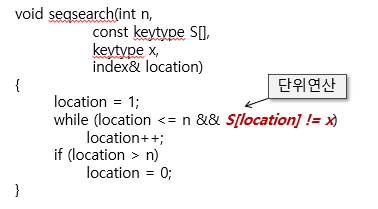
REPORT

(알고리즘 1차 과제)



|  |  |
| --- | --- |
| 제목 | 시간 복잡도 이해 |
| 제출일자 | 20.09.24 |
| 소속학과 | 컴퓨터공학과 |
| 학번 | 2017305039 |
| 성명 | 신동민 |

■**단위연산(Basic Operation)**

-문제 해결에 대한 명령문

-해결책의 명령문은 다양하지만 효율적인

명령문인 단위연산을 찾아야한다.

-효율적인 알고리즘을 찾는 방법 중 하나인

시간 복잡도에 사용된다.

**EX) 대입연산, 사칙연산, 비교구문, 함수호출**

■**시간 복잡도(time complexity)**

-알고리즘이 어떤 문제를 해결하는데 걸리는 시간

※ 문제해결에 대한 걸리는 시간이지 프로그램 전체의 시간은 아니다.

**-**단위연산이 수행되는 횟수를 입력 크기(n)에 대한 함수로 구하여 분석하고

(총 수행되는 횟수)\*(단위연산 실행시간)을 하면 필요한 시간을 알 수 있다.

**-**외부요소[하드웨어, 프로그래밍 언어]에 따라 실행시간이 달라질 수 있어 명령어의 실행 횟수만을 구한다.

**EX) 6n+3, n2+2n, log n, 1**

■**시간 복잡도(time complexity)의 종류**

**1) 모든 경우 시간 복잡도 분석(Every-case time complexity analysis)**

-입력의 크기가 단위연산의 횟수와 같을 경우, 입력의 값과 상관없이 항상 단위연산의 횟수가 같은 경우, 즉 정직하게 명령문이 정해진만큼 돌아야 문제가 해결되는 경우.

**EX) 배열의 수 더하기 배열의 크기 구하기**

**단위연산: 총합을 계산하기 위한 덧셈 단위연산: 크기를 구하기위한 sizeof()**

**입력크기: n, 배열에 있는 항의 개수 입력크기: n, 배열에 있는 항의 개수**

**T(n) = n T(n) = 1**

**정직하게 n번의 실행을 해야 한다. 입력은 n이지만 단위연산은 1번 실행한다.**

**2) 그렇지 않은 경우 분석**

**2-1)최선의 경우 시간 복잡도(Best-case time complexity analysis)**

-문제에 따라 명령문의 실행 횟수가 달라 질 수 있는데 그중 단위연산을 한번만 해서 해결이 된 경우

※ 운으로 해결책이 빠르게 나온 것이지 매번 이럴 수가 없으니 의미가 없다.

**EX) 순차탐색**

**단위연산: 배열의 값과 키 x와 비교 연산**

**입력크기: n, 배열에 있는 항의 개수**

**T(n) = 1~n 중에 비교 연산이 1번만에 된 경우**

**최선의 경우가 되며 실행시간은 1 \* (단위연산 실행시간)이 된다.**

**2-2)최악의 경우 시간 복잡도(Worst-case time complexity analysis)**

-문제에 따라 명령문의 실행 횟수가 달라 질 수 있는데 그중 해결하고자 하는 값이 없어 해결하지 못했거나 마지막에 해결이 된 경우

**EX) 순차탐색**

**단위연산: 배열의 값과 키 x와 비교 연산**

**입력크기: n, 배열에 있는 항의 개수**

**T(n) = 1~n 중에 비교 연산이 n번만에 된 경우나 해결하지 못했을 경우**

**최악의 경우가 되며 실행시간은 n \* (단위연산 실행시간)이 된다.**

**2-3)평균의 경우 시간 복잡도(Average-case time complexity analysis)**

-문제에 따라 명령문의 실행 횟수가 달라 질 수 있는데 그중 평균적인 실행시간을 구하기 위해 확률을 이용해 나타낸 경우.

