Measuring Running Times

Today's Class

- Bubble Sort Experiments
- O Notation

Good Programs Are:

- Correct / Complete
- Robust
- Maintainable
- Efficient

Measuring Efficiency

- Empirical/Experimental - use clock to get actual running times for different inputs

- Sample Experiment

BubbleSort Experiment

touching lives

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

void BubbleSort(int arr[], int size) {
   int i,j,tmp;
   for(j=1; j<size; j++) {
     for (i = 0; i < size - j; i++) {
        if (arr[i] > arr[i + 1]) {
           tmp = arr[i];
           arr[i] = arr[i + 1];
           arr[i + 1] = tmp;
        }
     }
   }
}
```

```
void main(int argc, char **argv) {
    int size = atoi(argv[1]);
    int a[size];
    for(int i=0; i<size; i++) {
        a[i]=rand();
        /* std::cout << a[i] << endl; */
    }
    cout << "starting the sort" << endl;
    clock_t start = clock();

    BubbleSort(a,size);

    clock_t end = clock();
    double time = (double) (end-start) /
CLOCKS_PER_SEC * 1000.0;
    cout << "finished " << time << "
milliseconds" << endl;
}
```

THE UNIVERSITY OF ALABAMA
DEPARTMENT OF COMPUTER SCIENCE

BubbleSort Experiment Continued

Now the next Bubble Sort Trials:

First run with sizes 1000, 2000, 4000, 8000, and 16000

Running time for 32,000 ints ?

