成均館大學校
SUNG KYUN KWAN UNIVERSITY

Problem Solving

Complexity Analysis

Instructor: Jae-Pil Heo (허재필)

Background

- Suppose that we have two solutions, how can we tell which is better?

- We could implement both solutions, run them both
  - Expensive and error prone

- Preferably, we should analyze them mathematically

Asymptotic Analysis

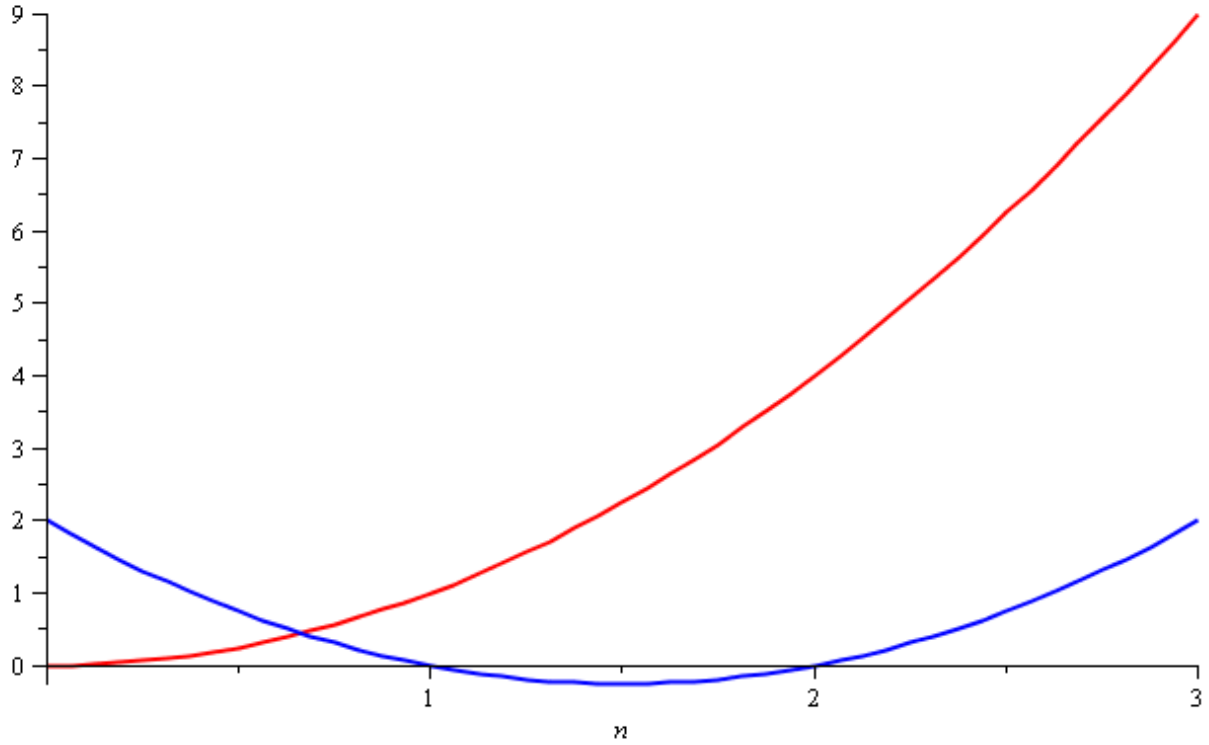
- In general, we will always analyze solutions with respect to one or more *variables*

- Examples with one variable:
  - The number of items n currently stored in an array or other data structure
  - The number of items expected to be stored in an array or other data structure
  - The dimensions of an $n \times n$ matrix

- Examples with multiple variables:
  - Dealing with n objects stored in m memory locations
  - Multiplying a $k \times m$ and an $m \times n$ matrix
  - Dealing with sparse matrices of size $n \times n$ with m non-zero entries