※ 알고리즘이 어렵거나 복잡하면 확률을 구하기 힘들어 사용 못할 수 있다.

**EX) 순차탐색**

**단위연산: 배열의 값과 키 x와 비교 연산**

**입력크기: n, 배열에 있는 항의 개수**

**x가 배열 안에 있는 경우**

* 1. **x가 배열의 k번째 있을 확률 = 1/n**
  2. **x가 배열의 k번째 있다면 단위연산의 횟수 = k 임으로 식을 만들면**
  3. ****
  4. **이고, 만약 입력 크기가 9일 때 평균적으로 5번의 명령어를 실행되야 문제를**
  5. **해결할 수 있으며 평균 실행시간은 5 \* (단위연산 실행시간)이 된다.**
  6. **※ x가 배열에 있다고 생각하여 만든 평균 실행 횟수이므로 완벽하지 않다.**
  7. **x가 배열 안에 있거나 없을 경우(없을 경우를 추가한다)**

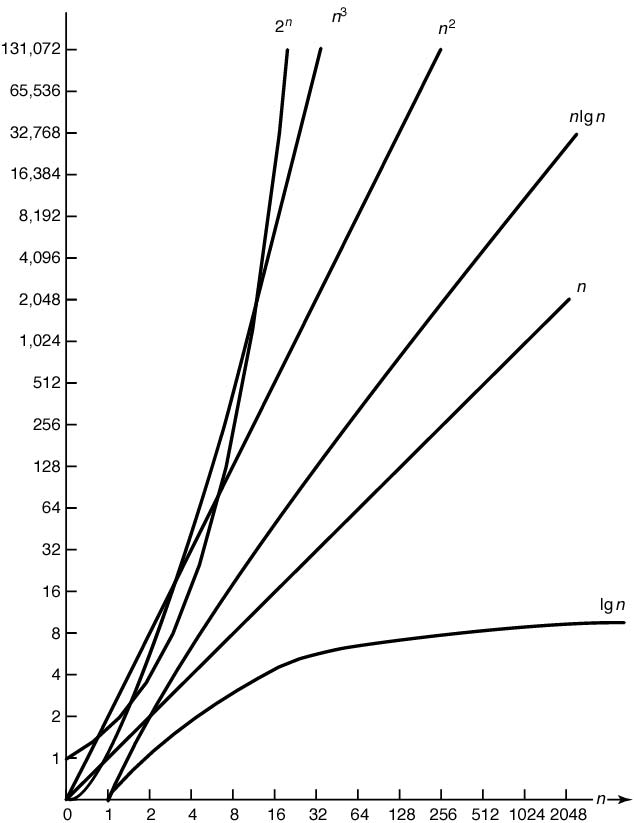
****

**있을 경우: p, 없을 경우: 1-p 두가지로 나누어 각자 확률을 구한 뒤 합해준다.**

**있을 경우는 앞에서 구했으니 넣어주고 없을 경우는 실행을 n번해야하니 곱해준다.**

* 1. **만약 k의 값(p = 1)이 있고 입력 크기가 9일 때를 계산해보면 앞에서 와 같이 평균적으로**
  2. **5번의 명령어가 실행되야 하고 없을 경우(p = 0) n번 실행되어야 한다.**
  3. ■**차수(order)**
  4. -시간 복잡도를 간편하게 표시하기 위하여 사용하는 표기법이며 효율적인 알고리즘을 찾기
  5. 위해 비교하는데 입력의 크기의 증가 양상을 따른 함수를 비교하는 점근 표기법**.**

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **0.01 n2** | **0.1n2** | **0.1n2+n+100** |
| **10** | **1** | **10** | **120** |
| **20** | **4** | **40** | **160** |
| **50** | **25** | **250** | **400** |
| **100** | **100** | **1,000** | **1,200** |
| **1,000** | **10,000** | **100,000** | **101,100** |

