Week-2

Asymptotic Notation

Analyzing Algorithms

- Predict the amount of resources required:
  - memory: how much space is needed?
  - computational time: how fast the algorithm runs?
- FACT: running time grows with the size of the input
- Input size (number of elements in the input)
  - Size of an array, polynomial degree, # of elements in a matrix, # of bits in the binary representation of the input, vertices and edges in a graph

Def: Running time = the number of primitive operations (steps) executed before termination

- Arithmetic operations (+, -, *), data movement, control, decision making (*if, while*), comparison

Algorithm Analysis: Example

- *Alg.:* MIN (a[1], …, a[n])

```
m ← a[1];
for i ← 2 to n
      if a[i] < m
    then m ← a[i];
```

- **Running time**:
  - the number of primitive operations (steps) executed before termination

$T(n) = 1$ [first step] + (n) [for loop] + $(n-1)$ [if condition] + $(n-1)$ [the assignment in then] = $3n - 1$

- Order (rate) of growth:
  - The leading term of the formula
  - Expresses the asymptotic behavior of the algorithm

Typical Running Time Functions

- 1 (constant running time):
  - Instructions are executed once or a few times
- logN (logarithmic)
  - A big problem is solved by cutting the original problem in smaller sizes, by a constant fraction at each step
- N (linear)
  - A small amount of processing is done on each input element
- N logN
  - A problem is solved by dividing it into smaller problems, solving them independently and combining the solution

