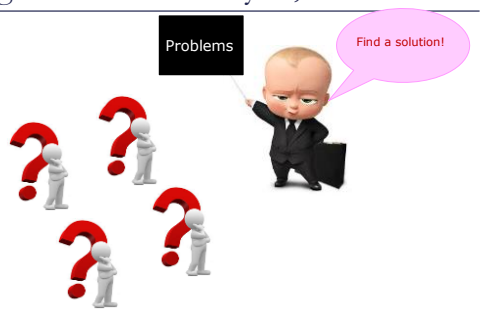
Preview

- Algorithm & Analyzing Algorithm with sorting Algorithms
  - Insertion Sort
  - Selection Sort
  - Bubble Sort
- Asymptotic Notation
  - Big Θ Notation
  - Big O Notation
  - Big Ω Notation

Algorithm and Analysis, What for ?



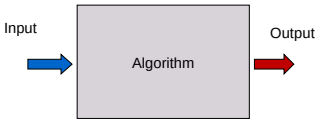
Algorithm and Analysis, What for ?

- Solvable or Unsolvable?
- If solvable, think and find a solution
- Prove it is a correct solution
- Is it enough?

Algorithm and Analysis, What for ?

Algorithm

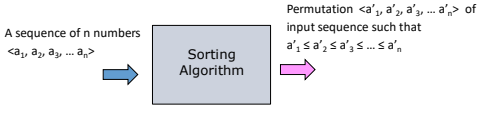
- What is algorithm?
  - An algorithm is any <u>well-defined computational procedure</u> that takes some value, or set of values, as input and produces some value, or set of values, as output.



Algorithm

- An algorithm is <u>a sequence of computational steps</u> that transform the input into the output.
- An algorithm is a tool for solving a well-specified computational problem.

Ex) sorting problem



Algorithm

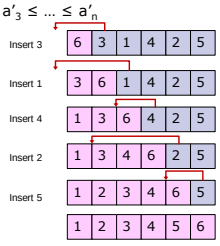
- What is a correct algorithm for solving a problem?
  - An algorithm is said to be correct if it halt with the correct out with every possible input.
- What is the criterion and how to evaluate which algorithm is better?
  - Measure of efficiency is speed – How long an algorithm takes to produce output.
  - By analyzing algorithms mathematically

Analyzing algorithms

- Model of Implementation
  - One processor random access machine – instructions are executed one after another (no concurrent execution!)
  - The Data (input or output) are integer and floating point.
  - Since the time taken by an algorithm grow with the size of the input, the running time of an algorithms can be described as **a function of the size of the input**.
  - The running time of an algorithm on a particular input is the number of primitive operations or steps executed.

Analyzing Sorting Algorithms
(Insertion Sort)

- Input: A sequence of n numbers $<a_1, a_2, a_3, \ldots a_n>$
- Output: Permutation $<a'_1, a'_2, a'_3, \ldots a'_n>$ of input sequence such that $a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$

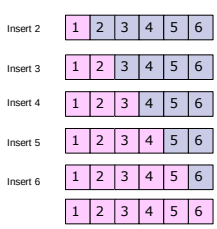


Analyzing Sorting Algorithms
(Insertion Sort)

- For each iteration, a element is placed into right position in sorted sub list.

```
Insertion sort (A)
{
  for j = 2 to length of array A do        C1
   {
      key = A[j]                           C2
      i = j – 1                            C3
      while i > 0 and A[i] >key do         C4
      {
           A[i +1] = A[i]                  C5
           i = i –1                        C6
      }
      A[i +1] = key                        C7
   }
}
```


Analyzing Sorting Algorithms
(Insertion Sort: Best Case)



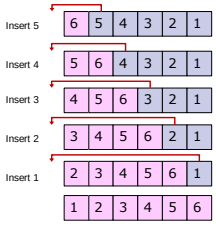

```
Insertion sort (A)
{
  for j = 2 to length of array A do
   {
      key = A[j]
      i = j – 1
      while i > 0 and A[i] >key do
      {
           A[i +1] = A[i]
           i = i –1
      }
      A[i +1] = key
   }
}
```


Analyzing Sorting Algorithms
(Insertion Sort: Best Case)

```
Insertion sort (A)
{
  for j = 2 to length of array A do        C1(n – 1)
   {
      key = A[j]                           C2(n – 1)
      i = j – 1                            C3(n – 1)
      while i > 0 and A[i] >key do         C4(n – 1)
      {
           A[i +1] = A[i]                  0
           i = i –1                        0
      }
      A[i +1] = key                        C7(n – 1)
   }
}
```

