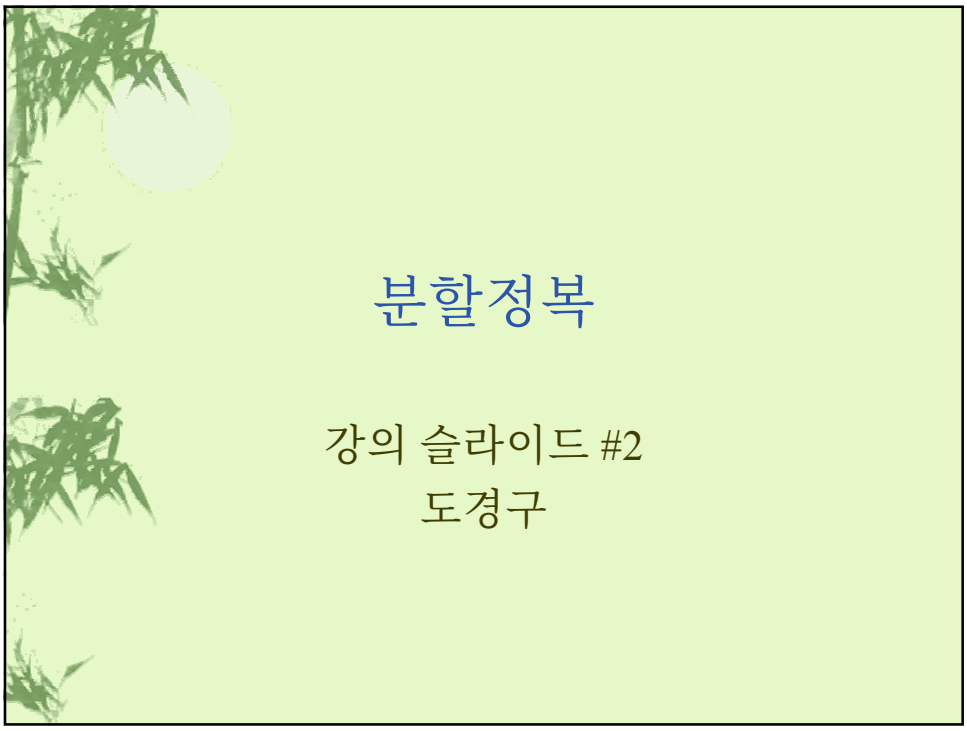

분할정복(Divide-and-Conquer)식 설계 전략

- 분할(Divide): 해결하기 쉽도록 문제를 여러 개의 작은 부분으로 나눈다.
- 정복(Conquer): 나눈 작은 문제를 각각 해결한다.
- 통합(Combine): (필요하다면) 해결된 해답을 모은다.

이러한 문제 해결 방법을 **하향식(top-down)** 접근방법이라고 한다.

이분검색(Binary Search): 재귀 알고리즘

- 문제: 크기가 n인 정렬된 배열 S에 x가 있는지를 결정하라.
- 입력: 자연수 n, 비내림차순으로 정렬된 배열 $S[1..n]$, 찾고자 하는 항목 x
- 출력: *locationout* - x가 S의 어디에 있는지의 위치. 만약 x가 S에 없다면 0
- 설계전략:
  - ✓ x가 배열의 중간에 위치하고 있는 항목과 같으면, "빙고", 찾았다! 그렇지 않으면:
  - ✓ 분할: 배열을 반으로 나누어서 x가 중앙에 위치한 항목보다 작으면 왼쪽에 위치한 배열 반쪽을 선택하고, 그렇지 않으면 오른쪽에 위치한 배열 반쪽을 선택한다.
  - ✓ 정복: 선택된 반쪽 배열에서 x를 찾는다.
  - ✓ 통합: (필요 없음)

2001-03-19 알고리즘 강의 슬라이드 2 3

이분검색(Binary Search): 재귀 알고리즘

```
index location (index low, index high) {
    index mid;

    if (low > high)
       return 0;                          // 찾지 못했음
    else {
       mid = (low + high) / 2             // 정수 나눗셈 (나머지 버림)
       if (x == S[mid])
            return mid;                   //  찾았음
       else if (x < S[mid])
            return location(low, mid-1);  // 왼쪽 반을 선택함
       else
            return location(mid+1, high); // 오른쪽 반을 선택함
    }
}
…
locationout = location(1, n);
...
```

2001-03-19 알고리즘 강의 슬라이드 2 4

관찰사항

- 왜 지역함수 *locationout*을 정의하여 알고리즘을 둘로 나누었을까?
  입력 파라미터 n, S, x는 알고리즘 수행 중 변하지 않는 값이다. 따라서 함수를 재귀호출(recursive call)할 때 마다 이러한 변하지 않는 파라미터를 가지고 다니는 것은 극심한 낭비이다.

2001-03-19 알고리즘 강의 슬라이드 2 5

관찰사항

- 재귀 알고리즘(recursive algorithm)에서 모든 재귀호출이 알고리즘의 마지막(꼬리) 부분에서 이루어 질 때 꼬리 재귀호출(tail recursion)이라고 함 - 그 알고리즘은 반복 알고리즘(iterative algorithm)으로 변환하기가 수월하다. 일반적으로 재귀 알고리즘은 재귀 호출할 때마다 그 당시의 상태를 활성 레코드(activation records) 스택에 저장해 놓아야 하는 반면, 반복 알고리즘은 그럴 필요가 없기 때문에 일반적으로 더 효율적이다(빠르다). 그렇다고 반복 알고리즘의 계산복잡도가 재귀 알고리즘보다 좋다는 의미는 아니다. 반복 알고리즘이 상수적(constant factor)으로만 좋다(빠르다)는 말이다. ML이나 Scheme 같은 함수형 언어는 컴파일러가 자동으로 재귀 프로그램을 반복 프로그램으로 바꾸어 준다.

