Search

- A note on arrays
- JDB, revisited - setting breakpoints
- Assignment 4, example
- Linear search
- Binary search
Arrays

• Declaring
  – C style (Java Gently): `<type> <id> [ ] ;`
  – Java style: `<type> [ ] <id> ;`

• Allocating a new array
  – C style:
    `<type> <id> [ ] = new <type> [ <limit> ]`
  – Java style:
    `<type> [ ] <id> = new <type> [ <limit> ]`

• It’s optional which style you use
jdb, Revisited

• Breakpoints work
  – Stop in <classname>.<method_name>
    – Causes execution to stop each time an invocation of that method occurs
  – Stop at <method_name>:<line_no>
    – Causes execution to stop at that line_no in that method
  – cont causes execution to resume until next breakpoint

• Type ? in JDB to see help on commands
jdb, Revisited

• Execution with breakpoints
  – **step** causes execution of next statement (can step by step through entire program)
  – **step up** continues execution until return to caller of current method
  – **clear <classname>.<method_name>**
    clear <method_name>:<line_no>
    both clear an already set breakpoint

• Breakpoint commands can be intermingled with other commands (e.g., list, locals, print)
Example

27 remus!assignment3-s98> jdb myTest
Initializing jdb...
0xee32b370:class(myTest)
> stop in Segment.pointOnSegment
Breakpoint set in Segment.pointOnSegment
> stop in Polygon.getPerimeter
Breakpoint set in Polygon.getPerimeter
> run
run myTest
running ...
main[1]
Breakpoint hit: Segment.pointOnSegment (Segment:230)
Example

```java
public boolean pointOnSegment(Point p){
    //first get endpoints of Segment
    double x1 = (this.getFirstPoint()).getX(),
    y1 = (this.getFirstPoint()).getY(),
    x2 = (this.getSecondPoint()).getX(),
    y2 = (this.getSecondPoint()).getY(),
    x3 = p.getX(),

    ...we continue to do list and watch execution of pointOnSegment...
    ...we could clear this breakpoint with clear Segment:230 or use step up
```
Example

main[1] step up //continues execution until reaches caller of this method
main[1]       //myTest.main; next breakpoint found is in getPerimeter

Breakpoint hit: Polygon.getPerimeter (Polygon:130)
main[1] list
126     //method to use an enumerator to calculate
127     //the perimeter of a Polygon object
128     //needs to use getLength() from Segment class
129     public double getPerimeter(){
130 =>    double perimeter = 0.0;
131    Enumeration edgeEnum = this.getEdges();
132    while (edgeEnum.hasMoreElements()){  
133        Segment seg = (Segment)edgeEnum.nextElement();
134        System.out.println(seg.toString() + "length= "
main[1] step //executes one statement at a time
main[1]
Breakpoint hit: Polygon.getPerimeter (Polygon:131)
```
public double getPerimeter()
{
    double perimeter = 0.0;

    Enumeration edgeEnum = this.getEdges();
    while (edgeEnum.hasMoreElements())
    {
        Segment seg = (Segment)edgeEnum.nextElement();
        System.out.println(seg.toString() + "length= " + seg.getLength());
    }
}
```
Assignment 4

• Given a set of Polygons, find the one with the closest vertex to the origin (0,0).

• Need a nested enumeration, one through the Set of Polygons, and for each Polygon, through each of its sides to find the corresponding vertices.

• Use Euclidean distance to compare points

\[ \text{dist} = x^2 + y^2 \]
Polygon p, psave = null; Point closest = null;
double Double.POSITIVE_INFINITY, dist1, dist2, p1X, p1Y, p2X, p2Y;
Enumeration polyenum = polys.elements();
while (polyenum.hasMoreElements()) {// extract polygon
    p = (Polygon) polyenum.nextElement();
    Enumeration sidesenum = p.getEdges();
    while (sidesenum.hasMoreElements()) {// extract side
        Segment ss = (Segment) sidesenum.nextElement();
        p1X = (ss.getFirstPoint()).getX();
        p1Y = (ss.getFirstPoint()).getY();
        p2X = (ss.getSecondPoint()).getX();
        p2Y = (ss.getSecondPoint()).getY();
        dist1 = p1X * p1X + p1Y * p1Y;
        dist2 = p2X * p2X + p2Y * p2Y;
        if (dist1 < d) {closest = ss.getFirstPoint();
            d = dist1; psave = p;}
        if (dist2 < d) {closest = ss.getSecondPoint();
            d = dist2; psave = p;}
    } // have found closest point on this polygon
    System.out.println("closest point to origin is " + closest + " on polygon " + psave);
}
Algorithm Complexity