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$
$$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

Analyzing Sorting Algorithms
(Insertion Sort: Worst Case)




```
1   Insertion sort (A)
2   {
3     for j = 2 to length of array A do
4     {
5         key = A[j]
6         i = j – 1
7         while i > 0 and A[i] >key do
8         {
9                 A[i +1] = A[i]
10                i = i –1
11        }
12        A[i +1] = key
13    }
14  }
```


Analyzing Sorting Algorithms
(Insertion Sort: Worst Case)

```
1   Insertion sort (A)
2   {
3     for j = 2 to length of array A do        C1(n − 1)
4     {
5         key = A[j]                           C2(n − 1)
6         i = j – 1                            C3(n − 1)
7         while i > 0 and A[i] >key do         C4(n(n−1)/2 − 1)
8         {
9             A[i +1] = A[i]                   C5(n(n−1)/2)
10            i = i –1                         C6(n(n−1)/2)
11        }
12        A[i +1] = key                        C7(n − 1)
13    }
14  }
```

$$T(n) = \left(\frac{C_4}{2} + \frac{C_5}{2} + \frac{C_6}{2}\right)n^2 + \left(C_1 + C_2 + C_3 + \frac{C_4}{2} - \frac{C_5}{2} - \frac{C_6}{2} + C_7\right)n - (C_1 + C_2 + C_3 + C_4 + C_7)$$

Basic Sorting Algorithms
(Insertion Sort Analysis)

Insertion Sort Analysis

- Best Case (order of growth = n)

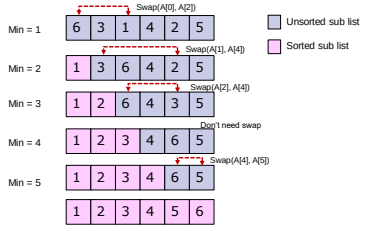
$T(n) = C_1(n-1) + C_2(n-1) + C_3(n-1) + C_4(n-1) + C_7(n-1)$
$= (C_1 + C_2 + C_3 + C_4 + C_7)n - (C_1 + C_2 + C_3 + C_4 + C_7)$

- Worst Case: (order of growth = n^2)

$T(n)$
$= C_1(n-1) + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n(n+1)}{2} - 1\right) + C_5\left(\frac{n(n-1)}{2}\right) + C_6\left(\frac{n(n-1)}{2}\right) + C_7(n-1)$
$= \left(\frac{C_4}{2} + \frac{C_5}{2} + \frac{C_6}{2}\right)n^2 + \left(C_1 + C_2 + C_3 + \frac{C_4}{2} - \frac{C_5}{2} - \frac{C_6}{2} + C_7\right)n - (C_1 + C_2 + C_3 + C_4 + C_7)$

Analyzing Sorting Algorithms
(Selection Sort)

- Input: A sequence of n numbers <a_1, a_2, a_3, … a_n>
- Output: Permutation <a'_1, a'_2, a'_3, … a'_n> of input sequence such that $a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$

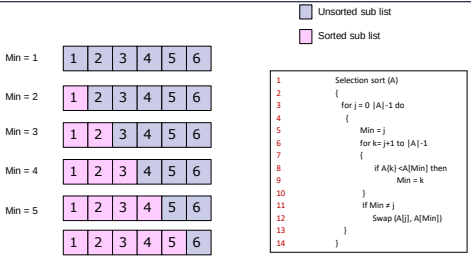


Analyzing Sorting Algorithms
(Selection Sort)

- For each iteration, a minimum is found in the sub list and it is placed into right position in sorted sub list.

```
1   Selection sort (A)
2   {
3     for j = 0 |A|-1 do                  C1
4     {
5         Min = j                         C2
6         for k= j+1 to |A|-1             C3
7         {
8             if A{k} <A[Min] then        C4
9                 Min = k                 C5
10        }
11        If Min ≠ j                      C6
12            Swap (A[j], A[Min])         C7
13    }
14  }
```


Analyzing Sorting Algorithms
(Selection Sort Analysis for Best Case)



Analyzing Sorting Algorithms
(Selection Sort Analysis for Best Case)

- For each iteration, a minimum is found in the sub list and it is placed into right position in sorted sub list.

