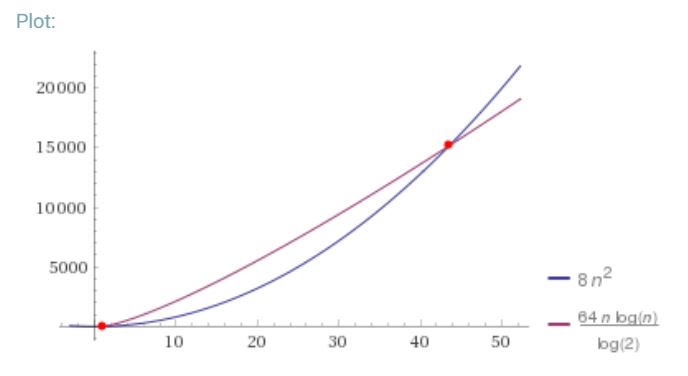
**20171614**

**1.2-2**

8\*n^2 과 64\*n\*lg(n)이 만나는 점은 대략



n 1.1, n 43.5593 이다.

그러므로 n값이 1초과 43이하일때 삽입정렬이 병합정렬보다 빠르다.

**1.2-3**

100\*n^2에서 2^n을 빼면 다음과 같은 값이 나온다.

|  |
| --- |
| >>> def f(n):  ... return 100 \* n\*\*2 - 2\*\*n  ...  >>> f(1)  98  >>> f(2)  396  …  >>> f(14)  3216  >>> f(15)  -10268  >>> |

14와 15사이에 근이 있다는 것을 알 수 있으므로

n=15이상일때 100\*n^2인 알고리즘이 더 빠르다.

**2.1-3**

Input: A sequence of n numbers A =<a0, a1, a2,…, an>and a value v

Output: An index i such that v = A[i] or the special value NIL if v does not appear in A.

|  |
| --- |
| Linear-Search(A, v):  result = Nil  for i = 0 to A.length  if v == A[i]  result = i  return result  return result |

Loop Invariant:

1. Initialization: 루프의 첫 반복이 시작되기전, i가 첫번째 인덱스일 때 검사한게 아무것도 없으므로 result값은 None이다.

2. Maintenance: for의 body부분은 A[i] 가 v값과 다를 때 i가 1증가하여 바로 옆의 인덱스가 되어 for루프의 다음 반복에서도 루프 불변성이 유지된다. 찾은경우 인덱스를 리턴하고 루프를 종료한다.

3. Termination: 값 v를 갖는 인덱스가 없을 때 Nil값을 리턴하거나

값 v를 갖는 인덱스가 있을 때 i값을 리턴한다.

두 경우 모두 원하던 출력값을 리턴하므로 이 알고리즘은 타당하다.

**2.2-1**

Express the function n3/1000 – 100\*n^2 -100n + 3 in terms of ‚ Θ-notation.

= Θ(n^3)

**2.2-3**

Consider linear search again (see Exercise 2.1-3). How many elements of the input sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array? How about in the worst case? What are the average-case and worst-case running times of linear search in ‚ Θ -notation? Justify your answers

Average case: Θ(n)

1번째 조사에서 찾을경우 1 step, 1/n 확률

2번째 조사에서 찾을경우 2 step, 1/n 확률. …

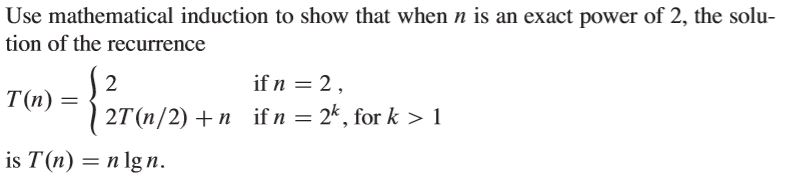
n번째 조사에서 찾을경우 n step, 1/n 확률 이므로 평균 조사 횟수는 (1+2+3+..+n )/n

1+2+…+n = n(n+1)/2 -> (n+1)/2 = Θ(n)의 복잡도를 가진다.

Worst case: Θ(n)

최악의 경우는 모든 원소를 조사했을 때, 즉 배열의 길이 n의 시간이 걸린다

**2.3-3**



Induction

1. n = 1 일때, T(2) = 2 = 2 \* lg(2) 이므로 참이다.