2001-03-19 알고리즘 강의 슬라이드 2 6

최악의 경우 시간복잡도 분석

- 단위연산: x와 S[mid]의 비교
- 입력 크기: 배열의 크기 n (= $high$ - low + 1)
- 알고리즘을 살펴보면 단위연산으로 설정한 조건 문을 while루프 내부에서 2번 수행하지만, 사실상 비교는 한번 이루어진다고 봐도 된다. 그 이유는: (1) 어셈블리 언어로는 하나의 조건 명령으로 충분히 구현할 수 있기 때문이기도 하고; (2) x를 찾기 전까지는 항상 2개의 조건 문을 수행하므로 하나로 묶어서 한 단위로 취급을 해도 되기 때문이기도 하다. 이와 같이 단위연산은 최대한 효율적으로(빠르게) 구현된다고 일반적으로 가정하여, 1단위로 취급을 해도 된다.

2001-03-19 알고리즘 강의 슬라이드 2 7

최악의 경우 시간복잡도 분석

- **경우 1: 검색하게 될 반쪽 배열의 크기가 항상 정확하게 $\frac{n}{2}$ 이 되는 경우**

  시간복잡도를 나타내 주는 재현식(recurrence)은 다음과 같다.

$$W(n) = W(\tfrac{n}{2}) + 1 \qquad n > 1 \text{ 이고, } n = 2^k (k \geq 1)$$
$$W(1) = 1$$

  이 식의 해는 다음과 같이 구할 수 있다.

$$W(1) = 1$$
$$W(2) = W(1) + 1 = 2$$
$$W(4) = W(2) + 1 = 3$$
$$W(8) = W(4) + 1 = 4$$
$$W(16) = W(8) + 1 = 5$$
$$\cdots$$
$$W(2^k) = k + 1$$
$$\cdots$$
$$W(n) = \lg n + 1$$

2001-03-19 알고리즘 강의 슬라이드 2 8

최악의 경우 시간복잡도 분석

- 이 해가 과연 맞는지 확인하기 위하여 증명해 보자.

  증명: 수학적귀납법:

  귀납출발점: $n = 1$이면, $W(1) = 1 = \lg 1 + 1$.

  귀납가정: 2의 거듭제곱(power)인 양의 정수 n에 대해서, $W(n) = \lg n + 1$라고 가정한다.

  귀납단계: $W(2n) = \lg(2n) + 1$임을 보이면 된다. 재현식을 사용하면,

$$\begin{aligned} W(2n) &= W(n)+1 && \text{재현식에 의해서} \\ &= \lg n+1+1 && \text{귀납가정에 의해서} \\ &= \lg n+\lg 2+1 \\ &= \lg(2n)+1 \end{aligned}$$

2001-03-19 알고리즘 강의 슬라이드 2 9

최악의 경우 시간복잡도 분석

- **경우 2: 일반적인 경우 - 반쪽 배열의 크기는 $\lfloor \frac{n}{2} \rfloor$이 됨**

  $\lfloor y \rfloor$란 y보다 작거나 같은 최대 정수를 나타낸다고 할 때, n에 대해서 가운데 첨자는 $mid = \left\lfloor \frac{1+n}{2} \right\rfloor$이 되는데, 이 때 각 부분배열의 크기는 다음과 같다.

n	왼쪽 부분배열의 크기	mid	오른쪽 부분배열의 크기
짝수	n/2 - 1	1	n/2
홀수	(n-1)/2	1	(n-1)/2

  위의 표에 의하면 알고리즘이 다음 단계에 찾아야 할 항목의 개수는 기껏해야 $\lfloor \frac{n}{2} \rfloor$개가 된다. 따라서 다음과 같은 재현식으로 표현할 수 있다.

$$W(n) = 1 + W\left(\lfloor \tfrac{n}{2} \rfloor\right) \quad n > 1\text{일때}$$
$$W(1) = 1$$

2001-03-19 알고리즘 강의 슬라이드 2 10

최악의 경우 시간복잡도 분석

- 이 재현식의 해가 $W(n)=\lfloor \lg n \rfloor+1$가 됨을 n에 대한 수학적귀납법으로 증명한다.

증명: 수학적귀납법

귀납출발점: $n=1$이면, 다음이 성립한다.

$$\lfloor \lg n \rfloor+1=\lfloor \lg 1 \rfloor+1=0+1=1=W(1)$$

귀납가정: $n>1$이고, $1<k<n$인 모든 k에 대해서, $W(k)=\lfloor \lg k \rfloor+1$가 성립한다고 가정한다.