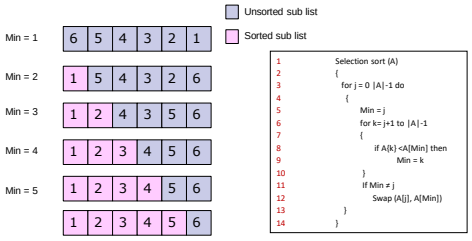
```
Selection sort (A)
{
  for j = 0 |A|-2 do                          C1n
   {
      Min = j                                 C2(n − 1)
      for k= j+1 to |A|-1                     C3(n − 1)
      {
         if A[k] <A[Min] then                 C4 n(n − 1)/2
            Min = k
      }
      If Min ≠ j                              C6(n − 1)
         Swap (A[j], A[Min])
   }
}
```


Analyzing Sorting Algorithms
(Selection Sort Analysis for Worst Case)

Unsorted sub list
Sorted sub list

Min = 1	6	5	4	3	2	1
Min = 2	1	5	4	3	2	6
Min = 3	1	2	4	3	5	6
Min = 4	1	2	3	4	5	6
Min = 5	1	2	3	4	5	6
	1	2	3	4	5	6

```
Selection sort (A)
{
  for j = 0 |A|-1 do
   {
      Min = j
      for k= j+1 to |A|-1
      {
         if A[k] <A[Min] then
            Min = k
      }
      If Min ≠ j
         Swap (A[j], A[Min])
   }
}
```



Analyzing Sorting Algorithms
(Selection Sort Analysis for Worst Case)

- For each iteration, a minimum is found in the sub list and it is placed into right position in sorted sub list.

```
Selection sort (A)
{
  for j = 0 |A|-2 do                          C1n
   {
      Min = j                                 C2(n − 1)
      for k= j+1 to |A|-1                     C3(n − 1)
      {
         if A[k] <A[Min] then                 C4 n(n − 1)/2
            Min = k                           C5 n(n − 1)/2
      }
      If Min ≠ j                              C6(n − 1)
         Swap (A[j], A[Min])                  C6(n − 1)
   }
}
```


Analyzing Sorting Algorithms
(Selection Sort Analysis)

Selection Sort Analysis

- Best Case (order of growth = n^2)

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n(n-1)}{2}\right) + C_6(n-1)$$
$$= \frac{C_4}{2}n^2 + \left(C_1 + C_2 + C_3 - \frac{C_4}{2} + C_6\right)n - (C_2 + C_3 + C_6)$$

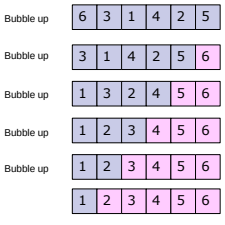
- Worst Case (order of growth = n^2)

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n(n-1)}{2}\right) + C_5\left(\frac{n(n-1)}{2}\right) + C_6(n-1) + C_7(n-1)$$
$$= \left(\frac{C_4}{2} + \frac{C_5}{2}\right)n^2 + \left(C_1 + C_2 + C_3 - \frac{C_4}{2} - \frac{C_5}{2} + C_6 + C_7\right)n - (C_2 + C_3 + C_6 + C_7)$$

Analyzing Sorting Algorithms
(Bubble Sort Analysis)

- Input: A sequence of n numbers $<a_1, a_2, a_3, \ldots a_n>$
- Output: Permutation $<a'_1, a'_2, a'_3, \ldots a'_n>$ of input sequence such that $a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$

Bubble up	6	3	1	4	2	5
Bubble up	3	1	4	2	5	6
Bubble up	1	3	2	4	5	6
Bubble up	1	2	3	4	5	6
Bubble up	1	2	3	4	5	6
	1	2	3	4	5	6

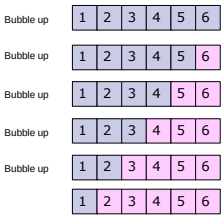


Analyzing Sorting Algorithms
(Bubble Sort Analysis)

- For each iteration, larger element move up (or smaller element move down).

```
BubbleSort(A)
{
  for j = |A|-1 down to 0 do                  C1
   {
      for k=2 to j do                         C2
      {
         If A[k-1] > A[k]                     C3
            Swap (A[k-1], A[k])               C4
      }
   }
}
```

Analyzing Sorting Algorithms
(Bubble Sort Analysis: Best Case)



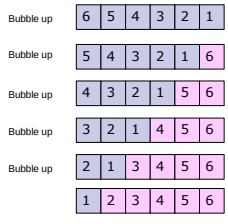

```
1   BubbleSort(A)
2   {
3     for j = |A|-1 down to 0 do
4     {
5         for k=2 to j do
6         {
7             If A[k-1] > A[k]
8                 Swap (A[k-1], A[k])
9         }
10    }
11  }
```


