Algorithms
A Look At Efficiency

1A

Counting Operations:
How much work is done?

Algorithms

- A computer program should be totally correct, but it should also
    - execute as quickly as possible (time-efficiency)
    - use memory wisely (storage-efficiency)
- How do we compare programs (or algorithms in general) with respect to execution time?
    - various computers run at different speeds due to different processors
    - compilers optimize code before execution
    - the same algorithm can be written differently depending on the programming paradigm

Analyzing Algorithms

- Worst Case
  - Case with maximum number of operations
- Best Case
  - Case with minimum number of operations
- Average Case
  - Average number of operations over all cases.

What are the operations?

Search an array for a value

```
public static int search(int[] x, int target)
{              1      2              3
     for (int i = 0; i < x.length; i++)
     {         4                5
         if (x[i] == target) return i;
   6 }
     return -1;
}
```

Counting Operations

Let the x.length = n.

How many times is operation 1 executed? **1**

How many times is operation 5 and 6 executed in total? **1**

Total so far: **2**

Worst Case

Worst Case: The target is not in the array.

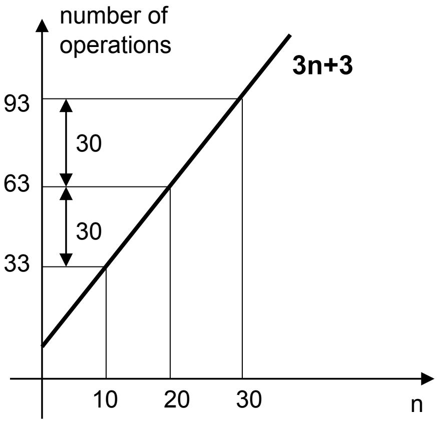
How many times is operation 2 executed? **n+1**

How many times is operation 3 executed? **n**

How many times is operation 4 executed? **n**

Total number of operations: **3n+3**

Worst Case



n (size of array)	number of operations
10	33
20	63
30	93

Counting Operations

Another Look

What if we counted just the comparison?

```
public static int search(int[] x, int target)
{
     for (int i = 0; i < x.length; i++)
     {
         if (x[i] == target) return i;
     }
     return -1;
}
```

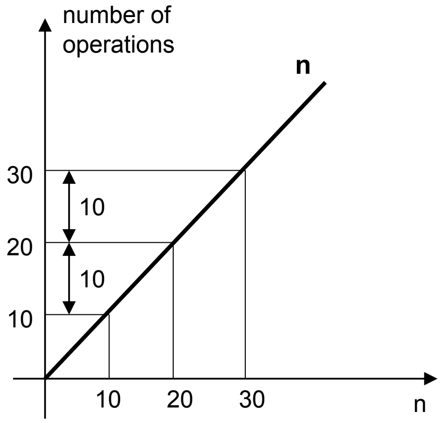
Worst Case

Worst Case: The target is not in the array.

How many times is the comparison executed?

n

Worst Case



n (size of array)	number of operations
10	10
20	20
30	30

Linear Algorithm

In both cases, the amount of work we do is linearly proportional to the number of data values in the array.

If we have n data values in the array, and we double the size of the array, how much work will we do searching the new array in the worst case?

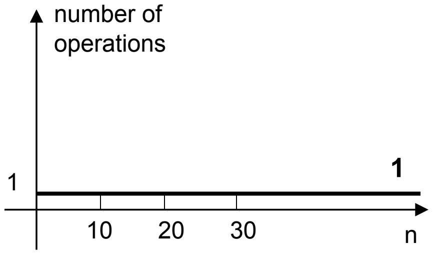
Counting Operations

In general, it doesn't matter what we count as operations, as long as we are consistent.

If we want to compare two algorithms that perform the same overall function, as long as we count the same type of operations in both, we can compare them for efficiency.

Best Case

How many comparisons are necessary in the best case for an array of n values?



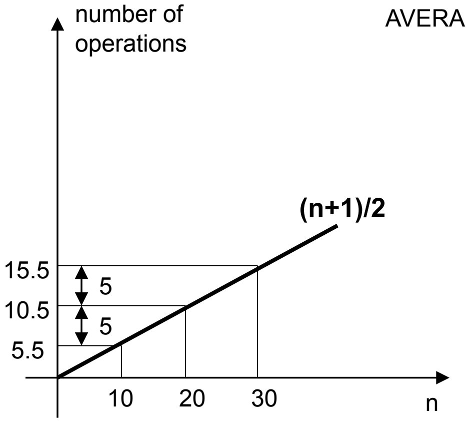
n (size of array)	number of operations
10	1
20	1
30	1

Average Case

How many comparisons are necessary in the average case for an array of n values (assuming the target is in the array)?

$$\frac{1+2+\ldots+(n-1)+n}{n}$$

Average Case



AVERAGE CASE IS LINEAR

n (size of array)	number of operations
10	5.5
20	10.5
30	15.5

Example 2

Do two array have no common values?

```
public static boolean diff(int[] x, int[] y)
{
     for (int i = 0; i < y.length; i++)
     {
         if (search(x, y[i]) != -1)
              return false;
     }
     return true;
}
```

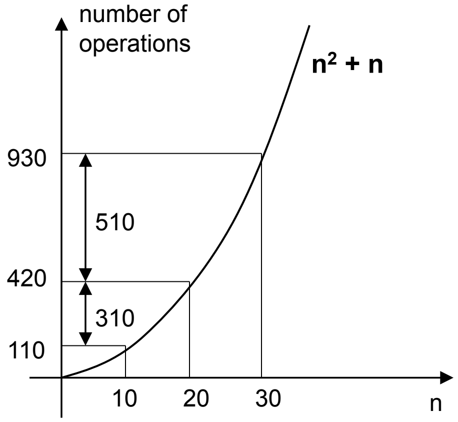
Worst Case Analysis

- Let m = the length of array x.
- Let n = the length of array y.
- The loop in `diff` repeats n times.
- Each call to search requires m comparisons.
- The total number of comparisons in the worst case is:

Worst Case Analysis

- Assume m = n. (The arrays are the same size.)
- Then the total number of comparisons in the worst case is:

Analysis



n (size of array)	number of operations
10	110
20	420
30	930

Example 3

Is each item in an array unique?

```
public static boolean unique(int[] x)
{
      for (int i = 0; i < x.length; i++) {
           for (int j = 0; j < x.length; j++) {
                 if (i != j && x[i] == x[j])
                       return false;
           }
      }
      return true;
}
```

Worst Case Analysis

Example 4

Is each item in an array unique? (2nd try)

```
public static boolean unique(int[] x)
{
      for (int i = 0; i < x.length; i++) {
           for (int j = i+1; j < x.length; j++) {
                 if (x[i] == x[j])
                       return false;
           }
      }
      return true;
}
```

Worst Case Analysis