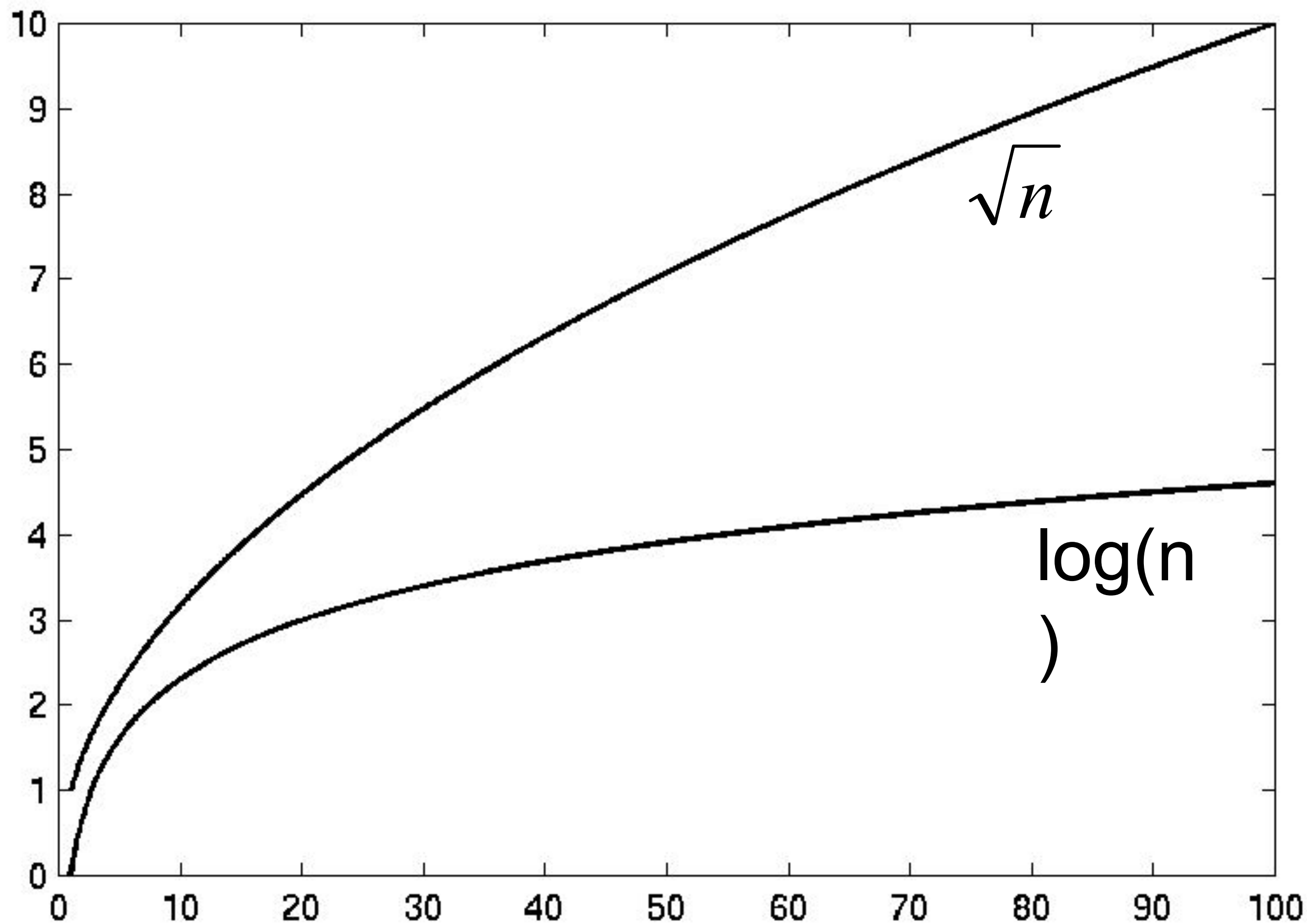
Typical Running Time Functions

- N^2 (quadratic)
  - Typical for algorithms that process all pairs of data items (double nested loops)
- N^3 (cubic)
  - Processing of triples of data (triple nested loops)
- N^K (polynomial)
- 2^N (exponential)
  - Few exponential algorithms are appropriate for practical use