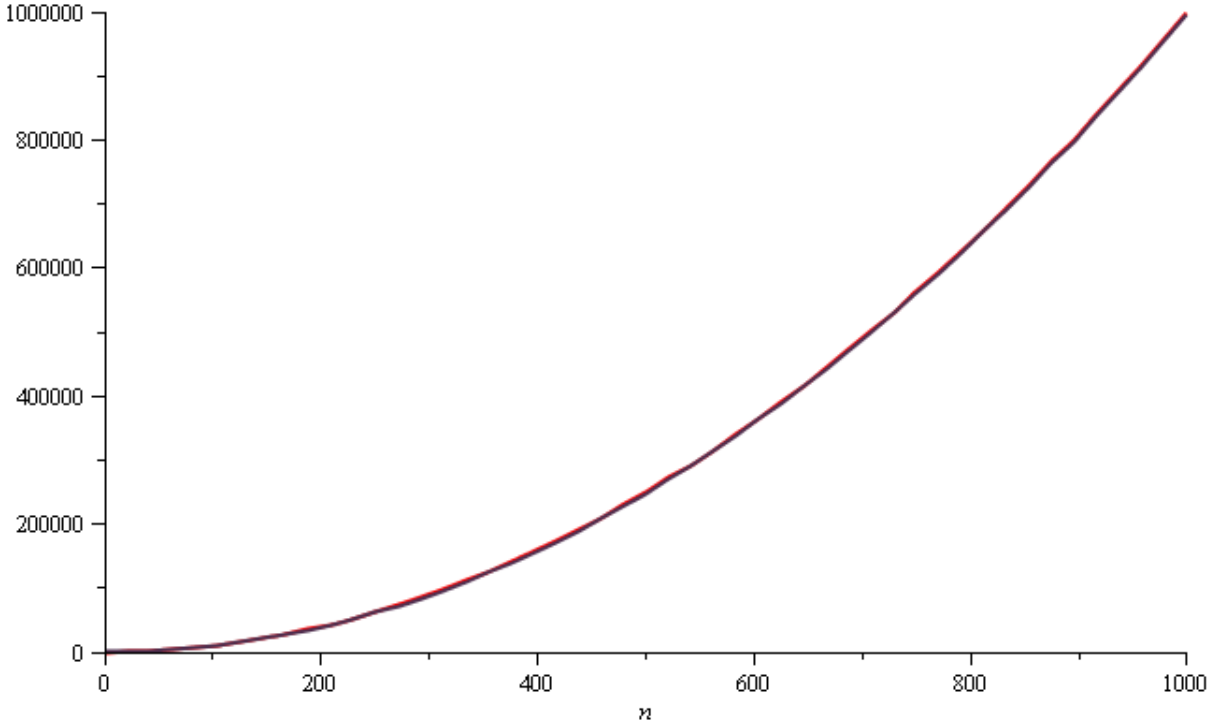
Quadratic Growth

- Consider the two functions
  - $f(n) = n^2$
  - $g(n) = n^2 - 3n + 2$
- Around $n = 0$, they look very different



Quadratic Growth

- Consider the two functions
  - $f(n) = n^2$
  - $g(n) = n^2 - 3n + 2$
  - Yet on the range $n = [0, 1000]$, they are (relatively) indistinguishable:



Quadratic Growth

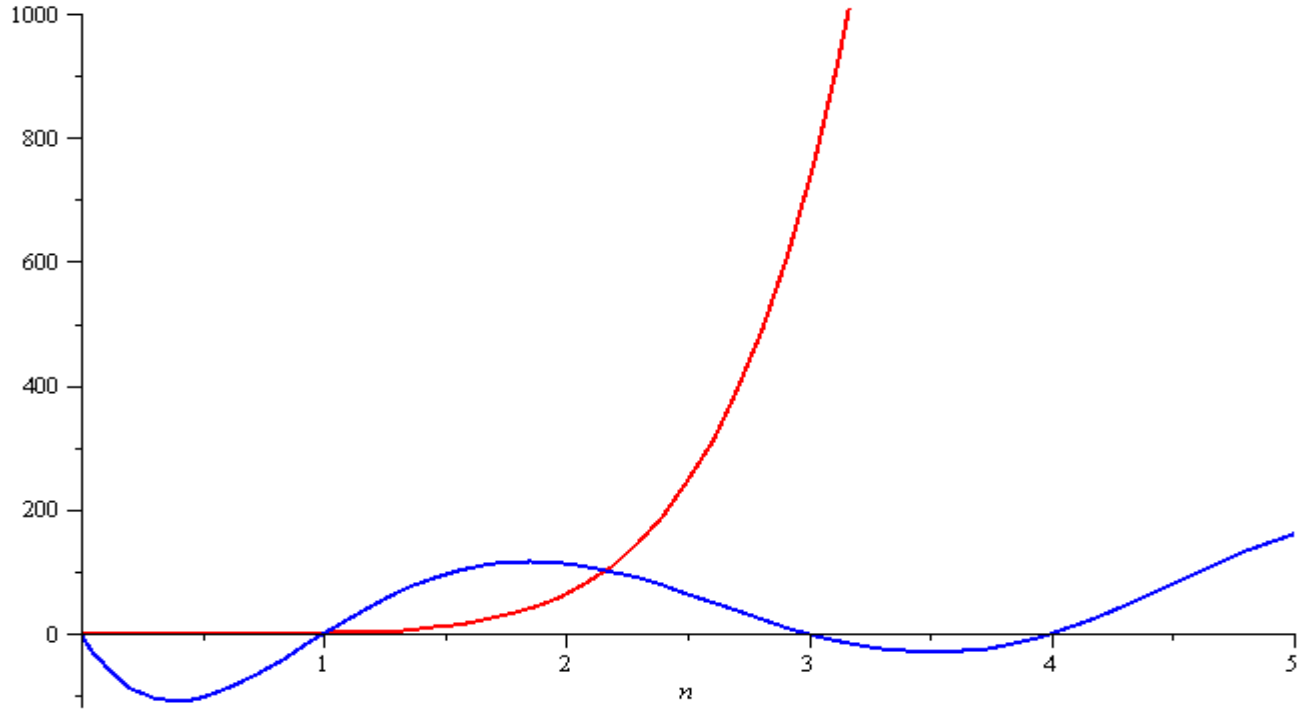
- The absolute difference is large, for example,
  - $f(1000) = 1{,}000{,}000$
  - $g(1000) = \quad 997{,}002$

- But the relative difference is very small
  - $\left|\frac{f(1000)-g(1000)}{f(1000)}\right| = 0.002998 < 0.3\%$

- This difference goes to zero as $n \to \infty$
  - $\lim_{n\to\infty} \left|\frac{f(n)-g(n)}{f(n)}\right| = 0$

