알고리즘분석 실습 자료

3주차

Photo by Piotr Guzik on Unsplash

실습 소요 시간 100분

2장 분할정복법 (divide-and-conquer)

실습프로그램

- ✓ 이분검색
- ✓ 합병정렬1
- ✓ 합병정렬2

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): 재귀적 방식

- 문제: 크기가 n인 정렬된 배열 S에 x가 있는지를 결정하라.
- 입력: 자연수 n, 비내림차순으로 정렬된 배열 $S[1..n]$, 찾고자 하는 항목 x
- 출력: 위치(location), x가 S의 어디에 있는지의 위치. 만약 x가 S에 없다면 0
- 설계전략:
  - ✓ x가 배열의 중간에 위치하고 있는 항목과 같으면, x 찾음. 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)

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);
...
```

[실습프로그램] 이분검색

```
def bs(data,item, low, high):



    이분 검색 구현



data=[1,3,5,6,7,9,10,14,17,19]
n=10
location=bs(data,17,0,n-1)
print(location)
```

[실습프로그램] 이분검색 알고리즘을 객체지향방법으로 구현

- Class data를 정의하여, binsearch라는 method를 구현한다.
- data명.binsearch(x)
- Binsearch
  - ✓ 입력: 특정값
  - ✓ 출력: 특정값이 있는 위치의 index. 데이터 내에 특정값이 존재하지 않을 경우 -1 return

[실습프로그램] 이분검색 알고리즘의 특정데이터를 찾을 때 까지의 데이터 비교 횟수 확인

- 데이터의 개수 : n
- k개의 문제를 생성하여 이분검색의 평균 데이터 비교 횟수를 확인한다.

