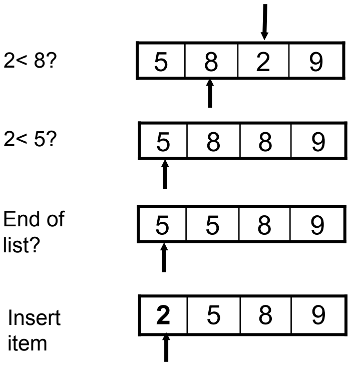
8	8	2	9

Insert item

5	8	2	9

The Insertion Sort Algorithm (2)

Checking third element



2< 8?

5	8	2	9

2< 5?

5	8	8	9

End of list?

5	5	8	9

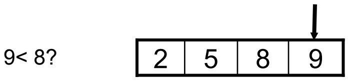
Insert item

2	5	8	9

Item to insert = 2

The Insertion Sort Algorithm (3)

Checking fourth element



9< 8?

2	5	8	9

Description Of The Algorithm: The Insertion Sort

- Check if there is one element in the list, if there is then we are done.
- If not then move onto the second element in the list.
  - Determine if the second element needs to be inserted into it's proper place near the beginning of the list.
  - If so
    - Shift all the elements that precede the second element to the "right" by one (towards the end of the list).
    - Insert the contents of second element in the appropriate place
    - Move onto the third element
  - If not move onto the third element

Description Of The Algorithm: The Insertion Sort (2)

- Move onto the third element in the list.
  - Determine if the third element needs to be inserted into it's proper place near the beginning of the list.
  - If so
    - Shift all the elements that precede the third element to the "right" by one (towards the end of the list).
    - Insert the contents of third element in the appropriate place
    - Move onto the fourth element
  - If not move onto the fourth element.
- Repeat until the end of the list has been reached.

An Example Of An Selection Sort: Class Driver

- The full example can be found in the directory:

/home/331/tamj/examples/searchingSorting/sorting/insertion

```
class Driver
{
    public static void main (String [] args)
    {
        Sort s1 = null;
        s1.insertion();
        s1.display();
    }
}
```

Class Sort

```
public class Sort
{
    private int [] list;
    public static final int MAX = 100;

    public Sort ()
    {
        int size = 10;
        list = new int [size];
    }

    public Sort (int newSize)
    {
        list = new int [newSize];
    }
```

Class Sort (2)

```
public void bestValues (int newSize)
{
    int i;
    for (i = 0; i < newSize; i++)
    {
        list[i] = i + 1;
    }
}

public void worstValues (int newSize)
{
    int i;
    for (i = 0; i < newSize; i++)
    {
        list[i] = newSize - i;
    }
}
```

Class Sort (3)

```
public void averageValues (int newSize)
{
    int i;
    int j;
    for (i = 0; i < (newSize/2); i++)
    {
        list[i] = i + 1;
    }
    j = 0;
    while (i < newSize)
    {
        list[i] = newSize - j;
        i++;
        j++;
    }
}
```

Class Sort (4)

```
public void insertion ()
{
    int i, j;
    int itemToInsert;
    for (i = 1; i < list.length; i++)
    {
        if (list[i] < list[i-1])
        {
             itemToInsert = list[i];
             j = i - 1;
             while (j >= 0)
             {
                   list[j+1] = list[j];
                   if ((j == 0) ||
                       (list[j-1] <= itemToInsert))
                       break;
                   else
                      j--;
             } // Inner while loop
```

Class Sort (5)

```
            list[j] = itemToInsert;
          } // if condition (elements out of order)
    } // outer for loop
} // end of method
```