* 1. **EX) 시간 복잡도가 0.01n2와 n을 비교해보면 입력의 크기가 적을 땐 0.01n2의 실행시간이**
  2. **짧지만 n >100이상부터는 실행 시간이 길어 지기 때문에 대부분 시간 복잡도가 n인 것보다**
  3. **궁극적으로 우수할 수 없다.**
  4. **그리고 차수가 같은 0.1n2와 0.1n2+n+100랑 비교한다면 처음에는 차이가 나지만 입력의**
  5. **크기가 늘어나면 점차적으로 실행시간의 차이가 줄어들게 됨으로 최고 차수가 실행시간의**
  6. **큰 영향을 끼치며 나머지 항들은 차이가 적다는 것을 알 수 있다.**
  7. **■복잡도 카테고리**

O(1): 상수

O(lg n): 로그(logarithmic)

O(n): 1차(linear)

O(n lg n)

O(n2): 2차(quadratic)

O(n3): 3차(cubic)

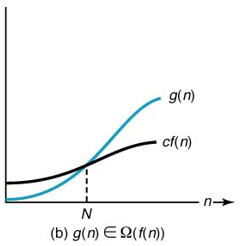
O(2n): 지수(exponential)

O(n!): factorial

-복잡도 카테고리는 차수에 따라 효율적인 알고리즘을 선택하는 순서를 표현한 것이다. 상수가

제일 효율적이며 아래로 내려 갈수록 실행시간이 늘어난다.

※ 적어도 n lg n이상의 차수를 해야 실행시간이 짧아지고 그 이하는 크기가 크면 길어진다.

**Big-오: O(*f*(*n*))**

-빅오 표시법은 c가 양의 실수이며 N은 음이 아닌

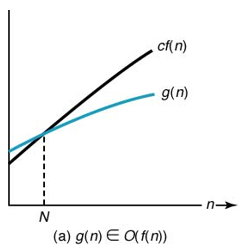
정수에서 특정 n값에서부터 g(n) ≤ c*f*(*n*)으로 점근

적으로 상한이다.즉 적어도 *f*(*n*)보다 항상 짧다.

**EX) O(*f*(n2))이라면**

**g(1), g(lg n): lg n+5, g(n): 11n, g(n lg n): 7lg n+11,**

**g(n2): 3n2+7가 항상 짧다.**

**오메가: Ω(*f*(*n*))**

-오메가 표시법은 c가 양의 실수이며 N은 음이 아닌

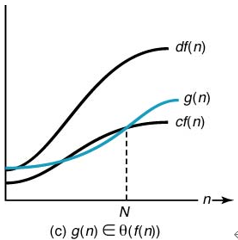
정수에서 특정 n값에서부터 g(n) ≥ c*f*(*n*)으로 점근

적으로 하한이다.즉 적어도 *f*(*n*)보다 항상 길다.

**EX) Ω (*f*(n2))이라면**

***g*(n2): 2n2, *g*(n3): n3+9,**

***g*(2n): 2n+9n, *g*(n!): n!+10가 항상 길다.**

**세타: θ(*f*(*n*))**

-세타 표시법은 c, d가 양의 실수이며 N은 음이 아닌

정수에서 특정 n값에서부터 θ(*f*(*n*)) = O(*f*(*n*)) ∩ Ω(*f*(*n*))

이다. 즉 O(*f*(*n*))보다 항상 짧고 Ω(*f*(*n*))보다 길다.