귀납단계: n이 짝수이면 (즉, $\lfloor \frac{n}{2} \rfloor=\frac{n}{2}$),

$$\begin{aligned}
W(n) &= 1+W(\lfloor \tfrac{n}{2} \rfloor) && \text{재현식에 의해서}\\
&= 1+\lfloor \lg \lfloor \tfrac{n}{2} \rfloor \rfloor+1 && \text{귀납가정에 의해서}\\
&= 2+\lfloor \lg \lfloor \tfrac{n}{2} \rfloor \rfloor\\
&= 2+\lfloor \lg \tfrac{n}{2} \rfloor && n\text{이 짝수이므로}\\
&= 2+\lfloor \lg n-1 \rfloor\\
&= 2+\lfloor \lg n \rfloor-1\\
&= 1+\lfloor \lg n \rfloor
\end{aligned}$$

최악의 경우 시간복잡도 분석

- n이 홀수이면 (즉, $\lfloor \frac{n}{2} \rfloor=\frac{n-1}{2}$),

$$\begin{aligned}
W(n) &= 1+W(\lfloor \tfrac{n}{2} \rfloor) && \text{재현식에 의해서}\\
&= 1+\lfloor \lg \lfloor \tfrac{n}{2} \rfloor \rfloor+1 && \text{귀납가정에 의해서}\\
&= 2+\lfloor \lg \lfloor \tfrac{n}{2} \rfloor \rfloor\\
&= 2+\lfloor \lg \tfrac{n-1}{2} \rfloor && n\text{이 홀수이므로}\\
&= 2+\lfloor \lg(n-1)-1 \rfloor\\
&= 2+\lfloor \lg(n-1) \rfloor-1\\
&= 1+\lfloor \lg(n-1) \rfloor\\
&= 1+\lfloor \lg n \rfloor && n\text{이 홀수이므로}
\end{aligned}$$

따라서, $W(n)=\lfloor \lg n \rfloor+1 \in \Theta(\lg n)$.

합병정렬(Mergesort)

- 문제: *n*개의 정수를 비내림차순으로 정렬하시오.
- 입력: 정수 *n*, 크기가 *n*인 배열 S[1..n]
- 출력: 비내림차순으로 정렬된 배열 S[1..n]
- 보기: 27, 10, 12, 20, 25, 13, 15, 22

합병정렬

- 알고리즘:

```
void mergesort (int n, keytype S[]) {
        const int h = n / 2, m = n - h;
        keytype U[1..h], V[1..m];

        if (n > 1) {
           copy S[1] through S[h] to U[1] through U[h];
           copy S[h+1] through S[n] to V[1] through V[m];
           mergesort(h,U);
           mergesort(m,V);
           merge(h,m,U,V,S);
        }
}
```

합병(Merge)

- 문제: 두개의 정렬된 배열을 하나의 정렬된 배열로 합병하시오.
- 입력: (1) 양의 정수 *h*, *m*, (2) 정렬된 배열 U[1..h], V[1..m]
- 출력: U와 V에 있는 키들을 하나의 배열에 정렬한 S[1..h+m]

2001-03-19 알고리즘 강의 슬라이드 2 15

- 알고리즘:

```
void merge(int h, int m, const keytype U[], const keytype V[], const keytype S[]) {
        index i, j, k;
        i = 1; j = 1; k = 1;
        while (i <= h && j <= m) {
            if (U[i] < V[j]) {
                    S[k] = U[i];
                    i++;
            }
            else {
                    S[k] = V[j];
                    j++;
            }
            k++;
        }
        if (i > h)
            copy V[j] through V[m] to S[k] through S[h+m];
        else
            copy U[i] through U[h] to S[k] through S[h+m];
}
```

2001-03-19 알고리즘 강의 슬라이드 2 16

시간복잡도 분석

- 합병 알고리즘의 최악의 경우 시간복잡도 분석
  - ✓ 단위연산: U[i]와 V[j]의 비교
  - ✓ 입력크기: 2개의 입력 배열에 각각 들어 있는 항목의 개수: h와 m
  - ✓ 분석: $i = h$이고, $j = m - 1$인 상태로 루프(loop)에서 빠져 나가는 때가 최악의 경우로서(V에 있는 처음 $m - 1$개의 항목이 S의 앞부분에 위치하고, U에 있는 h개의 모든 항목이 그 뒤에 위치하는 경우), 이 때 단위연산 의 실행 횟수는 $h + m - 1$이다. 따라서, 최악의 경우 합병하는 시간복잡도는 $W(h,m) = h + m - 1$.

2001-03-19 알고리즘 강의 슬라이드 2 17

시간복잡도 분석

- 합병정렬 알고리즘의 최악의 경우 시간복잡도 분석
  - ✓ 단위연산: 합병 알고리즘 merge에서 발생하는 비교
  - ✓ 입력크기: 배열 S에 들어 있는 항목의 개수 n
  - ✓ 분석: 최악의 경우 수행시간은 $W(h,m) = W(h) + W(m) + h + m - 1$이 된다. 여기서 W(h)는 U를 정렬하는데 걸리는 시간, $W(m)$은 V를 정렬하는데 걸리는 시간, 그리고 $h + m - 1$은 합병하는데 걸리는 시간이다. 정수 n을 $2^k (k \geq 1)$이라고 가정하면, $h = \frac{n}{2}, m = \frac{n}{2}$이 된다. 따라서 최악의 경우 재현식은:

    $W(n) = 2W(\frac{n}{2}) + n - 1 \quad n > 1$이고, $n = 2^k (k \geq 1)$

    $W(1) = 0$ 왜냐하면 합병이 전혀 이루어지지 않으므로

    이 재현식의 해는 아래의 도사정리의 2번을 적용하면, $W(n) = \Theta(n \lg n)$ 이 된다.