Analyzing Sorting Algorithms
(Bubble Sort Analysis: Best Case)

- For each iteration, larger element move up (or smaller element move down).

```
1   BubbleSort(A)
2   {
3     for j = |A|-1 down to 0 do      <-- C1 n
4     {
5         for k=2 to j do             <-- C2 (n − 1)
6         {
7             If A[k-1] > A[k]        <-- C3 n(n − 1)/2
8                 Swap (A[k-1], A[k])
9         }
10    }
11  }
```


Analyzing Sorting Algorithms
(Bubble Sort Analysis: Worst Case)




```
1   BubbleSort(A)
2   {
3     for j = |A|-1 down to 0 do
4     {
5         for k=2 to j do
6         {
7             If A[k-1] > A[k]
8                 Swap (A[k-1], A[k])
9         }
10    }
11  }
```


Analyzing Sorting Algorithms
(Bubble Sort Analysis: Worst Case)

- For each iteration, larger element move up (or smaller element move down).

```
1   BubbleSort(A)
2   {
3     for j = |A|-1 down to 0 do      <-- C1 n
4     {
5         for k=2 to j do             <-- C2 (n − 1)
6         {
7             If A[k-1] > A[k]        <-- C3 n(n − 1)/2
8                 Swap (A[k-1], A[k]) <-- C4 n(n − 1)/2
9         }
10    }
11  }
```


Basic Sorting Algorithms
(Bubble Sort Analysis)

Bubble Sort Analysis

- Best Case:(order of growth = n^2)

$$T(n) = C_1 n + C_2(n-1) + C_3\frac{n(n-1)}{2}$$
$$= \frac{C_3}{2}n^2 + \left(C_1 + C_2 - \frac{C_3}{2}\right)n - C_2$$

- Worst Case:(order of growth = n^2)

$$T(n) = C_1 n + C_2(n-1) + C_3\frac{n(n-1)}{2} + C_4\frac{n(n-1)}{2}$$
$$= \left(\frac{C_3}{2} + \frac{C_4}{2}\right)n^2 + \left(C_1 + C_2 - \frac{C_3}{2} - \frac{C_4}{2}\right)n - C_2$$

Analyzing Sorting Algorithms

- Importance of Worst Case Analysis
  - The worst-case running time gives an upper bound on the running time for any input.
  - For some algorithms, the worst case occurs fairly often.
  - The average case is usually as bad as the worst case.

Analyzing Sorting Algorithms

- Instead of using exact running time, we can use simplifying abstractions to ease our analysis of algorithms.
- We can use asymptotic notation for describing the speed of a algorithm as order of growth based on the input size n.
- Ex) Insertion sort
  - best case: order of growth = n
  - worst case: order of growth = n^2

Asymptotic Notations (Big - Θ)

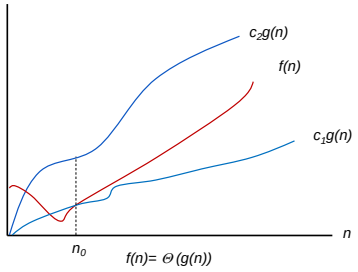
Θ -Notation

- For a given function *g(n)*, we denote by Θ(*g(n)*) the set of functions such that

  Θ(*g(n)*) = {*f(n)* |there exist any positive constant c_1, c_2 and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$}

- *f(n)* = Θ(*g(n)*) indicate a function *f(n)* is a member of Θ(*g (n)*). - *g(n)* is *asymptotically tight bound* for *f (n)*.

Asymptotic Notations (Big - Θ)



Asymptotic Notations (Big - Θ)

Ex) Show $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

We need to find out positive c_1, c_2 and n_0 such that

$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$ for all $n \ge n_0$

$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$

$c_1 \le \frac{1}{2} - \frac{3}{n}$ is hold with $c_1 \le \frac{1}{14}$ and $n \ge 7$

$\frac{1}{2} - \frac{3}{n} \le c_2$ is hold with $c_2 \ge \frac{1}{2}$ and $n \ge 1$

$\frac{1}{14}n^2 \le \frac{1}{2}n^2 - 3n \le \frac{1}{2}n^2$ is hold with $n \ge 7$

Asymptotic Notations (Big - Θ)

$let\ f(n) = \frac{1}{2} - \frac{3}{n}\ with\ n > 1$

n	1	2	3	4	5	6	7	8
$f(n)$	$-\frac{5}{2}$	-1	$-\frac{1}{2}$	$-\frac{1}{4}$	$-\frac{1}{10}$	0	$\frac{1}{14}$	..

