

Chater 7



- Goal
- Factors of program's efficiency
- Bubble sort algorithm
- Efficiency algorithm
- algorithm Efficiency
- Algorithm analysis
- simple algorithm analysis
- bubble sort algorithm analysis
- Big-Oh

Weekly Objectives

- ◆ This week, we learn how to analyze the efficiency of our program
 - ◆ Algorithm analysis
- ◆ Objectives are
 - ◆ Memorizing the definition and the rules of the big-Oh notation
 - ◆ Understanding what determines the efficiency of programs
 - ◆ Understanding simple algorithms
 - ◆ Memorizing the insert and the delete of lists, stacks, and queues
 - ◆ Memorizing the bubble sort
 - ◆ Able to apply the big-Oh notation analysis to programs

- 알고리즘 분석
- big-Oh 사용법
- 프로그램 효율적으로 작성 이해
- 쉬운 알고리즘
- 프로그램 능력을 big-Oh를 사용해서 표기

Factors of program's efficiency

◆ Algorithm

- ◆ A clearly specified set of simple instructions to be followed to solve a problem
 - ◆ Takes a set of values as inputs
 - ◆ Produces a set of values as outputs

◆ Specified in

- ◆ English
- ◆ A computer program
- ◆ Pseudo-code

◆ Data structures

- ◆ Methods of organizing data

◆ Program

- ◆ = algorithms + data structures

N iterations
with Array
List



Insertion
Algorithm



$\log N$
iterations
with BST

알고리즘:

명확히 정의된 심플한 명령의 과정

Pseudo-code

사람의 언어로 적혀있지만 컴퓨터의 프로그램을 짜기위한 알고리즘

프로그램이란

알고리즘 + 자료구조

프로그램의 효율성을 결정짓는건

알고리즘 자료구조

Bubble sort algorithm

Bubble sort algorithm

◆ Examples of algorithms

- ◆ Insertion, deletion, search of linked lists, stacks, queues...
- ◆ Sorting of linked lists...
 - ◆ Various sorting methods
 - ◆ Bubble sort, Quick sort, Merge sort...

◆ Bubble Sort(list)

- ◆ For itr1=0 to length(list)
 - ◆ For itr2=itr1+1 to length(list)
 - ◆ If list[itr1] < list[itr2]
 - ◆ Swap list[itr1], list[itr2]
- ◆ Return list

◆ This program uses

- ◆ Data structure: List
- ◆ Algorithm: Bubble sort

```
import random

N = 10
lstNumbers = range(N)
random.shuffle(lstNumbers)

print lstNumbers

def performSelectionSort(lstNumbers):
    for itr1 in range(0, N):
        for itr2 in range(itr1+1, N):
            if lstNumbers[itr1] < lstNumbers[itr2]:
                lstNumbers[itr1], lstNumbers[itr2] = \
                    lstNumbers[itr2], lstNumbers[itr1]
        return lstNumbers

print performSelectionSort(lstNumbers)
```

```
[2, 5, 0, 3, 3, 3, 1, 5, 4, 2]
[5, 5, 4, 3, 3, 3, 2, 2, 1, 0]
```

insert , delete 와 다른 sorting

다양한 sort방법중 하나

Bubble sort algorithm

Example of bubble sort execution

◆ Let's observe the execution of the bubble sort

```
import random  
  
N = 10  
lstNumbers = range(N)  
random.shuffle(lstNumbers)
```

```
print lstNumbers
```

```
def performSelectionSort(lstNumbers):  
    for itr1 in range(0, N):  
        for itr2 in range(itr1+1, N):  
            if lstNumbers[itr1] < lstNumbers[itr2]:  
                lstNumbers[itr1], lstNumbers[itr2] = \  
                    lstNumbers[itr2], lstNumbers[itr1]  
    return lstNumbers
```

```
print performSelectionSort(lstNumbers)
```

[2, 5, 0, 3, 3, 3, 1, 5, 4, 2]

→ (itr1 = 0, itr2=1..9) = 9 iterations

→ (itr1 = 0, itr2 = 1)

→ 2 < 5, Hit and swap!!!

→ list[0] = 5, list[1] = 2 from now

→ (itr1 = 0, itr2 = 2)

→ 5 < 0, No hit

→ (itr1 = 0, itr2 = 3)

→ 5 < 3, No hit

→

→ (itr1 = 1, itr2=2..9) = 8 iterations

→

→

→ (itr1 = 8, itr2=9..9) = 1 iterations

→

◆ Total iterations

◆ = 9+8+....+1

◆ = 45 iterations

◆ $= \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$

$((N*N)/2) - (n/2) = \text{total iterations}$

Why do we care about efficiency?