2001-03-19 알고리즘 강의 슬라이드 2 18

시간복잡도 분석

- n이 2의 승(power)의 형태가 아닌 경우의 재현식은 다음과 같이 된다.

$$W(n) = W(\lfloor \tfrac{n}{2} \rfloor) + W(\lceil \tfrac{n}{2} \rceil) + n + 1 \qquad n > 1\text{일때}$$
$$W(1) = 0$$

그러나 이 재현식의 정확한 해를 구하기는 복잡하다. 그러나, 앞의 이분검색 알고리즘의 분석에서도 보았듯이, $n = 2^k$라고 가정해서 해를 구하면, 이 재현식의 해와 같은 카테고리의 시간복잡도를 얻게 된다. 따라서 앞으로 이와 비슷한 재현식의 해를 구할 때, $n = 2^k$라고 가정해서 구해도 점근적으로는 같은 해를 얻게 된다.

2001-03-19 알고리즘 강의 슬라이드 2 19

공간복잡도 분석

- 입력을 저장하는데 필요한 만큼 저장장소를 사용하지 않고 정렬하는 알고리즘을 **제자리정렬(in-place sort) 알고리즘**이라고 한다. 위에서 본 합병정렬 알고리즘은 제자리정렬 알고리즘이 아니다. 왜냐하면 입력인 배열 S이외에 U와 V를 추가로 만들어서 사용하기 때문이다.
- 그러면 얼마만큼의 추가적인 저장장소가 필요할까? mergesort를 재귀호출할 때마다 크기가 S의 반이 되는 U와 V가 추가적으로 필요하다. merge 알고리즘에서는 U와 V가 주소로 전달이 되어 그냥 사용되므로 추가적인 저장장소를 만들지 않는다. 따라서 mergesort를 재귀호출할 때마다 얼마만큼의 추가적인 저장장소가 만들어져야 하는지를 계산해 보면 된다. 처음 S의 크기가 n이면, 추가적으로 필요한 U와 V의 저장장소 크기의 합은 n이 된다. 다음 재귀 호출에는 $\frac{n}{2}$의 추가적으로 필요한 총 저장장소의 크기는 $n + \frac{n}{2} + \frac{n}{4} + \cdots = 2n$이다. 결론적으로 이 알고리즘의 공간복잡도는 $2n \in \Theta(n)$이라고 할 수 있다.
- 추가적으로 필요한 저장장소가 n이 되도록, 즉, 공간복잡도가 n이 되도록 알고리즘을 향상시킬 수 있다(다음 절의 알고리즘). 그러나 합병정렬 알고리즘이 제자리정렬 알고리즘이 될 수는 없다.

2001-03-19 알고리즘 강의 슬라이드 2 20

공간복잡도가 향상된 알고리즘

- 합병정렬(Mergesort)
  - ✓ 문제: n개의 정수를 비내림차순으로 정렬하시오.
  - ✓ 입력: 정수 n, 크기가 n인 배열 S[1..n]
  - ✓ 출력: 비내림차순으로 정렬된 배열 S[1..n]
  - ✓ 알고리즘:

```
void mergesort2 (index low, index high) {
        index mid;
        if (low < high) {
            mid = (low + high) / 2;
            mergesort2(low, mid);
            mergesort2(mid+1, high);
            mergesort2(low, mid, high);
         }
}
…
mergesort2(1, n);
...
```

2001-03-19 알고리즘 강의 슬라이드 2 21

공간복잡도가 향상된 알고리즘

- 합병(Merge)
  - ✓ 문제: 두 개의 정렬된 배열을 하나의 정렬된 배열로 합병하시오.
  - ✓ 입력: (1) 첨자 low, mid, high, (2) 부분 배열 S[low..high], 여기서 S[low..mid]와 S[mid+1..high]는 이미 각각 정렬이 완료되어 있음.
  - ✓ 출력: 정렬이 완료된 부분배열 S[1..high]

2001-03-19 알고리즘 강의 슬라이드 2 22

- 알고리즘:

```
void merge2(index low, index mid, index high) {
        index i, j, k;   keytype U[low..high];   // 합병하는데 필요한 지역 배열
        i = low; j = mid + 1; k = low;
        while (i <= mid && j <= high) {
            if (S[i] < S[j]) {
                    U[k] = S[i];
                    i++;
            }
            else {
                    U[k] = S[j];
                    j++;
            }
            k++;
        }
        if (i > mid)
            copy S[j] through S[high] to U[k] through U[high];
        else
            copy S[i] through S[mid] to U[k] through U[high];
        copy U[low] through U[high] to S[low] through S[high];
}
```


도사 정리(The Master Theorem)

- a와 b를 1보다 큰 상수라 하고, $f(n)$을 어떤 함수라 하고, 음이 아닌 정수 n에 대해서 정의된 재현식 $T(n)$이 다음의 형태를 이룬다고 하자.

$$T(n) = a \times T\left(\frac{n}{b}\right) + f(n)$$

그러면 $T(n)$은 다음과 같이 점근적인 한계점(asymptotic bound)을 가질 수 있다.