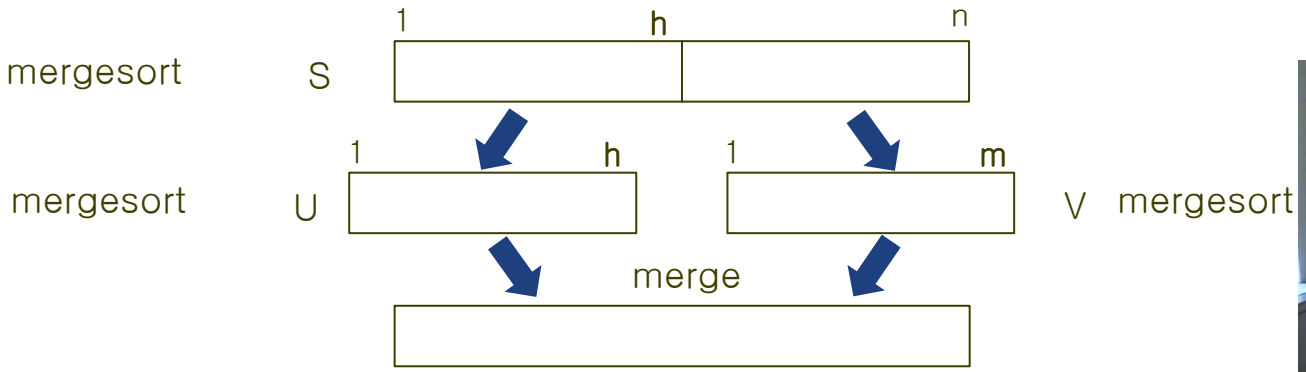
합병정렬(mergesort)

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

합병정렬

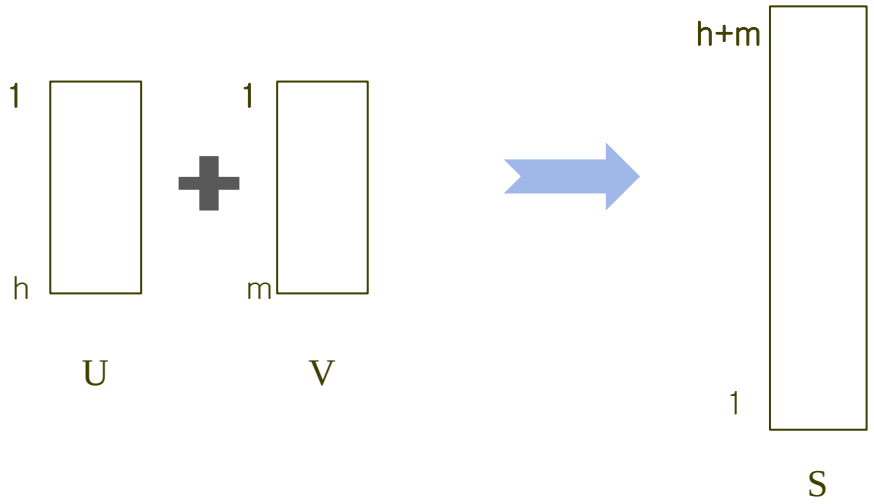
- 알고리즘:

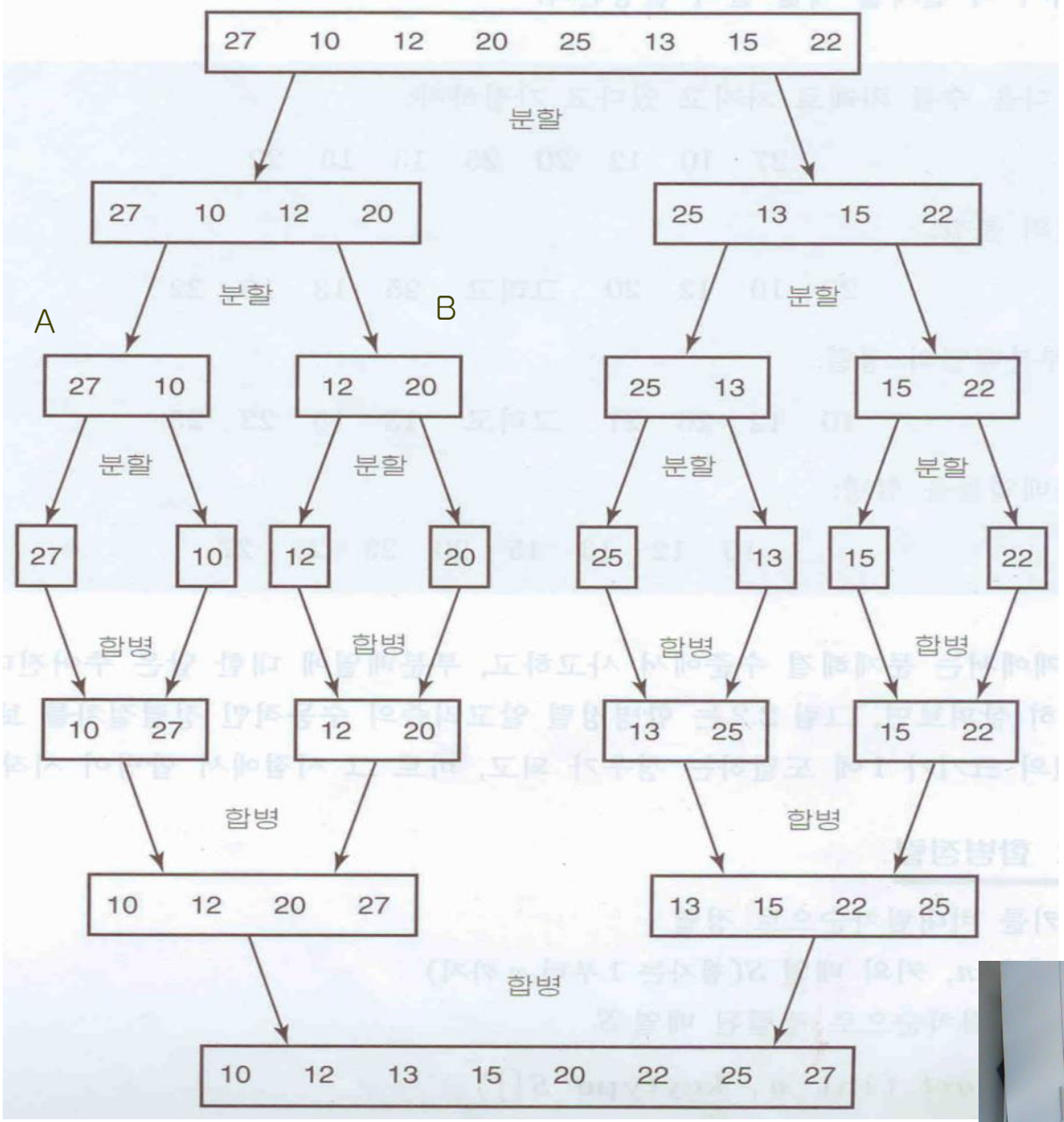
```
void mergesort (int n, keytype S[]) {
     const int h = n / 2, m = n - h;
     keytype U[1..h], V[1..m];

     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: 두 개의 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*]





- Fig 2.2 The steps done by a human when sorting with Mergesort

k	U	V	S (결과)
1	**10** 12 20 27	**13** 15 22 25	10
2	10 **12** 20 27	**13** 15 22 25	10 12
3	10 12 **20** 27	**13** 15 22 25	10 12 13
4	10 12 **20** 27	13 **15** 22 25	10 12 13 15
5	10 12 **20** 27	13 15 **22** 25	10 12 13 15 20
6	10 12 20 **27**	13 15 **22** 25	10 12 13 15 20 22
7	10 12 20 **27**	13 15 22 **25**	10 12 13 15 20 22 25
—	10 12 20 27	13 15 22 25	10 12 13 15 20 22 25 27 ← 최종값

* 비교되는 아이템은 진하게 표시되어 있다.