- ◆ Writing a working program is not good enough
 - ◆ The program could be inefficient
 - ◆ If the program runs on a large data, the running time becomes a big issue
 - ◆ Sometimes, a program may not be usable because of the efficiency
 - ◆ Imagine a transaction system of a financial company
 - ◆ 1 transaction = 0.001 sec
 - ◆ 10 transactions by 10,000 account holders = 100 sec
 - ◆ Side effect
 - If there is no reaction from the system, the users click the request again!
 - Increased requests when there is a delay
 - ◆ Imagine a bubble sorting function for bank accounts
 - ◆ 10,000 accounts → roughly 50,000,000 iterations for sorting
 - ◆ Therefore, we need a guarantee of the worst-case scenario
 - ◆ The worst-case running time of a single transaction
 - ◆ The worst-case transaction request numbers of a single day

최악의 경우에 얼마만큼의 효율을 낼 수 있는지 파악할 수 있어야 함

worst-case running time of a single transaction

큰 숫자에 단순한 버블소트를 사용하면 많은 시간이 걸린다

Definition of Algorithm Analysis

- ◆ Analyzing an algorithm
 - ◆ Estimating the resources that the algorithm requires
 - ◆ Memory
 - ◆ Communication bandwidth
 - ◆ Computational time (the most important resource in the most of cases)
- ◆ Factors affecting the running time
 - ◆ Computer used for executions
 - ◆ Algorithms
 - ◆ Data structures
 - ◆ Input data size
- ◆ After analyzing the algorithms
 - ◆ We estimate the worst-case of the costs by the factors
 - ◆ i.e. Computational time by input data size
 - ◆ i.e. Iterations by input data size

알고리즘 분석

알고리즘이 요구하는 자원을 추측

자원 : 메모리 통신 시간

running time에 영향을 주는 요인

- 컴퓨터 성능
- 알고리즘
- 자료구조
- 데이터 사이즈

Simple algorithm analysis

```
def calculateIntegerRangeSum(intFrom, intTo):  
1   intSum = 0  
2   for itr in range(intFrom, intTo):  
3       intSum = intSum + itr  
4   return intSum  
  
print calculateIntegerRangeSum(0, 10)
```

- ◇ Line 1 to 4
 - ◇ Line 1 : 1 iteration
 - ◇ Line 2, 3:
(intTo-intFrom) iterations X 2 lines= N iterations X 2lines
= 2N iterations
 - ◇ Line 4: 1 iteration
- ◇ Total # of iterations = 2N+2 iterations= **$O(N)$**

Bubble sort algorithm analysis

```
def performBubbleSort(lstNumbers):  
    1 for itr1 in range(0, N):  
        2     for itr2 in range(itr1+1, N):  
            3         if lstNumbers[itr1] < lstNumbers[itr2]:  
                4             lstNumbers[itr1], lstNumbers[itr2] = \  
                    lstNumbers[itr2], lstNumbers[itr1]  
    5 return lstNumbers  
  
print performBubbleSort(lstNumbers)
```

Line 1 to 5

- Line 1 : N iterations
- Line 2, 3, 4 : $N-i$ iterations (i is from 0 to $N-1$) X 3 lines
 - 1 to N , 2 to N , ..., $N-1$ to N
 - In other words, $(\sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} 1)$ iterations X 3 lines
 - Assuming that "if" always results in true
- Line 4: 1 iteration

Total # of iterations = $\frac{3}{2}n^2 - \frac{3}{2}n + n + 1$ iterations = $O(N^2)$

Asymptotic notation: Big-Oh

- ◆ What do $O(N)$ and $O(N^2)$ mean?
- ◆ That's the Big-Oh notation
 - ◆ Notation to show the **worst-case** running time
 - ◆ Do you remember?
 - ◆ Assuming that "if" always results in true
 - ◆ So, this is a worst scenario for the run-time
 - ◆ Because the program should run the statements in the "if" block
- ◆ Definition of the Big-Oh notations
 - ◆ $f(N) = O(g(N))$
 - ◆ There are positive constants c and n_0 such that
 - ◆ $f(N) \leq c g(N)$ when $N \geq n_0$
 - ◆ The growth rate of $f(N)$ is less than or equal to the growth rate of $g(N)$
 - ◆ $g(N)$ is an upper bound on $f(N)$

점근 표기법은 알고리즘 효율을
따지기 위한 도구

worst case 실행속도를 보기 위한
표기법