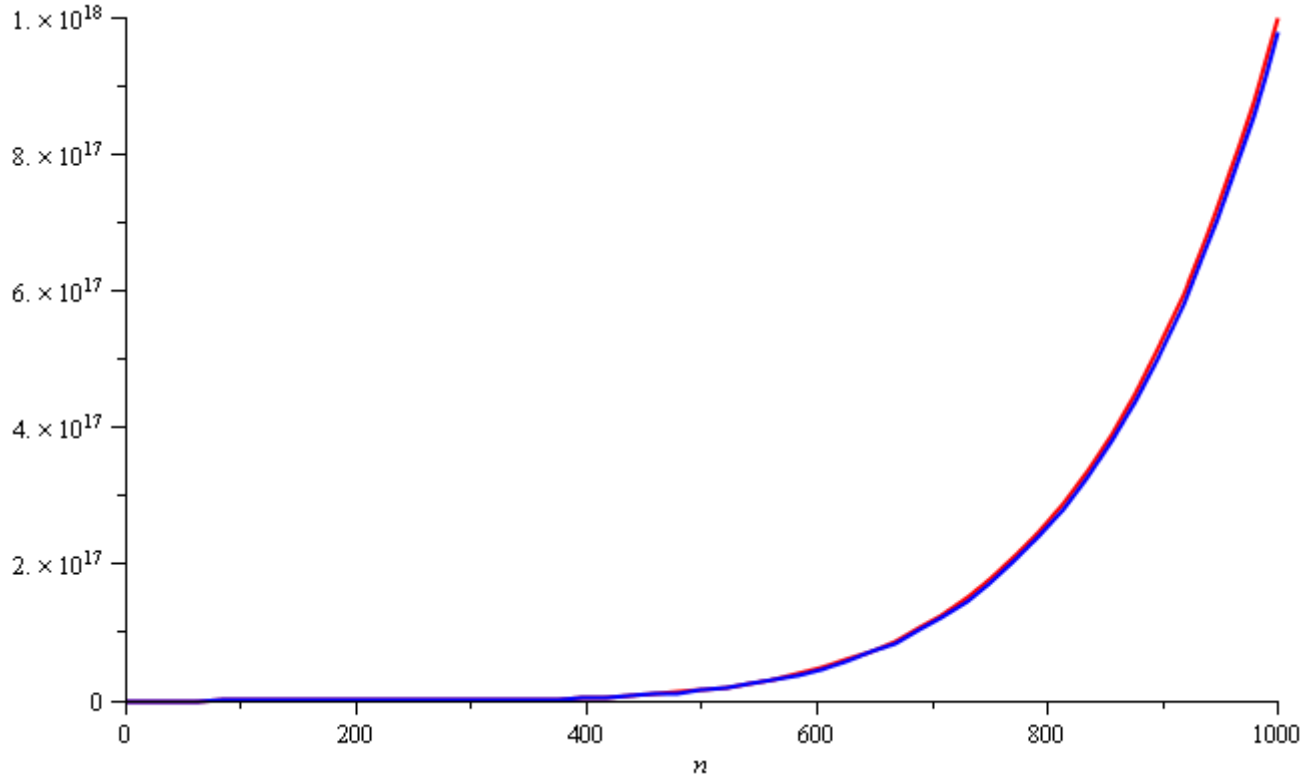
Polynomial Growth

- To demonstrate with another example,
  - $f(n) = n^6$
  - $g(n) = n^6 - 23n^5 + 193n^4 - 729n^3 + 1206n^2 - 648n$
  - Around $n = 0$, they are very different



Polynomial Growth

- To demonstrate with another example,
  - $f(n) = n^6$
  - $g(n) = n^6 - 23n^5 + 193n^4 - 729n^3 + 1206n^2 - 648n$
  - Still, around $n = 1000$, the relative difference is less than 3%



Polynomial Growth

- The justification for both pairs of polynomials being similar is that, in both cases, they each had the same leading term:
  - n^2 in the first case, n^6 in the second

- Suppose however, that the coefficients of the leading terms were different
  - In this case, both functions would exhibit the same rate of growth, however, one would always be proportionally larger

Counting Instructions

- Suppose that we had two solutions which sorted a list of size n and the runtime (in the number of instructions) is given by:
  - $b_{worst}(n) = 4.7n^2 - 0.5n + 5$
  - $b_{best}(n) = 3.8n^2 + 0.5n + 5$
  - $s(n) = 4n^2 + 14n + 12$

- The smaller the value n, the fewer instructions are run
  - For $n \leq 21$, $b_{worst}(n) < s(n)$
  - For $n \geq 22$, $b_{worst}(n) > s(n)$