1. 어떤 상수 $\varepsilon > 0$에 대해서, 만약 $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$이면, $T(n) = \Theta\left(n^{\log_b a}\right)$
2. $f(n) = \Theta\left(n^{\log_b a}\right)$이면, $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$.
3. 어떤 상수 $\varepsilon > 0$에 대해서, 만약 $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$이고, 어떤 상수 $c < 1$과 충분히 큰 모든 n에 대해서, $a \times f\left(\frac{n}{b}\right) \le c \times f(n)$이면, $T(n) = \Theta(f(n))$.

여기서 $\frac{n}{b}$은 $\left\lfloor \frac{n}{b} \right\rfloor$로 여겨도 되고, $\left\lceil \frac{n}{b} \right\rceil$으로 여겨도 된다.

도사정리 적용의 예

- $T(n)=9T(\frac{n}{3})+n$
  여기서 $a=9, b=3, f(n)=n$이고, $n^{\log_b a}=n^{\log_3 9}=\Theta(n^2)$ 이므로, $\varepsilon=1$일 때, $f(n)=O(n^{\log_3 9-\varepsilon})$이라고 할 수 있다. 도사정리 1번을 적용하면, $T(n)=\Theta(n^{\log_3 9})=\Theta(n^2)$ 이 된다.

- $T(n)=T(\frac{2n}{3})+1$
  여기서 $a=1, b=\frac{3}{2}, f(n)=1$이고, $n^{\log_b a}=n^{\log_{\frac{3}{2}} 1}=n^0=\Theta(1)$이므로, $f(n)=\Theta(1)$이라고 할 수 있다. 도사정리 2번을 적용하면, $T(n)=\Theta(1\lg n)=\Theta(\lg n)$이 된다.

도사정리 적용의 예

- $T(n)=3T(\frac{n}{4})+n\lg n$
  여기서 $a=3, b=4, f(n)=n\lg n$이고, $n^{\log_b a}=n^{\log_4 3}=O(n^{0.793})$이므로, $\varepsilon\approx 0.2$일 때, $\varepsilon=1$일 때, $f(n)=\Omega(n^{\log_4 3+\varepsilon})$ 이라고 할 수 있다. 도사정리 3번을 적용할 수 있는지 보기 위해서, 충분히 큰 n에 대해서, $3f(\frac{n}{4})\le c\times f(n)$이 성립하는 1보다 작은 c가 존재하는가를 보아야 한다. 여기서, $c=\frac{3}{4}$이면, $3\frac{n}{4}\lg(\frac{n}{4})\le\frac{3}{4}n\lg n$ 은 충분히 큰 n에 대해서 항상 성립한다. 따라서 $T(n)=\Theta(n\lg n)$이 된다.

- $T(n)=2T(\frac{n}{2})+n\lg n$
  여기서 $a=2, b=2, f(n)=n\lg n$이고, $n^{\log_b a}=n^{\log_2 2}=\Theta(n)$ 이므로, $f(n)=\Omega(n^{\log_2 2+\varepsilon})$ 이라고 할 수 있다. 여기서 도사정리 3번을 적용할 수 있는지 보기 위해서, 충분히 큰 n에 대해서, $2f(\frac{n}{2})\le c\times f(n)$ 이 성립하는 1보다 작은 c가 존재하는가를 보아야 한다. 그러나, $2\frac{n}{2}\lg(\frac{n}{2})\le cn\lg n$에서 충분히 큰 n에 대해서 항상 성립하는 c는 없다. 왜냐하면, 위의 식을 정리하면 $\frac{\lg n-1}{\lg n}\le c$ 가 되고, 어떠한 c를 선택하더라도 이 부등식은 성립할 수 없다. 따라서 도사정리를 이용하여 해를 구할 수 없다. 이런 경우는 다음의 도사보조정리를 이용하여 해를 구할 수 있다.

도사보조정리

- $T(n) = a \times T\left(\frac{n}{b}\right) + f(n)$
  에서, $k \geq 0$인 어떤 k에 대해서 $f(n)$이 $\Theta(n^{\log_b a} \lg^k n)$ 이면
  $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$이 된다. (증명 생략)
- 도사보조정리의 적용의 예
  위의 보기 4번의 해는 $f(n) = \Theta(n \lg n)$이므로 $T(n) = \Theta(n \lg^2 n)$이 된다.

  이 절에서 공부한 도사정리는 재현식을 푸는데 상당히 유용하게 쓰인다. 따라서 보지 않고도 재현식을 푸는데 적용할 수 있을 만큼 이 정리를 완전히 이해하기를 권장한다.

2001-03-19 알고리즘 강의 슬라이드 2 27

빠른정렬(Quicksort)

- 1962년에 영국의 호아(C.A.R. Hoare)의 의해서 고안
- 빠른정렬(Quicksort)란 이름이 오해의 여지가 있음. 왜냐하면 사실 절대적으로 가장 빠른 정렬 알고리즘이라고 할 수는 없기 때문이다. 차라리 “분할교환정렬(partition exchange sort)”라고 부르는 게 더 정확함.
- 보기: 15 22 13 27 12 10 20 25

2001-03-19 알고리즘 강의 슬라이드 2 28

빠른정렬 알고리즘

- 문제: n개의 정수를 비내림차순으로 정렬
- 입력: 정수 n > 0, 크기가 n인 배열 S[1..n]
- 출력: 비내림차순으로 정렬된 배열 S[1..n]
- 알고리즘:

```
void quicksort (index low, index high) {
        index pivotpoint;
        if (high > low) {
            partition(low,high,pivotpoint);
            quicksort(low,pivotpoint-1);
            quicksort(pivotpoint+1,high);
        }
}
```