• Constant number of operations in innermost loop
• Perform these once per side for each of k Polygons.
• Complexity of the nested loop over all will be proportional to
  Sum (over all Polygons) #sides of each Polygon
  so if all Polygons were triangles, it would be 3k.
• If sum all sides over all Polygons to obtain s sides, then work is proportional to constant*s.
Search

- Standard useful algorithm involves looking through a list of values for a particular value
- Can use arrays for this task
- Efficiency is important, especially for long lists
Linear Search I

• Search an unordered list of values for 0

5 2 7 3 4 9 2 0 1 7 //stored in array

```java
int desired = 0;
for (int i = 0; i < a.length; i++)
    if (a[i] == desired){
        System.out.println(desired + " found at position " + i);
        break f1;
    }
//may search entire array before know
//value not contained therein
```
Linear Search I

• If desired value not in array may have to search entire array to find out.

• If desired value in array it may be at the end so may have to search entire array to find it. Worst case

• If desired value in array, you may find it in the first element! Best case
Linear Search II

• Search an **ordered** list of values for 3
• 0 1 2 4 5 7 7 9 //stored in sorted array
  0 1 2 4 5 7 7 9
  0 1 2 4 5 7 7 9
  0 1 2 4 5 7 7 9 NOT FOUND!

• Proceed up from smallest value, comparing to desired value, until hit a value which is larger than the desired value.

• Don’t have to search entire array, unless desired value is bigger than all values in array or is largest value in array, **Worst case**

• In **Best case**, find value in first element.
Linear Search II

- Search an ordered list of values for 5
- 0 1 2 4 5 7 7 9 //stored in sorted array
  0 1 2 4 5 7 7 9
  0 1 2 4 5 7 7 9
  0 1 2 4 5 7 7 9
  0 1 2 4 5 7 7 9
  0 1 2 4 5 7 7 9 FOUND!
Linear Search II: Code

```java
int desired = 2;
for (int i = 0; i < a.length; i++) {
    if (desired < a[i]) break f1;
    else if (a[i] == desired) {
        System.out.println(desired + "found at element" + i);
        break f1;
    }
}
//only search until find number larger
//than desired
```
Worst Case Complexity

- Linear search I: n checks if desired not in unordered array of n values
  - On average, desired value could be anywhere in the array
- Linear search II: 2n checks, if desired is larger than the largest element;
  - On average, will check n/2 elements
- Is there a better way?
Twenty Questions

- The game allows you twenty questions to guess the number I’m thinking of between 1 and 1 million
- Suppose the chosen number is 445,362
  - [1:1,000,000]: 500,000 ⇒ lower
  - [1:500,000]: 250,000 ⇒ higher
  - [250,000:500,000]: 375,000 ⇒ higher
  - [375,000:500,000]: 437,500 ⇒ higher
  - [437,500:500,000]: 468,750 ⇒ lower
  - [437,500:468,750]: 453,125 ⇒ lower
  - [437,500:453,125]: 445,312 ⇒ higher
Twenty Questions

- Know the number is between 445,312 and 453,125 having asked only 7 questions!
- Each question eliminates half the possible numbers left.
- How many questions will it take?
  - How many times can 1,000,000 be divided by 2?
  - $2^{20} = 1,048,576$, so 20 questions suffice.
- Let’s use this idea to search an ordered list of numbers
Binary Search

Find 2 in the array, if it is there.

0 1 2 3 5 6 8 9
2 == 9? F; 2 == 0? F

0 1 2 3 4 5 6 7
indices in the array

0 1 2 3 5 6 8 9
2 == 3? F; 2 < 3? T

0 1 2 3 4 5 6 7

0 1 2 3 5 6 8 9
2 ==1? F; 2<1? F

0 1 2 3 4 5 6 7

0 1 2 3 5 6 8 9
2 ==2? T; found with index 2

0 1 2 3 4 5 6 7
Binary Search

Find index of 4, if it is there.