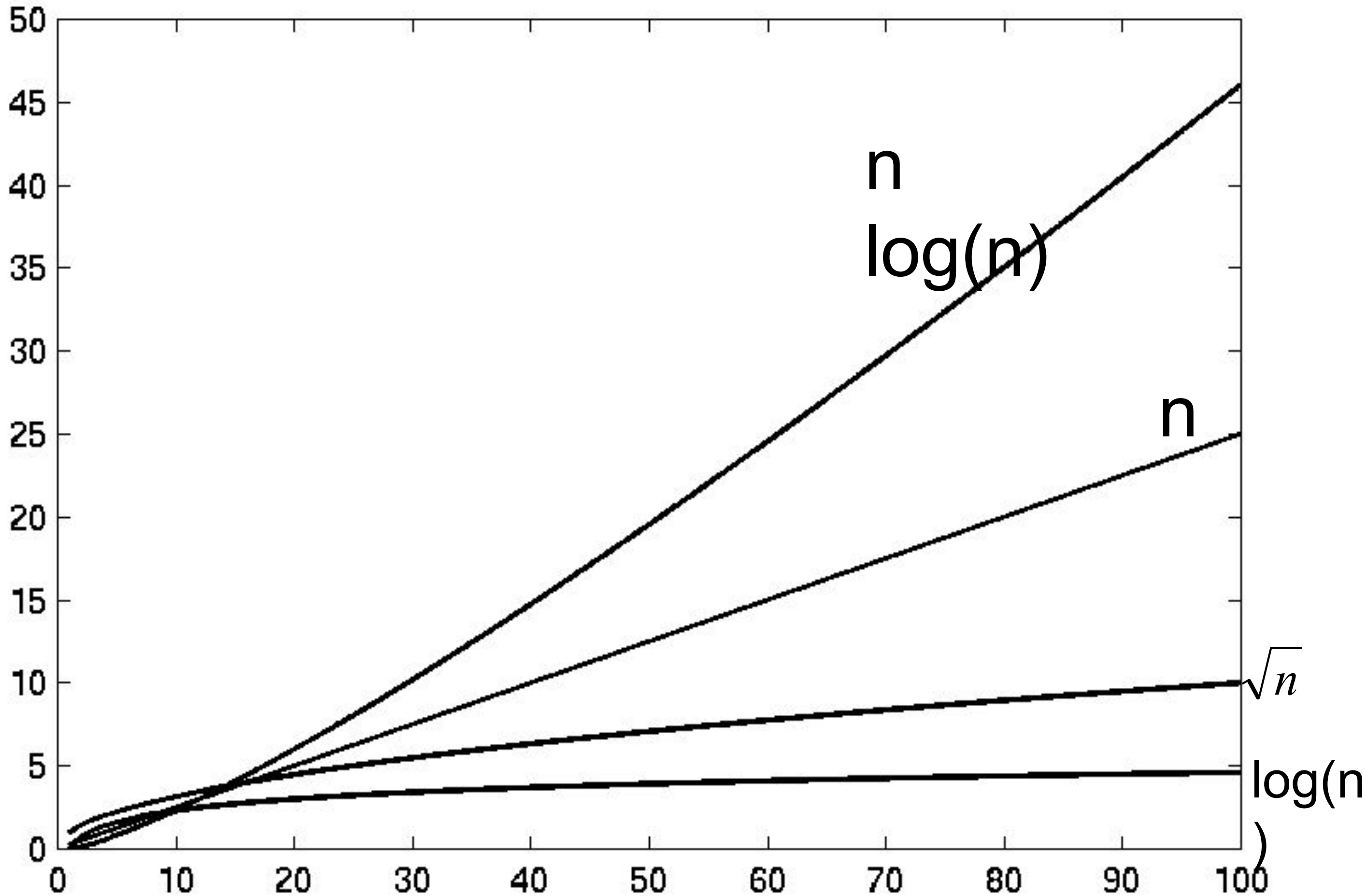
Growth of Functions

n	1	lgn	n	nlgn	n^2	n^3	2^n
1	1	0.00	1	0	1	1	2
10	1	3.32	10	33	100	1,000	1024
100	1	6.64	100	664	10,000	1,000,000	1.2×10^{30}
1000	1	9.97	1000	9970	1,000,000	10^9	1.1×10^{301}