분할 알고리즘

- 문제: 빠른정렬을 하기 위해서 배열 S를 둘로 쪼갠다.
- 입력: (1) 첨자 low,high, (2) 첨자 low에서 high까지의 S의 부분배열
- 출력: 첨자 low에서 high까지의 S의 부분배열의 기준점(pivot point), pivotpoint
- 알고리즘:

```
void partition (index low, index high, index& pivotpoint) {
        index i, j;
        keytype pivotitem;
        pivotitem = S[low];//pivotitem을 위한 첫번째 항목을 고른다
        j = low;
        for(i = low + 1; i <= high; i++)
            if (S[i] < pivotitem) {
                    j++;
                    exchange S[i] and S[j];
            }
        pivotpoint = j;
        exchange S[low] and S[pivotpoint];    // pivotitem 값을 pivotpoint에 넣는다
}
```

분석

- 분할 알고리즘의 모든 경우를 고려한 시간복잡도 분석
  - ✓ 단위연산: S[i]와 key와의 비교
  - ✓ 입력크기: 부분배열이 가지고 있는 항목의 수, $n = high - low + 1$
  - ✓ 분석: 배열의 첫번째 항목만 제외하고 모든 항목을 한번씩 비교하므로, $T(n) = n - 1$이다.

2001-03-19 알고리즘 강의 슬라이드 2 31

분석

- 빠른정렬 알고리즘의 최악의 경우를 고려한 시간복잡도 분석
  - ✓ 단위연산: 분할알고리즘의 S[i]와 key와의 비교
  - ✓ 입력크기: 배열이 S가 가지고 있는 항목의 수, n
  - ✓ 분석: 이미 비내림차순으로 정렬이 되어 있는 배열을 정렬하려는 경우가 최악의 경우가 된다. 왜 그럴까? 비내림차순으로 정렬되어 있으면 첫번째(기준점) 항목보다 작은 항목은 없으므로, 크기가 n인 배열은 크기가 0인 부분배열은 왼쪽에 오고, 크기가 n-1인 부분배열은 오른쪽에 오도록 하여 계속 쪼개진다. 따라서,

$$T(n) = T(0) + T(n-1) + n - 1$$

그런데, $T(0) = 0$이므로, 재현식은 다음과 같이 된다.

$$T(n) = T(n-1) + n - 1, \; n > 0\text{이면}$$
$$T(0) = 0$$

2001-03-19 알고리즘 강의 슬라이드 2 32

분석

이 재현식을 풀면,

$$T(n) = T(n - 1) + n - 1$$
$$T(n - 1) = T(n - 1) + n - 2$$
$$T(n - 2) = T(n - 3) + n - 3$$
$$\cdots$$
$$T(2) = T(1) + 1$$
$$T(1) = T(0) + 0$$
$$T(0) = 0$$
$$T(n) = 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2}$$

가 되므로, 이미 정렬이 되어 있는 경우 빠른정렬 알고리즘의 시간복잡도는 $\frac{n(n-1)}{2}$ 이 된다는 사실을 알았다. 그러면 시간이 더 많이 걸리는 경우는 있을까? 이 경우가 최악이 경우이며, 따라서 이 보다 더 많은 시간이 걸릴 수가 없다는 사실을 수학적으로 엄밀하게 증명해 보자.

2001-03-19 알고리즘 강의 슬라이드 2 33

- 모든 정수 n에 대해서, $W(n) \le \frac{n(n-1)}{2}$ 임을 증명하시오.

증명: (수학적귀납법)
귀납출발점: $n = 0$일 때, $W(0) \le \frac{0(0-1)}{2}$
귀납가정: $0 \le k < n$인 모든 k에 대해서, $W(k) \le \frac{k(k-1)}{2}$
귀납단계: $W(n) \le \frac{n(n-1)}{2}$

$W(n) \le W(p-1) + W(n-p) + n - 1$ pivotpoint 값이 p인 경우 재현식에 의해서
$\le \frac{(p-1)(p-2)}{2} + \frac{(n-p)(n-p-1)}{2} + n - 1$ 귀납가정에 의해서
$= \frac{p^2 - 3p + 2 + (n-p)^2 - n + p + 2n - 2}{2}$
$= \frac{p^2 + (n-p)^2 + n - 2p}{2}$

여기서 p가 n - 1일 때 최대값을 가진다. 따라서

$$max_{1 \le p \le n-1}(p^2 + (n-p)^2) = 1^2 + (n-1)^2 = n^2 - 2n + 2$$

가 되고, 결과적으로

$$W(n) \le \frac{p^2 + (n-p)^2 + n - 2p}{2} \le \frac{n^2 - 2n + 2 + n - 2}{2} = \frac{n^2 - n}{2} = \frac{n(n-1)}{2}$$