**EX) θ(*f*(n2))이라면**

**Ω(*f*(n2)) ≤ *g*(n2): 8n2+1 ≤ O(*f*(*n*))가 같은 차수에**

**Ω(*f*(n2))보다 길고 O(*f*(*n*))보다 짧다.**

**※ 여기에서 짧고 길다는 것은 실행시간=단위연산 횟수를 말한다.**

**분할정복**

**부분문제 = 부분해 -> 문제해 구함**

**퀵정렬 2분할, 크기가 일정하지 않는 크기로 감소**

**이진탐색 2분할, 1/2**

**선택문제 2분할, 1/2로 크기가 일정하지 않는 크기로 감소**

**삽입 정렬, 피보나치 수 크기가 1,2개씩 감소**

**재귀 알고리즘(속도는 느린데 간단하고 명료하다)vs반복 알고리즘(속도는 빠른데 이해하기 어렵)**

**-Logic:논리가 간단 O(n log n) -for문 O(n^2)**

**ㅣheap l heap**

**ㅣ----------------- l-----------------**

**ㅣ f() l**

**ㅣ f() l**

**ㅣ f() stack:스택영역(활성레코드)이 넘쳐 오류발생 l f() 반복**

**꼬리재귀 알고리즘(피보나찌) : 함수를 부르는 것 말고는 다른 코드가 없을 때 스택에 쌓인다.**

**-컴파일러가 좋으면 재귀로짜면 알아서 반복문으로 바꿔주기 때문에 좋다.**

**(스택메모리가 적게 들면서 빠르고 코드가 재귀여서 간단하고 명료하다.)**

**시간 복잡도 분석**

**-귀납법(induction) -연역법(deduction)**

**현상들 관찰 진리**

**규칙을 반영 세부적인 상황**

**규칙이 정당화 상황을 파악 항상 참이다.**

**재현식 : 점화식**

**W(n)=w(n/2)+1**

**W(1)=1**

**W(1) = 1**

**W(2) = w(1)+1 =2**

**W(4) = w(2)+1 =3**

**W(n)=lgn+1**

**귀납출발 n = 1이면 lg1 +1 =0+1 =1이맞다.**

**귀납가정 2의 제곱인 양의 정수n이 w(n) = lg n +1이라고 가정**

**귀납단계 w(2n) = lg 2n + 1**

**W(2n) = w(n) +1**

**= lg n +1 +1**

**= lg n + lg 2 +1**

**합병정렬(이진탐색)**

**-** n개의 숫자들을 n/2개씩 2개의 부분문제로 분할하고, 각각의 부분문제를 **재귀적으로 합병 정렬한 후, 2개의 정렬된 부분을 합병하여 정렬 (정복)한다. 🡪 합병 과정이 (문제를) 정복하는 것이다.**

**합병(정렬) \* 층수**

* **분할하는 부분은 배열의 중간 인덱스 계산과 2번의 재귀 호출이므로 O(1) 시간 소요**
* **->N을 분할하는데 k번해야하면 1 = n/2^k 즉 k = lg n(층수)**
* **합병의 수행 시간은 입력의 크기에 비례. 즉, 2개의 정렬된 배열 A와 B의 크기가 각각 n과 m이라면, 최대 비교 횟수= (n+m-1).**
* **->각 층에서 (n/2+n/2-1) O(n)이 된다.**

**O(n) \* lg n = O(n lg n)**

**이건 정렬이 되어있어도 이만큼 시간이 걸림**

**랜덤 알고리즘**

**퀵 정렬(기준값이 랜덤이기에 평균시간 복잡도 사용)**

**-피봇을 기준으로 작은 것과 큰 것을 정렬한 다음 분할함.**

**피봇(퀵정렬의 기준이 되는 수) 값 3개를 뽑아 중간을 피봇으로 한다.**

**-피봇과 low, high를 비교해서 자리를 바꿈 O(n)**

**-최악은 O(n^2)이지만 랜덤이라 확률이 적으며 n/2분할이 최선으로 O(lg n)**

**제일좋은 정렬 O(n lg n) 공간x**

**선택문제(퀵 정렬과 이진탐색을 합침 피봇의 인덱스 기준으로 반을 버린다)**

**-good 분할 bad 분할**

**-랜덤이긴 하지만 처음비교횟수는 좋은 분할을 하고 2가지경우인 2를 곱함**

**2\*(n+3/4n+….) = 2(2n) = O(n)**

**최근접 점(영역을 나누어 그중 가장 근접 점의 후보를 구하고 비교해서 찾음)**

**단, 나누는 구역에서 근접 점이 있을 수 있음 O(n (lg n)^2)**

**x값정렬 O(n log n)**

**최근접 함수 중간 y값정렬 O(n log n) \* 나눈 횟수O(log n)**

**분할정복 주의사항**

**-분할할 시에 입력의 크기가 더 커지면 안됨 피보나찌가 반복문이 더 좋은 이유**

**-정복할 때가 더 중요 정복에 알고리즘을 어떤것을쓰냐에 따라 복잡도가 달라짐 이중포문쓰면**

**N^2\*log 2**