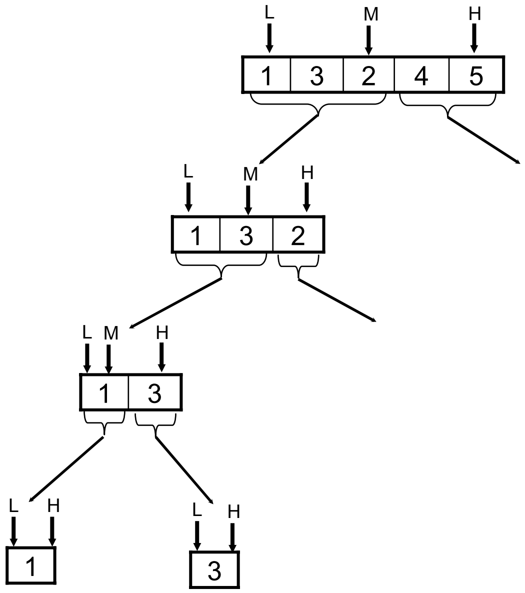
Analysis Of The Insertion Sort

- Best case: O(n)
- Worse case: $O(n^2)$
- Average case: $O(n^2)$

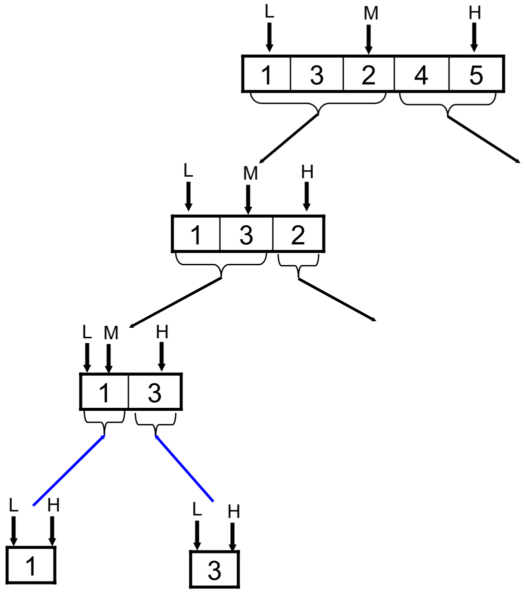
Divide And Conquer

- Split the problem into sub-problems (through recursive calls)
- Continue splitting each of the sub-problems into smaller parts until you cannot split the problem up any further
- Solve each of the sub-problems separately and combine the solutions yielding the solution to the original problem