가 된다. 따라서 최악의 경우 시간복잡도는

$$W(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

2001-03-19 알고리즘 강의 슬라이드 2 34

분석

- 빠른정렬 알고리즘의 평균의 경우를 고려한 시간복잡도 분석
  - ✓ 단위연산: 분할알고리즘의 S[i]와 key와의 비교
  - ✓ 입력크기: 배열이 S가 가지고 있는 항목의 수, n
  - ✓ 분석: 배열 안에 있는 항목이 어떤 특정 순으로 정렬이 되어 있는 경우는 사실 별로 없다. 그러므로 분할 알고리즘이 주는 기준점 값은 1부터 n사이의 어떤 값도 될 수가 있고, 그 확률은 모두 같다고 봐도 된다. 따라서, 평균의 경우를 고려한 시간복잡도 분석을 해도 된다. 기준점이 p가 될 확률은 $\frac{1}{n}$이고, 기준점이 p일 때 두 부분배열을 정렬하는데 걸리는 평균 기간은 $[A(p-1)+A(n-p)]$이고, 분할하는데 걸리는 시간은 $n-1$이므로, 평균적인 시간복잡도는 다음과 같이 된다.

$$A(n)=\sum_{p=1}^{n}\frac{1}{n}[A(p-1)+A(n-p)]+n-1$$

$$=\frac{2}{n}\sum_{p=1}^{n}A(p-1)+n-1$$

2001-03-19 알고리즘 강의 슬라이드 2 35

양변을 n으로 곱하면,

$$nA(n)=2\sum_{p=1}^{n}A(p-1)+n(n-1) \quad (1)$$

n대신 $n-1$을 대입하면,

$$(n-1)A(n-1)=2\sum_{p=1}^{n-1}A(p-1)+(n-1)(n-2) \quad (2)$$

(1)에서 (2)를 빼면,

$$nA(n)-(n-1)A(n-1)=2A(n-1)+2(n-1)$$

간단히 정리하면,

$$\frac{A(n)}{n+1}=\frac{A(n-1)}{n}+\frac{2(n-1)}{n(n+1)}$$

여기서,

$$a_n=\frac{A(n)}{n+1}$$

라고 하면, 다음과 같은 재현식을 얻을 수가 있다.

$$a_n=a_{n-1}+\frac{2(n-1)}{n(n+1)} \quad n>0\text{이면}$$

$$a_0=0$$

그러면,

$$a_n=a_{n-1}+\frac{2(n-1)}{n(n+1)} \quad a_{n-1}=a_{n-2}+\frac{2(n-2)}{(n-1)n} \quad \cdots \quad a_2=a_1+\tfrac{1}{3} \quad a_1=a_0+0$$

2001-03-19 알고리즘 강의 슬라이드 2 36

따라서, 해는

$$a_n = \sum_{i=1}^{n} \frac{2(i-1)}{i(i+1)}$$
$$= 2\left(\sum_{i=1}^{n} \frac{1}{i+1} - \sum_{i=1}^{n} \frac{1}{i(i+1)}\right)$$

여기에서 오른쪽 항은 무시해도 될 만큼 작으므로 무시한다.

$\ln n = \log_e n$이고,

$$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = \ln n$$

이므로, 해는 $a_n \approx 2 \ln n$. 따라서,

$$A(n) \approx (n+1)2 \ln n$$
$$= (n+1)2(\ln 2)(\lg n)$$
$$\approx 1.38(n+1) \lg n$$
$$\in \Theta(n \lg n)$$

2001-03-19 알고리즘 강의 슬라이드 2 37

행렬 곱셈(Matrix Multiplication)

- 단순한 행렬곱셈 알고리즘
  - ✓ 문제: $n \times n$ 크기의 행렬의 곱을 구하시오.
  - ✓ 입력: 양수 n, $n \times n$ 크기의 행렬 A와 B
  - ✓ 출력: 행렬 A와 B의 곱인 C
  - ✓ 알고리즘:

```
void matrixmult (int n, const number A[][], const number B[][],
                 number C[][]) {
    index i, j, k;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++) {
            C[i][j] = 0;
            for (k = 1; k <= n; k++)
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
}
```

2001-03-19 알고리즘 강의 슬라이드 2 38

행렬 곱셈

- 시간복잡도 분석 I:
  - ✓ 단위연산: 가장 안쪽의 루프에 있는 곱셈하는 연산
  - ✓ 입력크기: 행과 열의 수, n
  - ✓ 모든 경우 시간복잡도 분석: 총 곱셈의 횟수는 $T(n) = n \times n \times n = n^3 \in \Theta(n^3)$이다.
- 시간복잡도 분석 II:
  - ✓ 단위연산: 가장 안쪽의 루프에 있는 덧셈하는 연산
  - ✓ 입력크기: 행과 열의 수, n
  - ✓ 모든 경우 시간복잡도 분석: 총 덧셈의 횟수는 $T(n) = (n-1) \times n \times n = n^3 - n^2 \in \Theta(n^3)$이다.

2001-03-19 알고리즘 강의 슬라이드 2 39

2×2 행렬곱셈: 쉬트라쎈의 방법

- 문제: 두 2×2 행렬 A와 B의 곱(product) C,

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- 쉬트라쎈(Strassen)의 해:

$$C = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

여기서

$$\begin{aligned} m_1 &= (a_{11} + a_{22}) \times (b_{11} + b_{22}) \\ m_2 &= (a_{21} + a_{22}) \times b_{11} \\ m_3 &= a_{11} \times (b_{12} + b_{22}) \\ m_4 &= a_{22} \times (b_{21} + b_{11}) \\ m_5 &= (a_{11} + a_{12}) \times b_{22} \\ m_6 &= (a_{21} - a_{11}) \times (b_{11} + b_{12}) \\ m_7 &= (a_{12} - a_{22}) \times (b_{21} + b_{22}) \end{aligned}$$

- 시간복잡도 분석: 단순한 방법은 8번의 곱셈과 4번의 덧셈이 필요한 반면, 쉬트라쎈의 방법은 7번의 곱셈과 18번의 덧셈/뺄셈을 필요로 한다. 언뜻 봐서는 전혀 좋아지지 않았다! 그러나 행렬의 크기가 커지면 쉬트라쎈의 방법의 가치가 들어 난다.

