

Feedback Questions:

- How was the reading assignment?
- Write a Java expression that generates a random number in the range [0, 2).
- Declare a 2-dim array of integers called int2d that contains 3 rows, each of which contains 10 integers.
- Write a statement that prints the last integer in the 3rd row of int2d.
- The binary search method requires that the array we are searching be _____.

Algorithm Analysis:

If the array elements are unsorted, on average linear search will compare the key to half the array elements.

So for an array of size n , linear search does $n/2$ comparisons in the average case.

```
public int binarySearch(int[] array, int key)
{
    int low = 0, high = array.length - 1;

    while(low <= high) // Search elements in positions low to high
    {
        int mid = (low + high)/2;
        if(array[mid] == key)
            return mid;
        else if(array[mid] < key) // Search top half
            low = mid + 1;
        else // Search bottom half
            high = mid - 1;
    }
    return -1;
}
```

Multi-Dimensional Arrays

2-dimensional arrays - used to represent tables, with rows and columns
2 index values - 1st is row number, 2nd is column number

Declaration:

```
int[][] a = new int[3][4];
```

The rows are numbered from 0 to 2, and the columns are numbered 0 to 3.

$a[i][j]$ is the element in row i , column j

To initialize an array element:

```
a[2][1] = 15;
```

Note: All array entries were set to 0 when the array was created.

To initialize all the array entries, use loops:

```
for(int row = 0; row < a.length; row++)
    for(int col = 0; col < a[row].length; col++)
        a[row][col] = 1;
```

Search Algorithms

Searching Arrays - Linear Search & Binary Search

Searching - locating a particular value (the *key*) in an array

Linear Search

- works well for small arrays or unsorted arrays
- looks at each entry, starting with first, until either search key is found or end of array is reached

```
public int linearSearch(int[] array, int key)
{
    for(int i = 0; i < array.length; i++)
        if(array[i] == key) return i;
    return -1;
}
```

Binary Search

- requires a sorted array - algorithm completely fails on unsorted arrays
- eliminates half of the remaining array elements from consideration after each comparison
- Compares the middle of the remaining array elements to search key on each pass
 - If they are equal, return index
 - If not, and search key < middle array element, search first half of array.
 - Else search second half of array.

Example: (in class)

Search for key = 40 in array containing: 2, 5, 7, 10, 15, 31, 40, 55

Algorithm Analysis:

Example -

In an array of 64 elements, binary search eliminates half the array from consideration after each comparison). Repeatedly dividing by 2, we get:

32, 16, 8, 4, 2, 1

So 6 comparisons are needed to search an array of size $2^6 = 64$ elements.

Note: Linear search on average would do $64/2 = 32$ comparisons.

In general, binary search needs n comparisons to search an array of size 2^n .

Or an array's elements can be initialized when the array is declared:

```
int[][] a = { {1, 2}, {3, 4} };
```

The elements 1, 2 are in the first row, and elements 3, 4 are in the second row.

The compiler determines the number of rows by counting the number of sub-initializer lists in the main initializer list. The compiler determines the number of columns in each row by counting the number of values in the corresponding sub-initializer list.

Example:

```
int[][] a = { {1, 2}, {3, 4, 5} };
```

Exercise: Write a method called sumIt that takes a 2 dim array, and returns the sum of its elements.

Sorting

Sorting - putting a collection of items in some specified order

Everyday examples:

1. Card players sort their cards.
2. Libraries sort books by classification number.
3. I sort student records in alphabetical order by last name.

Selection Sort

To sort A's entries into ascending order:

Step 0: Find smallest entry in A[0..N-1] and swap it with A[0].

Step 1: Find smallest entry in A[1..N-1] and swap it with A[1].

...

Step i: Find smallest entry in A[i..N-1] and swap it with A[i].

Solution grows from low end of array - after ith step, A[0], A[1], ..., A[i] contain the correct values.

Example: (in class) 3 J 10 7 2

```
void Selection_Sort(float[] A, int N)
{
    float tmp;
    int minIndex, i, j;

    for(i = 0; i < N; i++)
    {
        minIndex = i;

        for(j = i+1; j < N; j++)
            if(A[j] < A[minIndex]) minIndex = j;

        // swap the contents of A[i] and A[minIndex]
        tmp = A[i];
        A[i] = A[minIndex];
        A[minIndex] = tmp;
    }
}
```

Insertion Sort

- Works the way many people sort their hands when playing cards.
- Pick up 1 item at a time and move it into place among the items already picked up and sorted.

Step 1: The subarray A[0..0] is already sorted.

If A[1] is not in its correct place in A[0..1], swap A[1] with A[0].
Now A[0..1] is sorted.

Step 2: The subarray A[0..1] is sorted.

If currentVal = A[2] is not in its correct place in A[0..2], swap it with A[1].
If currentVal is still not in its correct place in A[0..2], swap it with A[0].
Now A[0..2] is sorted.

...

Step i: Subarray A[0..i-1] is sorted.

If currentVal = A[i] is not in its correct place in A[0..i], swap currentVal successively with A[i-1], A[i-2], ... until currentVal is in its correct place in A[0..i].
Now A[0..i] is sorted.

Sorting Arrays

Problem: Given an array A[N], arrange the elements of A in ascending order.

2 basic operations:

1. swap - exchange the position of 2 entries

2. pairwise comparison - determine which of 2 entries is largest

Note: What we mean by "largest" depends on the application.

Today: Selection Sort and Insertion Sort

To make a sensible choice of sorting algorithm for your application, you need to consider both the number of swaps and the number of comparisons an algorithm does.

- Sometimes comparisons are costly - e.g. takes longer to compare 2 Strings than 2 ints
- Sometimes swaps are expensive - if we're sorting large objects, swapping takes time.

Algorithm:

For i = 0 to N-1

 Find min of A[i], A[i+1], ..., A[N-1]. Call min A[minIndex].

 Swap A[i] and A[minIndex].

Example: (in class) Array containing 6 5 8 1 15 3

Analysis of Selection Sort:

Note - order of data does not affect number of swaps or comparisons done.

of swaps:

of comparisons:

There are faster sorting algorithms, but selection sort doesn't do a lot of swaps. Handy if we're sorting large objects. Might be willing sometimes to sacrifice more comparisons for fewer swaps.

Card Example: 3 J 10 7 2

Insertion Sort

For $i = 0$ to $N-1$

For($j = i; j > 0$ && $A[j] < A[j-1]; j--$)
swap $A[j]$ and $A[j-1]$

Example: Array containing 10 9 5 6