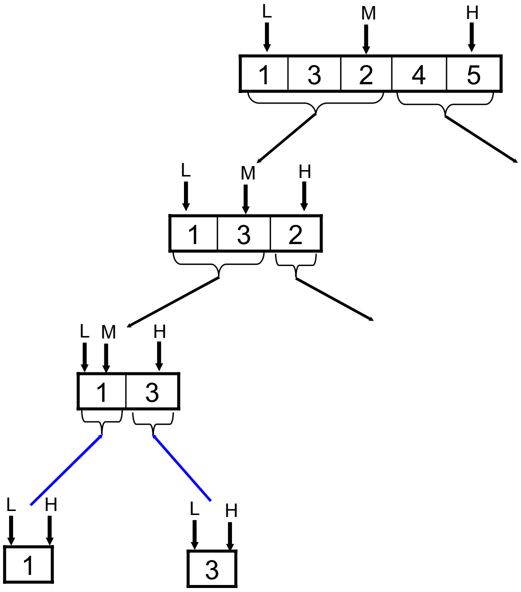
Merge Sort Algorithm



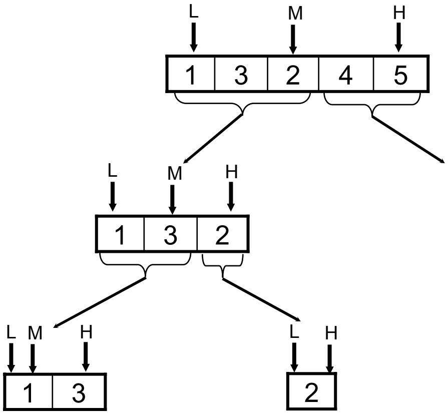
Merge Sort Algorithm



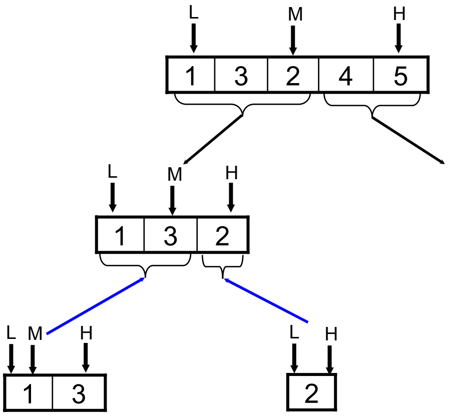
Merge Sort Algorithm



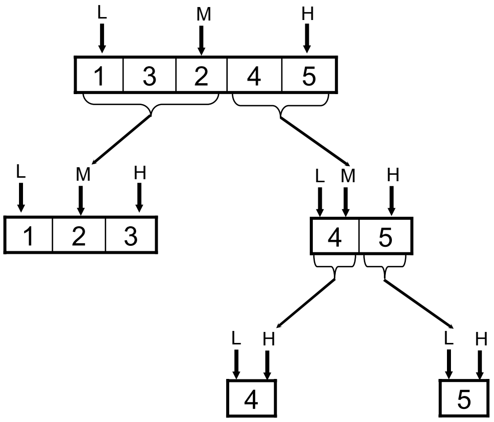
Merge Sort Algorithm



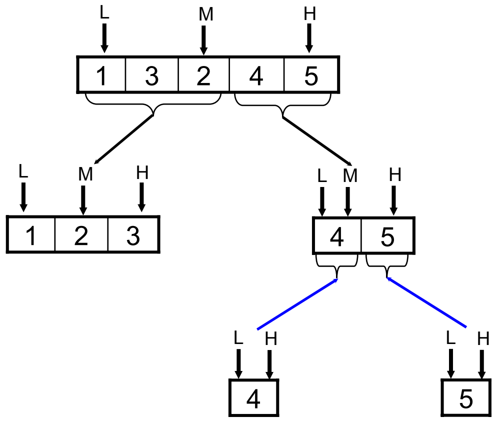
Merge Sort Algorithm



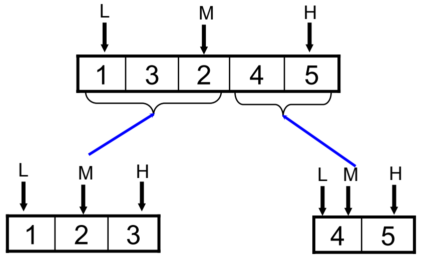
Merge Sort Algorithm



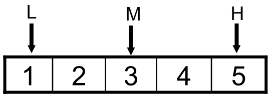
Merge Sort Algorithm



Merge Sort Algorithm



Merge Sort Algorithm



Merge Sort: Description Of The Algorithm

- Recursively divide the array into a left and right sub-array.
- Each sub-array is then divided further into smaller left and right sub-arrays until a sub-array consists of only a single element at which point the recursive call stops.
- When the recursive call stops for the left and right sub-array combine them by copying their contents, in order, into a temporary array.
- When the contents of the temporary arrays are copied back to the original list the sort is complete because all the elements are now in order.

An Example Of A Merge Sort: Class Driver

- The full example can be found in the directory:

/home/331/tamj/examples/searchingSorting/sorting/merge

```
class Driver
{
    public static void main (String [] args)
    {
        Sort s1 = null;
        s1.startSort();
        s1.display();
    }
}
```

Class Sort

```
public class Sort
{
    private int [] list;
    public static final int MAX = 100;
    private int time = 0;

    public Sort ()
    {
        int size = 10;
        list = new int [size];
    }

    public Sort (int newSize)
    {
        list = new int [newSize];
    }
```

Class Sort (2)

```
public void randomValues ()
{
    Random generator;
    int i;
    generator = new Random ();
    for (i = 0; i < list.length; i++)
    list[i] = generator.nextInt(Sort.MAX)+1;
}

public void bestValues (int newSize)
{
    int i;
    for (i = 0; i < newSize; i++)
    {
        list[i] = i;
    }
}
```

Class Sort (3)

```
public void worstValues (int newSize)
{
    int counter = 0;
    int left = 0;
    int right = newSize / 2;
    while (counter < newSize)
    {
        if (counter % 2 == 0)
        {
            list[left] = counter;
            left++;
        }
        else
        {
            list[right] = counter;
            right++;
        }
        counter++;
    }
}
```

Class Sort (4)

```
    public void averageValues (int newSize)
    {
        int i;

        if (newSize != 8)
            return;

        for (i = 0; i < (newSize/2); i++)
            list[i] = i * 2;

        list[4] = 3;
        list[5] = 7;
        list[6] = 5;
        list[7] = 9;
}
```

Class Sort (5)

```
    public void inputValues ()
    {
        int i;
        for (i =0; i < list.length; i++)
        {
            System.out.print("Enter value for list element: ");
            list[i] = Console.in.readInt();
            Console.in.readLine();
        }
    }

    public void startSort ()
    {
        mergeSort(0,list.length-1);
        System.out.println("Time units (based on compares): " +
             time);
    }
```

Class Sort (6)

```
public void mergeSort (int first, int last)
{
    if (first < last)
    {
        int middle = (first + last) / 2;
        mergeSort(first,middle);
        mergeSort(middle+1,last);
        merge(first,middle,last);
    }
}
```

Class Sort (7)

```
public void merge (int first, int middle, int last)
{
    int [] temp = new int[last-first+1];
    int first1 = first;
    int last1 = middle;
    int first2 = middle + 1;
    int last2 = last;
    int index = 0;
    while ((first1 <= last1) && (first2 <= last2))
    {
        if (list[first1] < list[first2])
        {
            temp[index] = list[first1];
            first1++;
        }
        else
        {
            temp[index] = list[first2];
            first2++;
        }
        index++;
    }
```