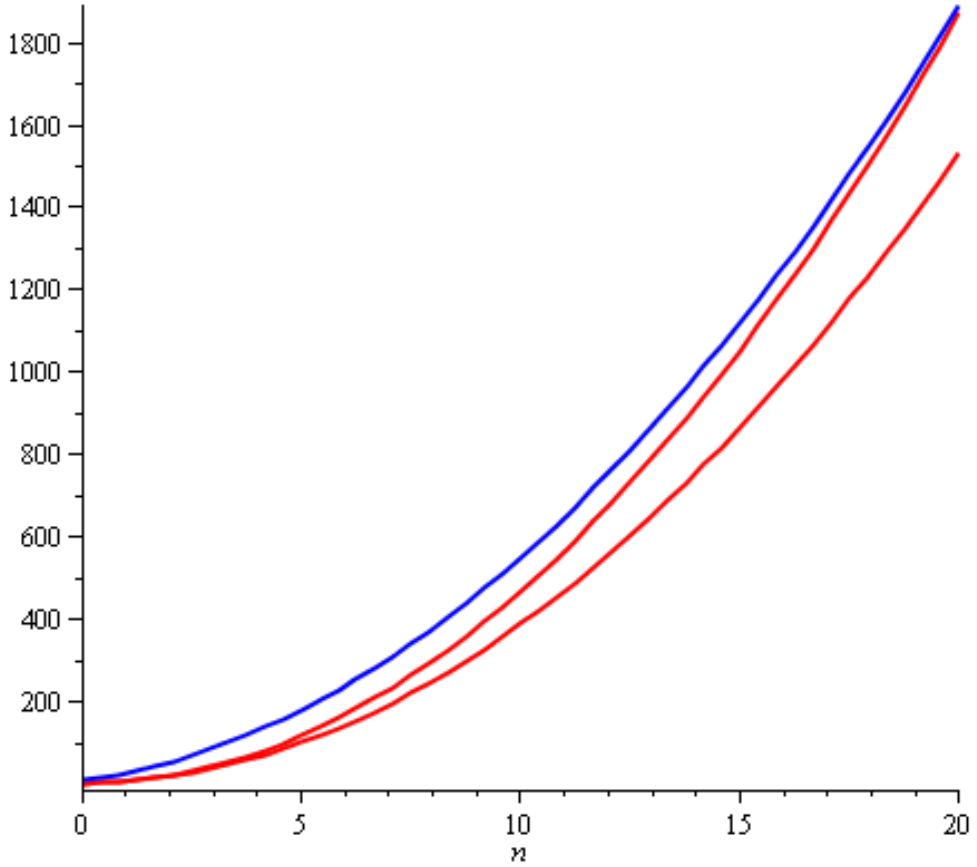
Counting Instructions

- Is this a serious difference between these two solutions?

- Because we can count the number instructions, we can also estimate how much time is required to run one of these algorithms on a computer.

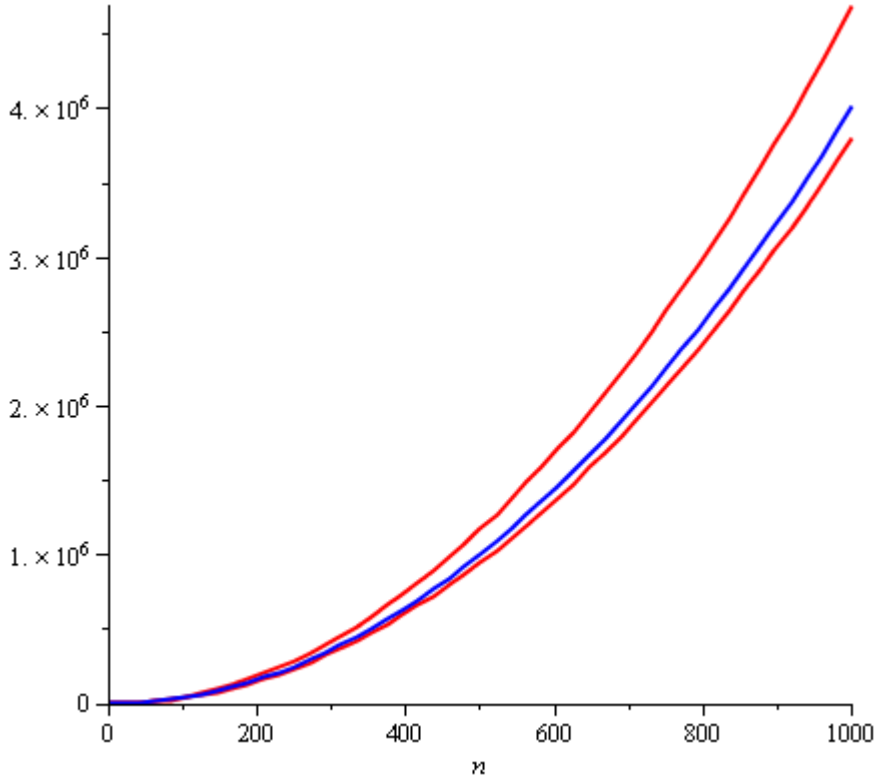
Counting Instructions

- With small values of n, the algorithm described by $s(n)$ requires more instructions than even the $b_{worst}(n)$



Counting Instructions

- Near $n = 1000$, $b_{worst}(n) \approx 1.175\ s(n)$ and $b_{best}(n) \approx 0.95\ s(n)$



Counting Instructions

- If $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots$ and $g(n) = b_k n^k + b_{k-1} n^{k-1} + \cdots$
  - For large enough n, it will always be true that