2001-03-19 알고리즘 강의 슬라이드 2 40

$n \times n$ 행렬곱셈: 쉬트라쎈의 방법

- 문제: n이 2의 거듭제곱이고, 각 행렬을 4개의 부분행렬(submatrix)로 나눈다고 가정하자. 두 $n \times n$ 행렬 A와 B의 곱 C:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

- 쉬트라쎈(Strassen)의 해:

$$C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

여기서

$$\begin{aligned} M_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ M_2 &= (A_{21} + A_{22}) \times B_{11} \\ M_3 &= A_{11} \times (B_{12} + B_{22}) \\ M_4 &= A_{22} \times (B_{21} + B_{11}) \\ M_5 &= (A_{11} + A_{12}) \times B_{22} \\ M_6 &= (A_{21} - A_{11}) \times (B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \end{aligned}$$

2001-03-19 알고리즘 강의 슬라이드 2 41

쉬트라쎈의 알고리즘

- 문제: n이 2의 거듭제곱일 때, $n \times n$ 크기의 두 행렬의 곱을 구하시오.
- 입력: 정수 n, $n \times n$ 크기의 행렬 A와 B
- 출력: 행렬 A와 B의 곱인 C
- 알고리즘:

```
void strassen (int n, n*n_matrix A, n*n_matrix B, n*n_matrix& C) {
    if (n <= 분기점)
        단순한 알고리즘을 사용하여 C = A * B를 계산;
    else {
        A를 4개의 부분행렬 A11, A12, A21, A22로 분할;
        B를 4개의 부분행렬 B11, B12, B21, B22로 분할;
        쉬트라쎈의 방법을 사용하여 C = A * B를 계산;
        // 되부르는 호출의 예: strassen(n/2, A11+A12, B11+B22,M1)
    }
}
```

- 용어: 임계점(threshold)이란? 단순한 알고리즘보다 쉬트라쎈의 알고리즘을 사용하는 편이 더 좋을 것이라고 예상되는 지점.

2001-03-19 알고리즘 강의 슬라이드 2 42

분석

- 시간복잡도 분석 I
  - ✓ 단위연산: 곱셈하는 연산
  - ✓ 입력크기: 행과 열의 수, n
  - ✓ 모든 경우 시간복잡도 분석: 임계값을 1이라고 하자. (임계값은 차수에 전혀 영향을 미치지 않는다.)

재현식은 $T(n) = 7T(\frac{n}{2}) \quad n > 1$이고, $n = 2^k (k \geq 1)$

$T(1) = 1$

이 식을 전개해 보면,

$$T(n) = 7 \times 7 \times \cdots \times 7 \quad (k\text{번})$$
$$= 7^k$$
$$= 7^{\lg n}$$
$$= n^{\lg 7}$$
$$= n^{2.81}$$
$$\in \Theta(n^{2.81})$$

이 결과는 수학적귀납법에 의해서 증명이 가능하다. 증명을 해 보라. 사실 위의 재현방정식은 도사정리 3가지 중에서 1번을 이용하면 간단히 해를 구할 수 있다.

2001-03-19 알고리즘 강의 슬라이드 2 43

분석

- 시간복잡도 분석 II
  - ✓ 단위연산: 덧셈/뺄셈하는 연산
  - ✓ 입력크기: 행과 열의 수, n
  - ✓ 모든 경우 시간복잡도 분석: 위에서와 마찬가지로 임계값을 1이라고 하자. 재현식은

$$T(n) = 7T(\tfrac{n}{2}) + 18(\tfrac{n}{2})^2 \quad n > 1\text{이고, } n = 2^k (k \geq 1)$$
$$T(1) = 0$$

도사정리의 3가지 중에서 1번을 이용하면 간단히 해를 구할 수 있다.

$$T(n) = \Theta(n^{\lg_2 7}) = \Theta(n^{2.81})$$

2001-03-19 알고리즘 강의 슬라이드 2 44

사색

- 두 개의 행렬을 곱하기 위한 문제에 대해서 시간복잡도가 $\Theta(n^2)$이 되는 알고리즘을 만들어 낸 사람은 아무도 없다.
- 게다가 그러한 알고리즘을 만들 수 없다고 증명한 사람도 아무도 없다.

2001-03-19 알고리즘 강의 슬라이드 2 45

분할정복을 사용하지 말아야 하는 경우

- 크기가 n인 입력이 2개 이상의 조각으로 분할되며, 분할된 부분들의 크기가 거의 n에 가깝게 되는 경우 ⇒ 시간복잡도: 지수(exponential) 시간
- 크기가 n인 입력이 거의 n개의 조각으로 분할되며, 분할된 부분의 크기가 n/c인 경우. 여기서 c는 상수이다. ⇒ 시간복잡도: $\Theta(n^{\lg n})$

2001-03-19 알고리즘 강의 슬라이드 2 46