Running time for 64,000 ints ?

Running time for 128,000 ints ?

Running time for 1 million ints (~ 8 * 128,000) ?

Running time for 8 million ints ?

Back to Measuring Efficiency

We generally use worst case performance

- Sometimes we need a guarantee (Time critical systems)
- What input do we use otherwise?
- Average case:
  - Average over what?
  - Are all inputs equally likely?
  - Average case analysis can be extremely difficult, mathematics involving probability distributions
- Best case?
  - How about we just assume that we will be lucky
  - Not realistic in practice

Problems with Measuring Efficiency

Problems:

- Different machines will have different performance
  - Phone or desktop or supercomputer
- Some machines better at some tasks
- What inputs to use
- Nearly impossible to compare

Solution: Abstraction

- Big-O Notation
  - Represent the running time as a function of the size of the input (usually denoted by N).
  - Simplify to just the “high order” term.
  - Examples:
    - O(N), O(N^2), O(N log(N)), O(1), …

Big-O Notation Formal Definition

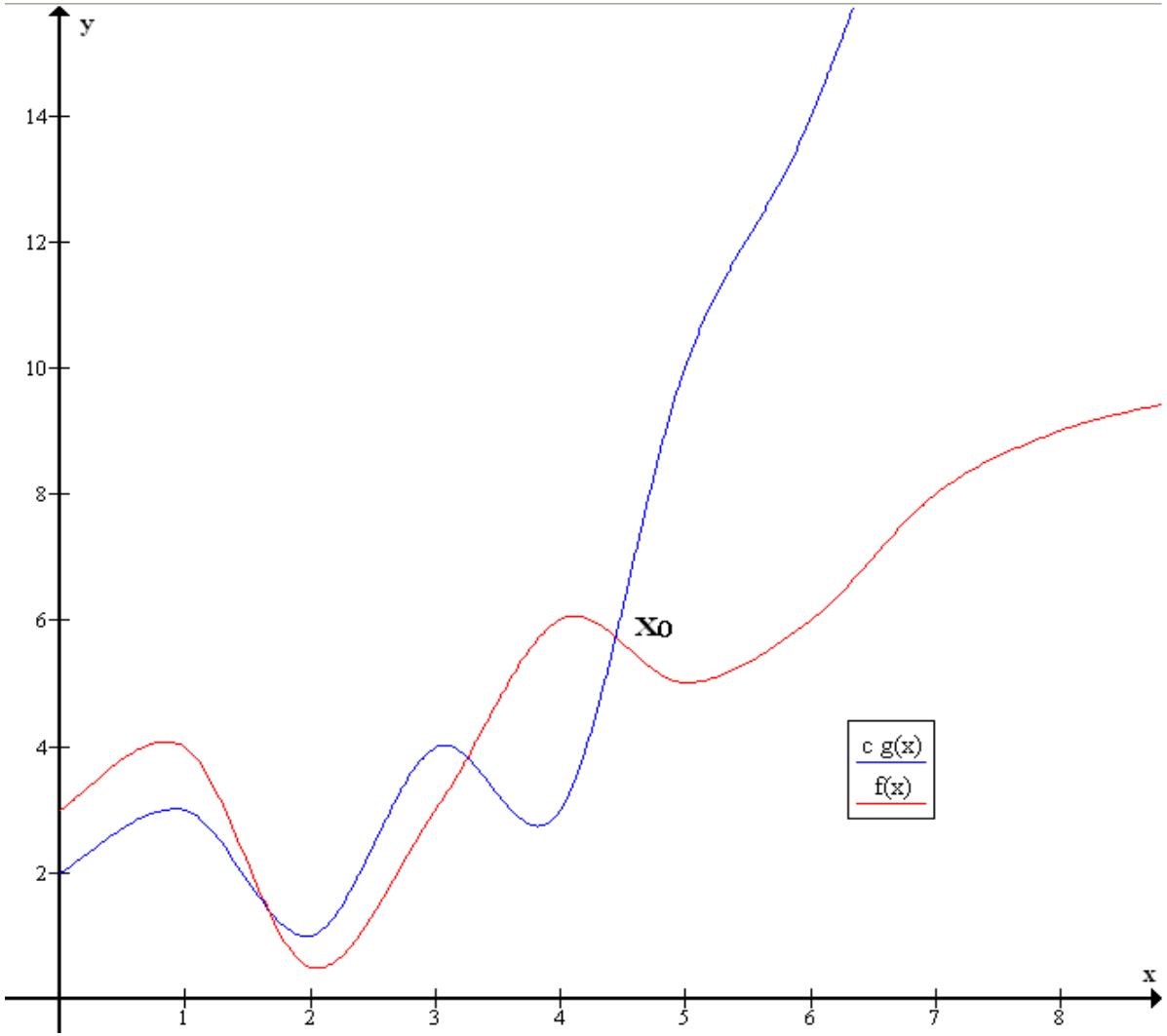
$$f(n) = O\big(g(n)\big)$$

iff $\exists$ constants c and n_0

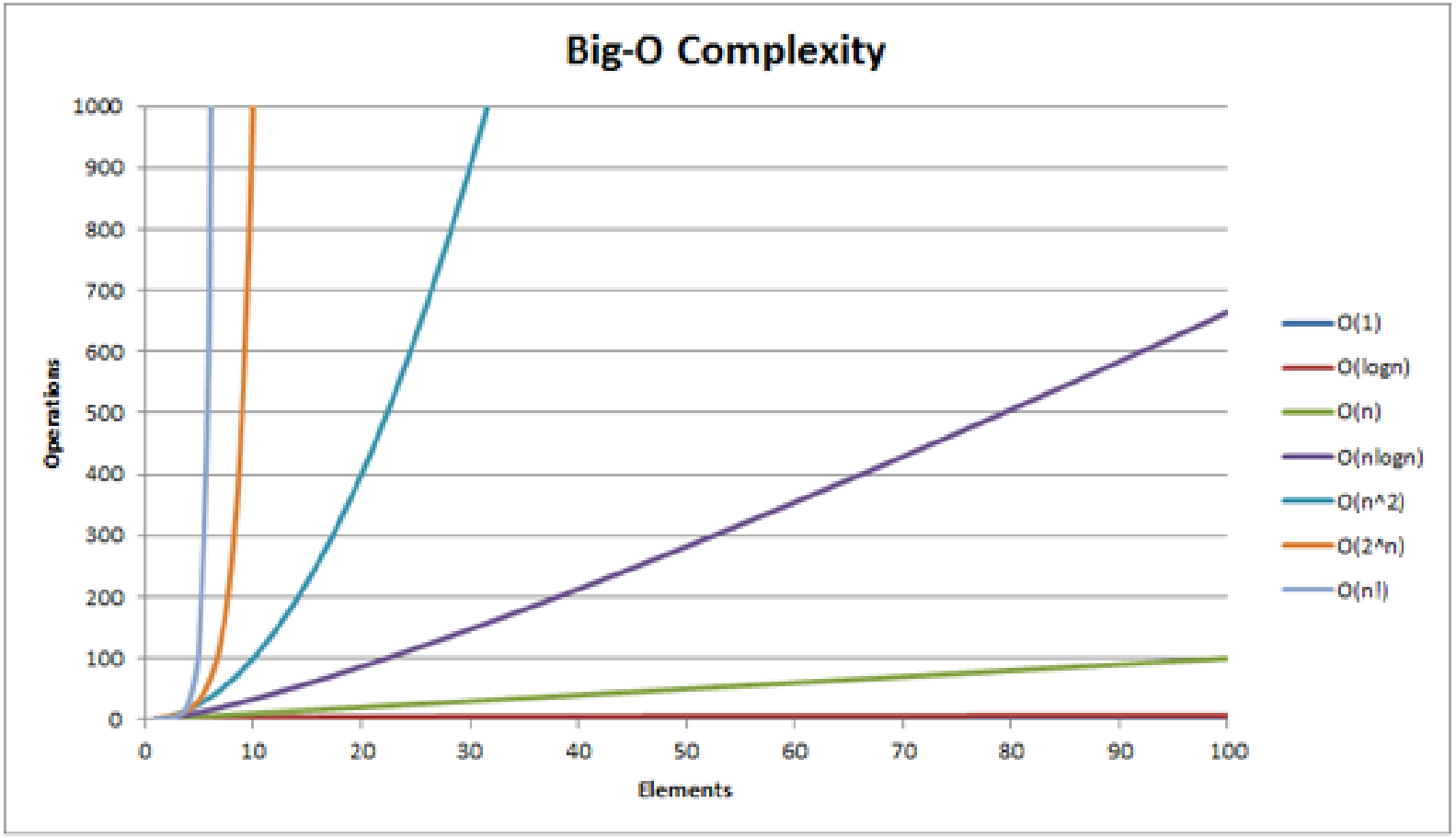
such that $\forall n > n_0$

$$f(n) \leq c\ g(n)$$

Big-O Notation Example



O Notation Comparison



A Practical Example


```
int max(int items[], int size) {
    int largest = items[0];
    for(int i=0; i<size; i++)
        if (items[i] > largest) largest = items[i];
    return largest;
}
```

- What is the running time of the max function if items[] has N values?
- About 2N + 2 steps ?
- O(N) time

Your Turn


```
bool linear_search(int items[], int size, int value) {
    for(int  i=0; i,<size; i++)
        if (items[i] == value) return true;
    return false;
}
```

What is the running time of the linear_search function if items[] has N values?

Your Turn Part 2


```
bool binary_search(int items[], int size, int value) {
    int low=0, high=size-1;
    while (low<=high) {
        mid=(low+high)/2;
        if (items[mid] == value) return true;
        else if (items[mid]<value) low=mid+1;
        else /* if (items[mid]>value) */ high=mid-1;
    }
    return false;
}
```

What is the running time of the binary_search function if items[] has N values?

Your Turn Part 3


```
int array_search(int items1[], int items2[], int size1, int size2) {
    for(int i=0; i<size1; i++)
        for(int j=0; j<size2; j++)
            if (items2[j] == items1[i]) return true
    return false;
}
```

What is the running time of the array_search function if items1[] and items2[] each have N values?