$$f(n) < Mg(n)$$

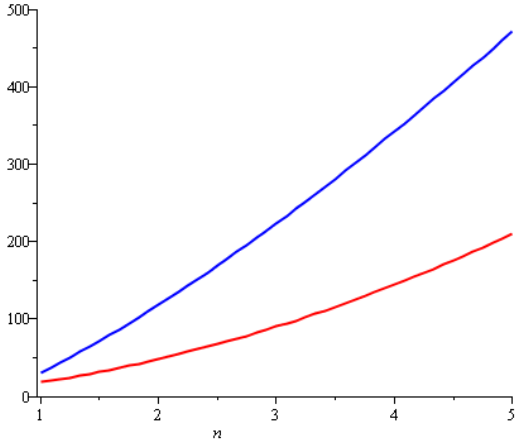
    where, we choose

$$M = \frac{a_k}{b_k} + 1$$

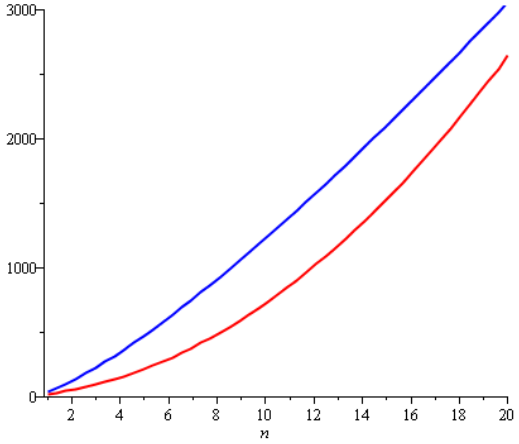
  - In this case, we only need a computer which is M times faster.

Counting Instructions

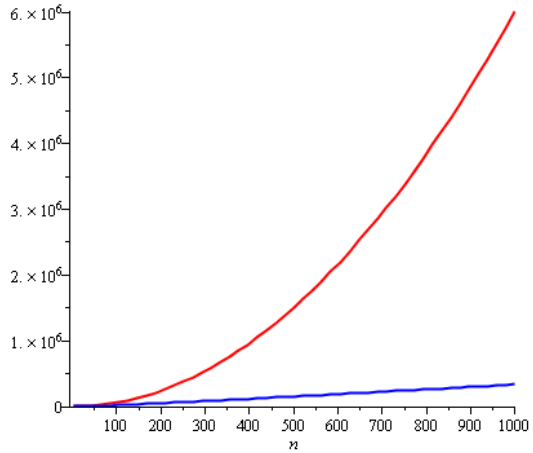
- Another example:
  - $f(n) = an^2$ and $g(n) = b\,n \lg n$



$n = [0, 5]$



$n = [0, 20]$



$n = [0, 1000]$

- Question: can we just buy a faster computer?

Weak Ordering

- Consider the following definitions:
  - We will consider two functions to be equivalent, $f \sim g$, if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c \qquad \text{where, } 0 < c < \infty$$

  - We will state that $f < g$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

- These define a weak ordering

Weak Ordering

- Let $f(n)$ and $g(n)$ describe either the run-time of two algorithms
  - If $f(n) \sim g(n)$, then it is always possible to improve the performance of one function over the other by purchasing a faster computer
  - If $f(n) < g(n)$, then you can never purchase a computer fast enough so that the second function always runs in less time than the first

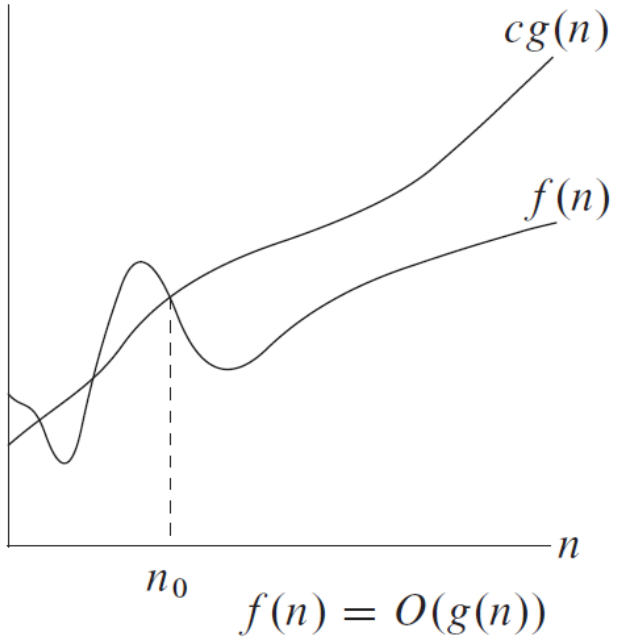
O – Notation

- O – Notation (big-oh)
  - A function $f(n) = O(\ g(n)\ )$ if there exist positive constants c and n_0 such that

$$0 \leq f(n) \leq cg(n)$$

    for all $n \geq n_0$

  - $g(n)$ is an *asymptotically upper bound* for $f(n)$
  - O is suitable to bound the worse-case running time.



