

알고리즘

개요

이영석

yslee@cs-cnu.org

학습목표

- 알고리즘 문제 해결 전략 익히기
 - 복잡도
 - 샘플 문제들
 - 완전탐색(Exhaustive search or brute-force)
 - 대안
 - 정렬
 - 자료구조
 - (재귀)

코딩 테스트 문제 해결의 이해

- 정확한 입출력
- 시간제한
 - 예) 복잡도 이해: $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$, $O(n!)$
 - 컴퓨터의 CPU 클럭 1 GHz
 - 대략 1초에 1,000,000,000번 명령 수행 가능
 - $O(n)$ 이면 입력 크기 최대 1,000,000,000
 - $O(n^2)$ 이면 입력 크기 최대 30,000
 - $O(n^3)$ 이면 입력 크기 최대 1,000

Notation	Name	Example
$O(1)$	constant	Determining if a binary number is even or odd; Calculating $(-1)^n$; Using a constant-size lookup table
$O(\log \log n)$	double logarithmic	Number of comparisons spent finding an item using interpolation search in a sorted array of uniformly distributed values
$O(\log n)$	logarithmic	Finding an item in a sorted array with a binary search or a balanced search tree as well as all operations in a Binomial heap
$O((\log n)^c)$ $c > 1$	polylogarithmic	Matrix chain ordering can be solved in polylogarithmic time on a parallel random-access machine .
$O(n^c)$ $0 < c < 1$	fractional power	Searching in a k-d tree
$O(n)$	linear	Finding an item in an unsorted list or in an unsorted array; adding two n -bit integers by ripple carry
$O(n \log^* n)$	$n \log$ -star n	Performing triangulation of a simple polygon using Seidel's algorithm , or the union-find algorithm . Note that $\log^*(n) = \begin{cases} 0, & \text{if } n \leq 1 \\ 1 + \log^*(\log n), & \text{if } n > 1 \end{cases}$
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, quasilinear, or " $n \log n$ "	Performing a fast Fourier transform ; Fastest possible comparison sort ; heapsort and merge sort
$O(n^2)$	quadratic	Multiplying two n -digit numbers by a simple algorithm; simple sorting algorithms, such as bubble sort , selection sort and insertion sort ; (worst case) bound on some usually faster sorting algorithms such as quicksort , Shellsort , and tree sort
$O(n^c)$	polynomial or algebraic	Tree-adjointing grammar parsing; maximum matching for bipartite graphs ; finding the determinant with LU decomposition
$L_n[\alpha, c] = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$ $0 < \alpha < 1$	L-notation or sub-exponential	Factoring a number using the quadratic sieve or number field sieve
$O(c^n)$ $c > 1$	exponential	Finding the (exact) solution to the travelling salesman problem using dynamic programming ; determining if two logical statements are equivalent using brute-force search
$O(n!)$	factorial	Solving the travelling salesman problem via brute-force search; generating all unrestricted permutations of a poset ; finding the determinant with Laplace expansion ; enumerating all partitions of a set

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
<u>Array</u>	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Stack</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Queue</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Singly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Doubly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Skip List</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n \log(n))$
<u>Hash Table</u>	N/A	$\theta(1)$	$\theta(1)$	$\theta(1)$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Binary Search Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Cartesian Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>B-Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Red-Black Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Splay Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>AVL Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>KD Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
<u>Quicksort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
<u>Mergesort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Timsort</u>	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Heapsort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(1)$
<u>Bubble Sort</u>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Insertion Sort</u>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Selection Sort</u>	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Tree Sort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(n)$
<u>Shell Sort</u>	$\Omega(n \log(n))$	$\Theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
<u>Bucket Sort</u>	$\Omega(n+k)$	$\Theta(n+k)$	$O(n^2)$	$O(n)$
<u>Radix Sort</u>	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	$O(n+k)$
<u>Counting Sort</u>	$\Omega(n+k)$	$\Theta(n+k)$	$O(n+k)$	$O(k)$
<u>Cubesort</u>	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$

입력 크기와 실전 복잡도

Table 3.1 Estimating time complexity from input size

Input size	Expected time complexity
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$ or $O(n)$
n is large	$O(1)$ or $O(\log n)$

문제1

- 문제
 - 배열 A에서 제일 큰 원소와 제일 작은 원소를 구하라
- 입력
 - 10만개 이하 정수(≤ 1 만) n개 배열 A
 - 만약 100만, 1000만, 1억 개라면?

문제1-답

- A의 각 원소에 대해서 현재까지 살펴본 원소들 중에서 최대(또는 최소)인지 검사
 - 완전탐색(exhaustive search or brute-force search)
 - max() or min()
 - built-in function
- $O(n)$

문제2

- 문제
 - 배열 A에서 k 번째 작은 (또는 큰) 원소를 구하라.
- 입력
 - 10만개 이하 정수(≤ 1 만) n개 배열 A

문제2-답

- 해결방법1
 - 정렬 후 $A[k-1]$
 - 정렬 복잡도: $O(n \log n)$
- 해결방법2
 - 자료구조 heap을 사용하기
 - heap 자료구조의 복잡도는?

문제2-답

- 해결방법 3
 - 재귀로 해결할 수 있을까?
 - k번째 최소값
 - 1번째 최소값 -> 나머지 중에서 최소값 -> ...
 - 복잡도는? $O(k * n)$

문제3

- 문제
 - 배열 A에서 $|x - y|$ 최대값이 얼마인지 구하라.
- 입력
 - 10만개 이하 정수(≤ 1 만) n개 배열 A

문제3-답

- 해결방법1
 - 모든 2개의 정수 쌍 비교
 - $O(n^2)$
- 해결방법2
 - 최대값과 최소값 찾기? (문제1)
 - 복잡도 $O(n)$

문제4: 최대연속부분합

- 정수 리스트가 주어질 때 연속된 정수들의 최대합을 구하시오.
- 예
 - -1 2 3 -4 5 -> 6

문제4: 최대연속부분합 해결방법

- 모든 n 개의 원소 경우의 수(연속되지 않은 것 포함)
 - n 개의 원소 \rightarrow 부분집합의 개수 $\rightarrow O(2^n)$
- 연속된 부분집합 경우의 수
 - $n*(n-1)/2 \rightarrow O(n^2)$
- 이전 최대값 기억하기
 - $O(n)$
 - -1 2 3 -4 5 \rightarrow 6
 - 이전 최대값, 현재 인덱스의 값, 현재 인덱스의 값+이전최대값

정리

- 기본적인 복잡도
- 문제해결방법의 이해
 - 완전탐색