<table>
<thead>
<tr>
<th>0 1 2 3 5 6 8 9</th>
<th>4 == 9? F; 4 == 0? F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>indices in the array</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 1 2 3 5 6 8 9</th>
<th>4 == 3? F; 4 &lt; 3? F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 1 2 3 5 6 8 9</th>
<th>4 == 6? F; 4 &lt; 6 ? T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 1 2 3 5 6 8 9</th>
<th>4 == 5? F; 4 &lt; 5? T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>4 isn’t found</td>
</tr>
</tbody>
</table>
Complexity: Binary Search

- At each step divide number of numbers left to examine in half: \( j, \ j/2, \ j/4, \ j/8, \ j/16, ..., 1 \)
- Do 2 comparisons each step (== and then <)
- Stop when reach \( k \) such that
  - \( j/(2^k) == 1 \) or \( j == 2^k \) or
  - \( \log_2 j = k \)
- Will do in the worst case, \( \log_2 j \) comparisons if number is not in the list of \( j \) numbers.
// assume have read in a[], desired
// have set hi=a.length-1, low=0,
// mid=(a.length-1)/2
System.out.println("desired = " + desired);
int hi=a.length-1, low=0, mid=(hi+low)/2;
System.out.println("low,mid,hi " + low + " + " + mid + " " + hi);
// debugging output

// first checks ends of the array
if (desired == a[hi])
    {System.out.println(" found " + desired + " at " + hi);
     return;}
else if (desired == a[low]) {System.out.println(" found " + desired + " at " + low);
    return;}

newbinsearch.java
else  

    w1:

        w2: while (hi >= low) {

            if (desired == a[mid]){
                System.out.println(
                    " found at a[" + mid + "]");
                break w1;}
            
            else if (desired < a[mid]) hi = mid-1;
            else low = mid+1;

            mid = (hi+low)/2;
            System.out.println("low,mid,hi " +
            low + " " + mid + " " + hi);

        }

        System.out.println(desired + " not found");

    }
Output

8  remus!111>  !java
java BinarySearch
   Enter 8 numbers in nondecreasing order
Input an integer: 2 4 6 8 10 12 14 16
Input desired value 4
desired = 4
low,mid,hi 0 3 7
low,mid,hi 0 1 2
   found at a[1]
10  remus!111>  !java
java BinarySearch
   Enter 8 numbers in nondecreasing order
Input an integer: 2 4 6 8 10 12 14 16
Input desired value 17
desired = 17
low,mid,hi 0 3 7
low,mid,hi 4 5 7
low,mid,hi 6 6 7
low,mid,hi 7 7 7
low,mid,hi 8 7 7
17 not found
11  remus!111>
Binary Search

- What changes in the code if we use a nonincreasing array of numbers rather than a nondecreasing array?

```java
if (desired == mid) ....
else if (desired < a[mid]) hi = mid-1;
    else low = mid+1;
```

has to change to

```java
if (desired == mid) ...
else if (desired < a[mid]) low = mid+1;
    else hi = mid-1;
```
Binary Search

• Problem decomposition
  – Finding desired in \(a[\text{low}]-a[\text{hi}]\) is reduced to finding it in \(a[\text{low}]-a[\text{mid}-1]\) or \(a[\text{mid}+1]-a[\text{hi}]\)
  – Problem size is halved at each step

• Do constant work at each step (2 compares) and no more than \(\log_2 n\) steps for \(n\) values

• What if wanted to search for objects in an array
  – Need \texttt{equals()} and \texttt{compareTo()}
Binary Search II - Recursive

- Suggests we can solve this *recursively*, as in GCD example with Bert and Ernie

```java
private static int binSearch(int low, int hi, int[] a, int desired) {
    int mid = (hi+low)/2;
    if (hi < low) {return -1;}  
    if (desired == a[mid])  return mid;  
    else if (desired < a[mid])
        return (binSearch(low,mid-1,a, desired));
    else return (binSearch(mid+1,hi,a, desired));
}
```

`newbinsearchRec.java`
How it works?

data:  1  3  5  7  9  10  14  18  9
index:  0  1  2  3  4  5  6  7  desired

Initial interval is a[0] to a[7];
Ask Ernie to find 9 in a[0] to a[7]. Ernie checks a[3] == 9, a[3] < 9?
Since answer is yes, Ernie asks Bert to find 9 in a[4] to a[7].
Since answer is no, Bert asks Elmo to find 9 in a[4] to a[4].
    Elmo checks a[4] == 9 ? and finds it is! Elmo tells Bert the answer is index 4.
    Bert tells Ernie the answer is 4.
Ernie tells the questioner the answer is 4.

3 pairs of comparisons!
8 = 2^3