O – Notation

- O – Notation (big-oh)
  - Examples:
    - $2n^2 + 8n - 2 = O(n^2)$
    - $2n^2 + 8n - 2 = O(n^3)$
    - $2n^2 + 8n - 2 = O(n^4)$

Bubble Sort

- Starting with the first item, assume that it is the largest

- Compare it with the second item:
  - If the first is larger, swap the two,
  - Otherwise, assume that the second item is the largest

- Continue up the array, either swapping or redefining the largest item

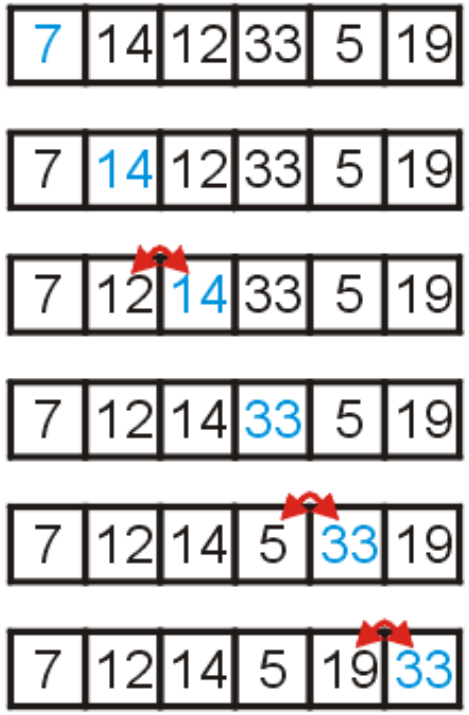
Bubble Sort

- After one pass, the largest item must be the last in the list
- Start at the front again:
  - the second pass will bring the second largest element into the second last position
- Repeat $n - 1$ times, after which, all entries will be in place

Example

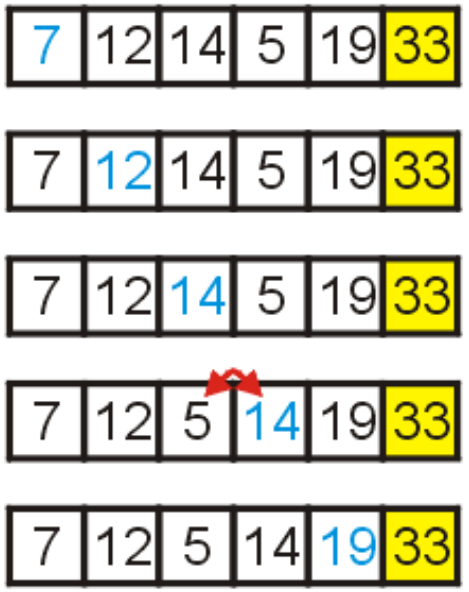
- Consider the unsorted array to the right

- We start with the element in the first location, and move forward:
  - if the current and next items are in order, continue with the next item, otherwise
  - swap the two entries



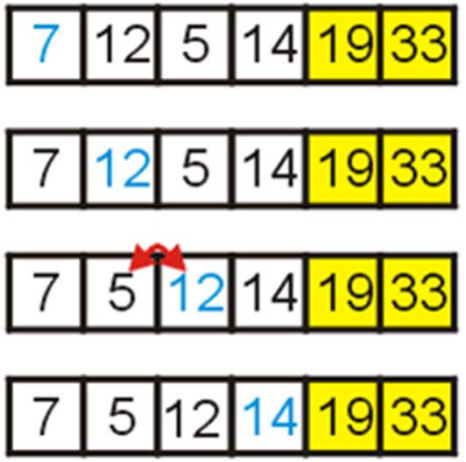
Example

- After one loop, the largest element is in the last location
  - Repeat the procedure