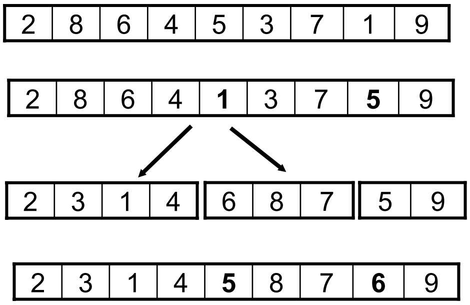
Class Sort (8)

```
        while (first1 <= last1)
        {
            temp[index] = list[first1];
            first1++;
            index++;
        }

        while (first2 <= last2)
        {
            temp[index] = list[first2];
            first2++;
            index++;
        }

        int listIndex = first;
        for (index = 0; index < temp.length; index++)
        {
            list[listIndex] = temp[index];
            listIndex++;
        }
}
```

Analysis Of The Merge Sort

- Best case: $O(n * \log_2 n)$
- Worse case: $O(n * \log_2 n)$
- Average case:$O(n * \log_2 n)$

Quick Sort Algorithm



2	8	6	4	5	3	7	1	9

2	8	6	4	**1**	3	7	**5**	9

2	3	1	4	6	8	7	5	9

2	3	1	4	**5**	8	7	**6**	9

Quick Sort: Description Of The Algorithm

- Another divide-and-conquer approach to sorting.
- Partition the array according to one element (called a “pivot”).
- Recursively rearrange the array according to the pivot so that elements before the pivot are smaller than the pivot and the elements after the pivot are larger than the pivot.
- The list is not completely sorted but sorted around the pivot (elements of the left sub array are smaller than the pivot while those on the right are larger).
- Divide the list into two sub lists according to the pivot and recursively sort each sub list by choose a new pivot for each.
- Stop the recursive calls when the list has been divided to some small size.

Choosing A Pivot

1. First element/last element
2. Median of three

An Example Of A Quick Sort: Class Driver

- The full example can be found in the directory:

/home/331/tamj/examples/searchingSorting/sorting/quick

```
class Driver
{
    public static void main (String [] args)
    {
        Sort s1 = null;
        s1.startSort();
        s1.display();
    }
}
```

Class Sort

```
public class Sort
{
    private int [] list;
    public static final int MAX = 100;
```

Class Sort (2)

```
    public void startSort ()
    {
        quick(0, list.length-1);
    }

    private void quick (int first, int last)
    {
        if (first == last)
            return;
        else if ((last - first) == 1)
        {
            if (list[first] > list[last])
            {
                int temp = list[first];
                list[first] = list[last];
                list[last] = temp;
             }
            return;
        }
```

Class Sort (3)

```
       else if ((last - first) == 2)
       {
           int middle = last - 1;

sortFirstMiddleLast(first,middle,last);
           return;
       }
    }
```

Class Sort (4)

```
private int partition (int first, int last)
{
    int middle = (first + last) / 2;
    sortFirstMiddleLast(first, middle, last);

    swap(middle, last-1);

    int pivotIndex = last-1;
    int pivot = list[pivotIndex];

    int indexFromLeft = first + 1;
    int indexFromRight = last - 2;

    boolean done = false;
```

Class Sort (5)

```
        while (done == false)
        {
            while (list[indexFromLeft] < pivot)
                indexFromLeft++;
            while (list[indexFromRight] > pivot)
                indexFromRight--;
            if (indexFromLeft < indexFromRight)
            {
                swap(indexFromLeft, indexFromRight);
                indexFromLeft++;
                indexFromRight--;
            }
             else
            {
               done = true;
            }
       }
       swap(pivotIndex, indexFromLeft);
       pivotIndex = indexFromLeft;
       return pivotIndex;
}
```

Class Sort (6)

```
 private void sortFirstMiddleLast(int first, int middle,
                                  int last)
{
    if (list[first] > list[middle])
        swap(first,middle);
    if (list[middle] > list[last])
        swap(middle,last);
    if (list[first] > list[middle])
        swap(first,middle);
}

private void swap (int firstIndex, int secondIndex)
{
    int temp = list[firstIndex];
    list[firstIndex] = list[secondIndex];
    list[secondIndex] = temp;
}
```

Analysis Of The Quick Sort

- Best case: $O(n * \log_2 n)$
- Worse case: $O(n^2)$
- Average case: $O(n * \log_2 n)$

You Should Now Know

- Common searching algorithms and their strengths and limitations
- Sorting algorithms
  - How are they implemented?
  - What are the best, worse and average cases?
  - What is their best, worst and average case speeds?
  - Strengths and weaknesses that go beyond algorithm analysis (i.e., considerations other than just Big-O).

Sources Of Material

- *Java: A Framework for Program Design and Data Structures* (2nd edition) by Kenneth A. Lambert and Martin Osborne
- *Data Structures and Abstractions with Java* by Frank M. Carrano and Walter Savitch.
- CPSC 331 course notes by Marina L. Gavrilova http://pages.cpsc.ucalgary.ca/~marina/331/