2. n = k 일때 T(2^k) = 2^k \* lg 2^k 이 참이라고 가정하자.

3. n = k+1 일때 T(2^(k+1)) = 2 \* T(2^k) +2^(k+1)

= 2 \* 2^k\* lg 2^k +2^(k+1)

= 2^(k+1) \* (lg 2^k +1)

= 2^(k+1) \* lg(2^(k+1)) 이므로 참이다.

**2.3-5**

Referring back to the searching problem (see Exercise 2.1-3), observe that if the sequence A is sorted, we can check the midpoint of the sequence against v and eliminate half of the sequence from further consideration. The binary search algorithm repeats this procedure, halving the size of the remaining portion of the sequence each time. Write pseudocode, either iterative or recursive, for binary search. Argue that the worst-case running time of binary search is Θ(lgn).

|  |
| --- |
| //ex) Binary-Search(A, v, 0, A.length)  Binary-Search(A, v, left, right)  if left > right  return Nil  mid = (right + left) / 2  if A[mid] == v  return mid  if A[mid] > v  return Binary-Search(A, v, left, mid)  if A[mid] < v  return Binary-Search(A, v, mid+1, right) |

Worst-case인 경우는 v값이 배열 A안에 존재 하지않았을 때 이다.

Binary-Search를 아래와 같이 right < left가 될때까지 시행하여 step수, 입력크기를 나타내보면

right – left +1: n -> n/2 -> n/4 -> n/8 -> n/16 … 0 ..

앞에 보았던 연습문제2.3-3과 merge-sort와 유사하다는 것을 알 수 있으므로 다음과 같다.

T(n) = T(n/2) + c, 즉 Θ(lgn).

**3.1-2**

Show that for any real constants a and b, where b > 0,



Θ (n^b)이 되려면 C1 \* n^b <= (n + a)^b <= C2 \* n^b 이 식을 만족 해야한다.

1. O(n^b): n0 >= |a|라고 가정하면 n >= n0 에서 다음을 만족한다.

(n+a)^b <= (2n)^b, C2 = 2^b.

2. Ω(n^b): n0 >= 2|a|라고 가정하면, n0 – |a| >= |a|, n0 - n0/2<=n0 -|a|, n0/2 <= n0 - |a|

n+a >= n0 – |a| >= n0/2 이므로

(n+a)^b >= (n/2)^b, C1 = (1/2)^b

따라서 n0= 2|a|, C2 = 2^b, C1 = (1/2)^b 일 때 Θ (n^b)이다.

**3.1-3**

Explain why the statement, “The running time of algorithm A is at least O(n^2)” is meaningless.

Big-O notation은 upper bound 이므로 “적어도”라는 말은 의미가 없다.

**3.1-4**



1. 2^n+1 == 2\*2^n이다.

2\*2^n <= 3\*2^n이므로

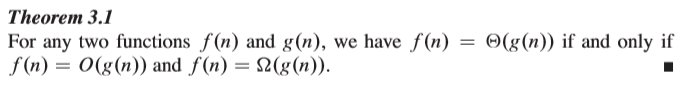
O(2^n)이라 할 수 있다.

2. 2^2n == 2^n \* 2^n==(2^n)^2 이다.

(2^n)^2 <= C\*2^n 에서 , n>=n0에서 이 조건을 만족 시키는 n0이 없으므로

O(2^n)이라 할 수 없다.

**3.1-5**



1. Big- Θ

f(n) = Θ(g(n))이려면 n>n0에서

0<= c1\*g(n) <= f(n) <= c2\*g(n) 을 만족하는 c1, c2, n0이 존재해야 한다.

2. Big-O

f(n) = O(g(n))이려면 n>n1에서

n>n1에서 0 <= f(n) <= c2\*g(n)을 만족하는 c2와 n1이 존재

3. Big- Ω

f(n) = Ω(g(n))이려면 n>n0에서

n>n2에서 0<= c1\*g(n) <= f(n)을 만족하는 c1, n2가 존재

1,2,3의 정의에따라 n1, n2중 큰값이 n0와 동일하면

정리3.1이 성립하다는 것을 증명 할 수 있다.