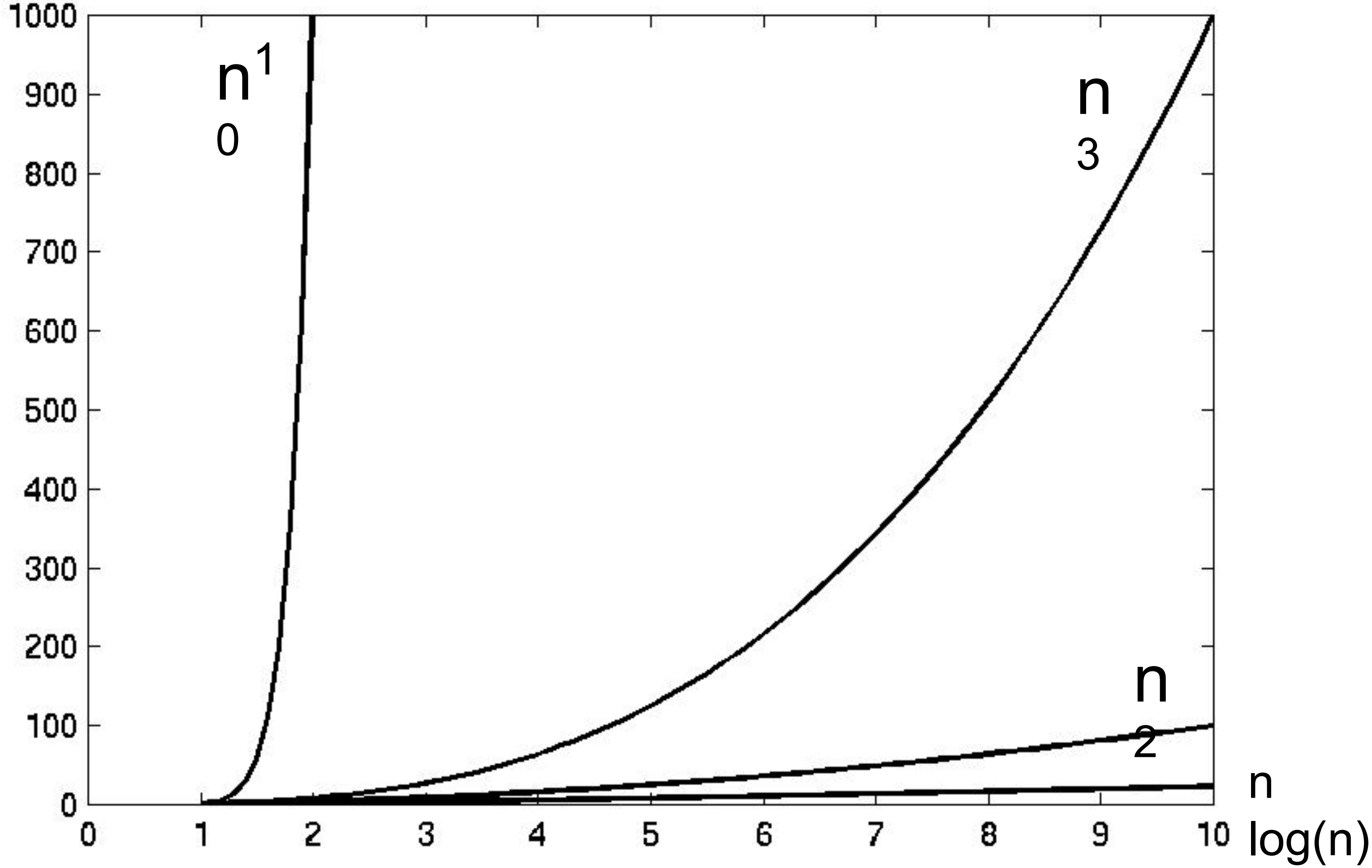
Complexity Graphs



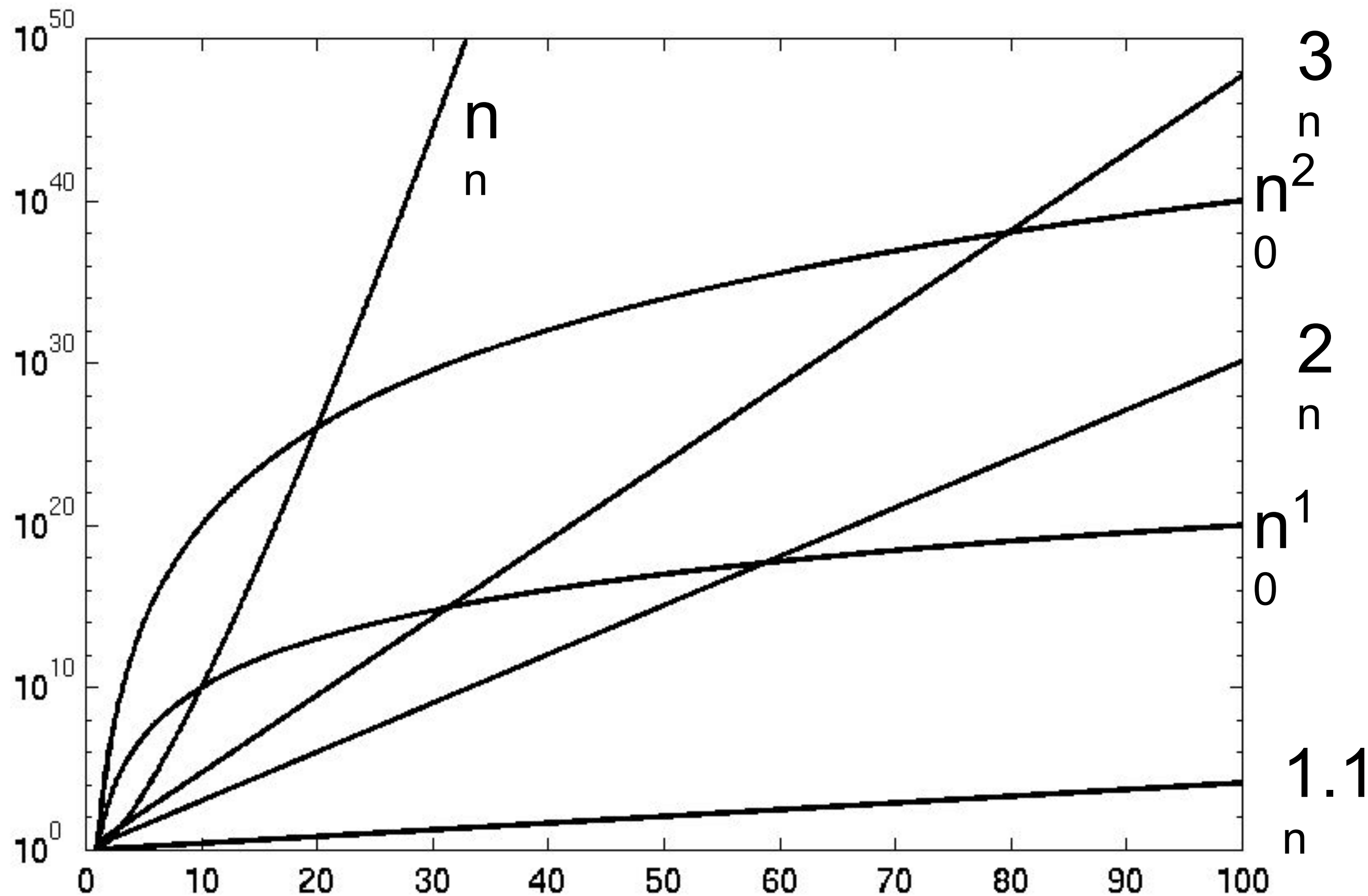
Complexity Graphs



Complexity Graphs



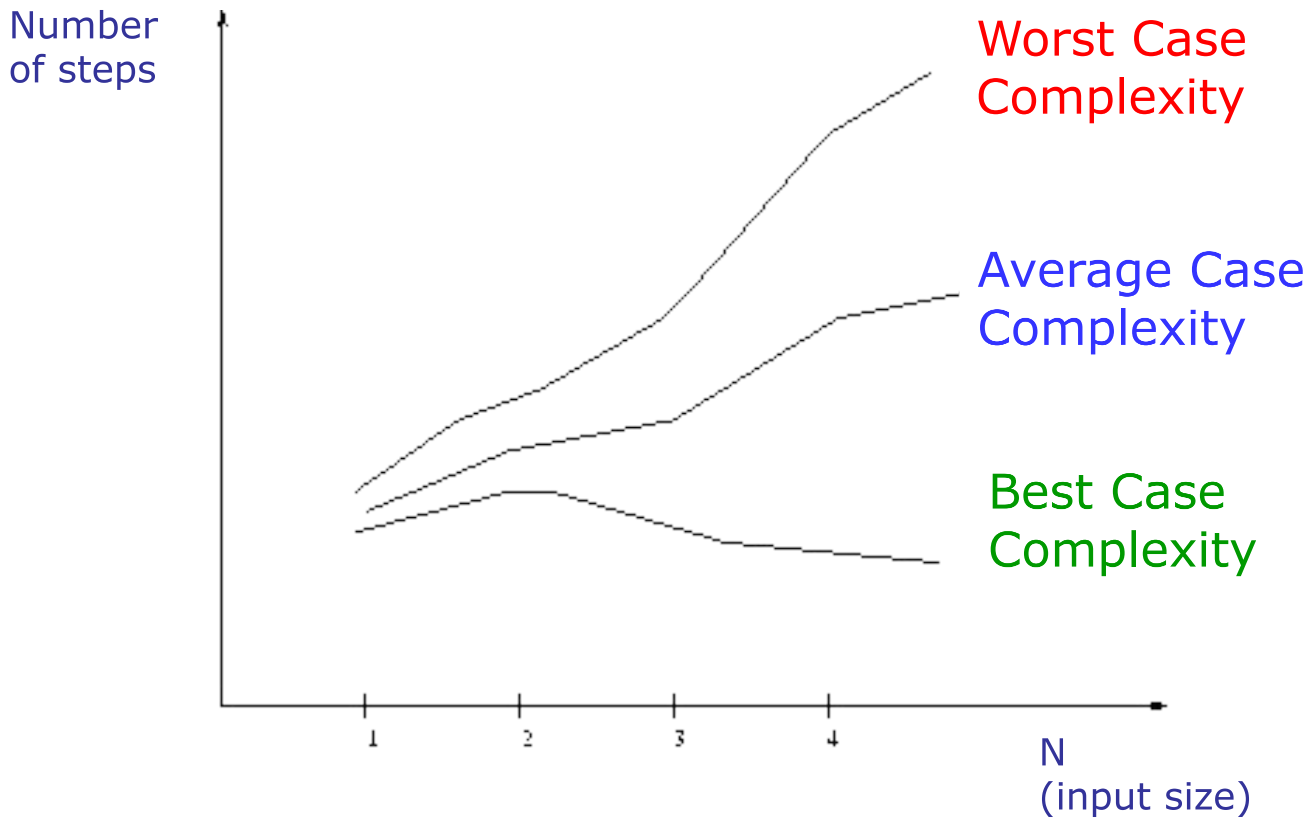
Complexity Graphs (log scale)



Algorithm Complexity

- Worst Case Complexity:
  - the function defined by the *maximum* number of steps taken on any instance of size *n*
- Best Case Complexity:
  - the function defined by the *minimum* number of steps taken on any instance of size *n*
- Average Case Complexity:
  - the function defined by the *average* number of steps taken on any instance of size *n*

Best, Worst, and Average Case Complexity



Example

- Code:
- `a = b;`

- Complexity:

Example

- Code:
- `a = b;`

- Complexity:

Example

- Code:
- `sum = 0;`
- `for (i=1; i <=n; i++)`
- `    sum += n;`

- Complexity:

Example

- Code:

```
sum = 0;
for (j=1; j<=n; j++)
    for (i=1; i<=j; i++)
        sum++;
for (k=0; k<n; k++)
    A[k] = k;
```

- Complexity:

Example

- Code:
- `sum1 = 0;`
- `for (i=1; i<=n; i++)`
- `   for (j=1; j<=n; j++)`
- `      sum1++;`
- Complexity:

Example

- Code:
- `sum2 = 0;`
- `for (i=1; i<=n; i++)`
- `   for (j=1; j<=i; j++)`
- `      sum2++;`
- Complexity:

Example

- Code:
- `sum1 = 0;`
- `for (k=1; k<=n; k*=2)`
- `   for (j=1; j<=n; j++)`
- `      sum1++;`
- Complexity:

Thank you