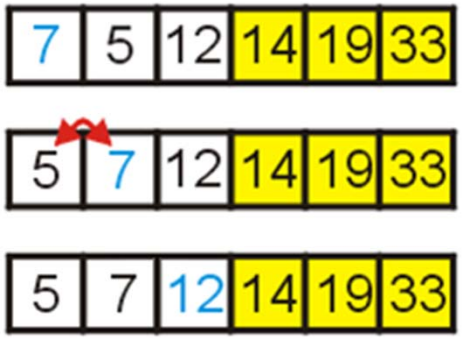
Example

- Now the two largest elements are at the end
  - Repeat again

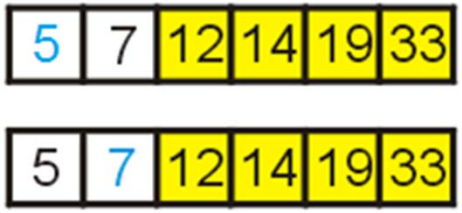


Example

- With this loop, 5 and 7 are swapped



- Finally, we swap the last two entries to order them
  - At this point, we have a sorted array



Implementation

- The default algorithm (C++ version):

```
template<typename Type>
void Bubble_Sort(Type *_array, int _n)
{
  for(int i=_n-1 ; i>0 ; i--)
  {
    for(int j=0 ; j<i ; j++)
    {
      if( _array[j] > _array[j+1] )
      {
        std::swap( _array[j] , _array[j+1] );
      }
    }
  }
}
```

Complexity of Bubble Sort

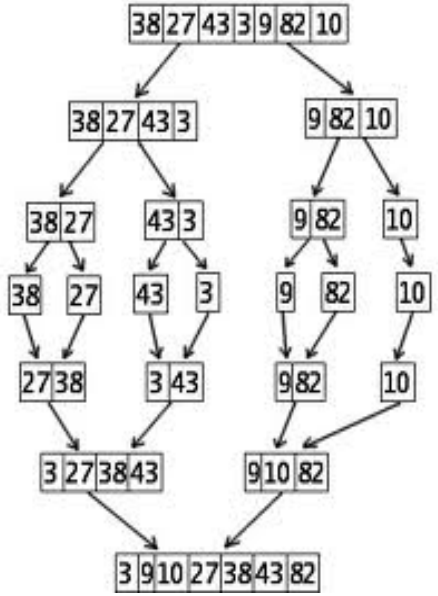
- Here we have two nested loops, and therefore calculating the run time is straight-forward:

$$\sum_{k=1}^{n-1}(n-k) = n(n-1) - \frac{n(n-1)}{2} = \frac{n(n-1)}{2} = O(n^2)$$

  - Worst-case: $O(n^2)$
  - Average-case: $O(n^2)$

More Sorting Methods and Their Complexities

- Will be intensively covered in Algorithm class ☺
- FYI,
  - Selection sort
    - divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. (Wikipedia)
    - Worst-case: $O(n^2)$ / Average-case: $O(n^2)$
  - Merge sort
    - Divides the unsorted list into n sublists, each containing 1 element, and repeatedly merges sublists to produce new sorted sublists until there is only 1 sublist remaining. (Wikipedia)
    - Worst-case: $O(n \log n)$ / Average-case:$O(n \log n)$



Example: Complexity of Matrix Multiplication

Matrix Multiplication

- Matrices A and B are multiplied; **C = AB**
- What is its (computational) complexity?
- Any efficient matrix multiplication method?

Complexity? O(n³)

$$A=\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \quad B=\begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}$$

$$C=\begin{bmatrix} \sum_{k=1}^{n} a_{1k}b_{k1} & 6 & \sum_{k=1}^{n} a_{1k}b_{kn} \\ 7 & 9 & 7 \\ \sum_{k=1}^{n} a_{nk}b_{k1} & 6 & \sum_{k=1}^{n} a_{nk}b_{kn} \end{bmatrix}$$

$$C=\begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix}=\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}\times\begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}$$

Complexity? O(n³)

$$C_1=A_1B_1+A_2B_3;\quad C_2=A_1B_2+A_2B_4$$
$$C_3=A_3B_1+A_4B_3;\quad C_4=A_3B_2+A_4B_4$$

$$P_1=A_1(B_2-B_4);P_2=(A_1+A_2)B_4;$$
$$P_3=(A_3+A_4)B_1;P_4=A_4(-B_1+B_3);$$
$$6\ 6\quad ;P_7=(-A_1+A_3)(B_1+B_2);$$

Complexity? O(n^log7=2.81)

$$C_1=-P_2+P_4+P_5+P_6;C_2=P_1+P_2$$
$$C_3=P_3+P_4;\quad C_4=P_1-P_3+P_5+P_7$$

Any Question?