$c_1 \le \frac{1}{2} - \frac{3}{n}$ is hold with $c_1 \le \frac{1}{14}$ and $n \ge 7$

Asymptotic Notations (Big O)

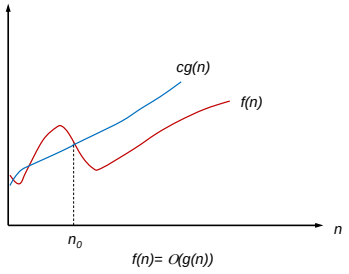
Big O-Notation

- For a given function *g (n)*, we denote by **O**(*g (n)*) the set of functions such that,

  O(*g(n)*)= {*f (n)* | there exist positive constant c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$}

- We write *f (n)* = **O** (*g(n)*) indicates a function *f (n)* is a member of **O** (*g(n)*). - *g (n)* is *asymptotically upper bound* for *f (n)*.

Asymptotic Notations (Big O)



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

Asymptotic Notations (Big O)

- $f(n) = \Theta(g(n)) \subseteq f(n) = \mathbf{O}(g(n))$,
  - since Θ-notation is stronger notation than **O**-notation.
- Written set-theoretically,

  $\Theta(g(n)) \subseteq \mathbf{O}(g(n))$.

Asymptotic Notations (Big O)

Ex) Show that $17n^2 - 5 = O(n^2)$

Sol)

- Based on big-O notation, we need find two integer c and n_0 such that

  $17n^2 - 5 \leq c\,n^2$ *for all* $n \geq n_0$
- With c = 17 and $n_0 = 1$

  $17n^2 - 5 \leq 17\,n^2$ for all $n \geq 1$

  $\therefore 17n^2 - 5 = O(n^2)$

Asymptotic Notations (Big O)

Ex) Show that $35n^3 + 100 = O(n^3)$

- Based on the Big-O notation, we need find two integer c and n_0 such that

  $35n^3 + 100 \leq c\,n^3$ *for all* $n \geq n_0$
- With c = 36 and $n_0 = 5$

  $35n^3 + 100 \leq 36\,n^3$ *for all* $n \geq 5$

  $\therefore 35n^3 + 100 = O(n^3)$

Asymptotic Notations (Big O)

Ex) Show $6 \times 2^n + n^2 = O(2^n)$

- Based on the Big-O notation, we need find two integer c and n_0 such that

  $6 \times 2^n + n^2 \leq c \times 2^n$ for all $n \geq n_0$
- With c = 7 and $n_0 = 5$

  $6 \times 2^n + n^2 \leq 7 \times 2^n$ for all $n \geq 5$

  $\therefore 6 \times 2^n + n^2 = O(2^n)$

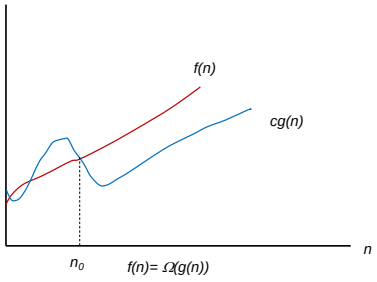
Asymptotic Notations (Big Ω)

Big Ω-Notation

- For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions such that

  $\Omega(g(n)) = \{f(n) \mid$ there exist positive constant c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$
- We write $f(n) = \Omega(g(n))$ indicate a function $f(n)$ is a member of $\Omega(g(n))$ - $g(n)$ is *asymptotically lower bound* for $f(n)$.

Asymptotic Notations (Big Ω)



Asymptotic Notations (Big Ω)

Ex) Show 3n + 2 = Ω(n)

- By Big- Ω notation we need to find out two integer c and n_0 such that
  $3n + 2 \geq cn$ for all $n \geq n_0$
- With c=1 and $n \geq 1$,
  $3n + 2 \geq n$ for all $n \geq 1$
  $\therefore$ 3n + 2 = Ω(n)

Asymptotic Notations (Big Ω)

Ex) Show $2^n + n^3 = \Omega(n^{100})$

- By Big- Ω notation we need to find out two integer c and n_0 such that
  $2^n + n^3 \geq c\, n^{100}$ for all $n \geq n_0$
- With c = 1 and large enough n_0,
  $2^n + n^3 \geq n^{100}$ for all $n \geq n_0$ is true, since left hand side glows exponentially and right hand side glows by polynomial