- 표 2.1 2개의 배열 U와 V를 하나의 배열 S로 합병하는 예

```
void  merge(int h, int m, const keytype U[], const keytype
            V[], 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];
}
```

- [m:n] : 인덱스가 m인 데이터부터 인덱스가 n-1 인 데이터까지를 표현
- m이 없을 경우는 처음부터, n이 없을 경우는 끝까지

```
a=[4,1,5,9,10]
h=3
leftHalf=a[:h]
rightHalf=a[h:]
print(leftHalf)
print(rightHalf)
```

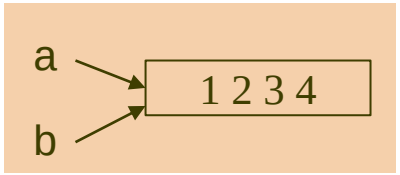
```
[4, 1, 5]
[9, 10]
>>>
```

```
a=['a','b','c','d','e']
print(a[1:3])
print(a[:3])
print(a[3:])
```

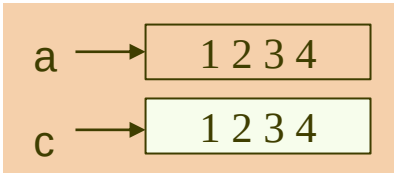
```
['b', 'c']
['a', 'b', 'c']
['d', 'e']
>>>
```

리스트는 mutable(값을 바꿀 수 있다)

- b=a 는 리스트 a를 b라고도 나타낼 수 있다.



- c=a[:] 는 리스트 a를 리스트 c에 복사한다.




```
a=[1,2,3,4]
b=a
c=a[:]
print(a,b,c)
a.append(5)
print(a,b,c)
```

```
          a               b               c
[1, 2, 3, 4] [1, 2, 3, 4] [1, 2, 3, 4]
[1, 2, 3, 4, 5] [1, 2, 3, 4, 5] [1, 2, 3, 4]
```

변한다 변하지 않는다

[실습프로그램] 합병정렬

```
def mergeSort(n, s):

    구현
```

```
def merge(h,m,u,v,s):

    구현


s=[3,5,2,9,10,14,4,8]
mergeSort(8,s)
print(s)
```

[실습프로그램] 합병정렬

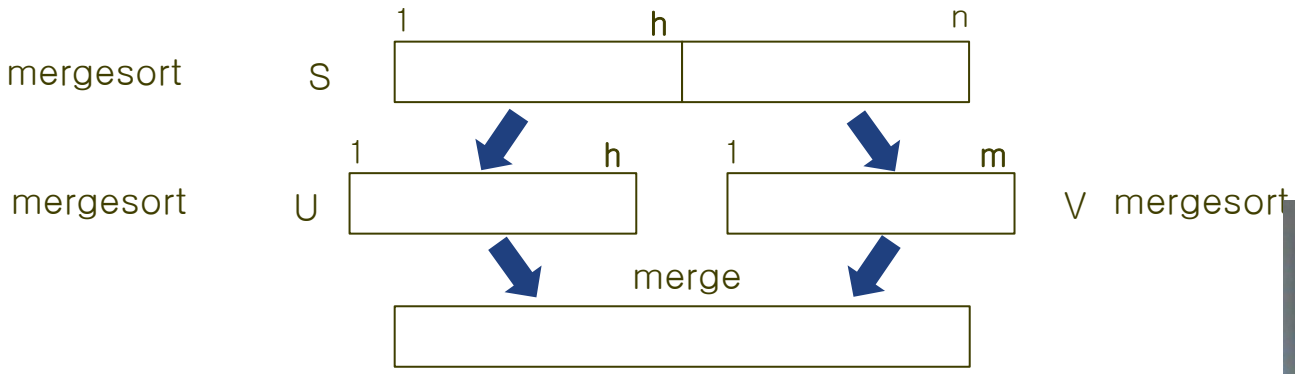
- 합병정렬을 recursion이 없는 방식으로 구현

공간복잡도 분석

- 추가적인 저장장소를 사용하지 않고 정렬하는 알고리즘

  - 제자리정렬(in-place sort) 알고리즘

- 합병정렬 알고리즘은 제자리정렬 알고리즘이 아님. 입력배열 S이외에 U와 V를 추가로 만들어서 사용
- 하단의 재귀호출이 종료될 때까지 상위의 재귀호출이 생성하는 공간이 유지되어야 함.



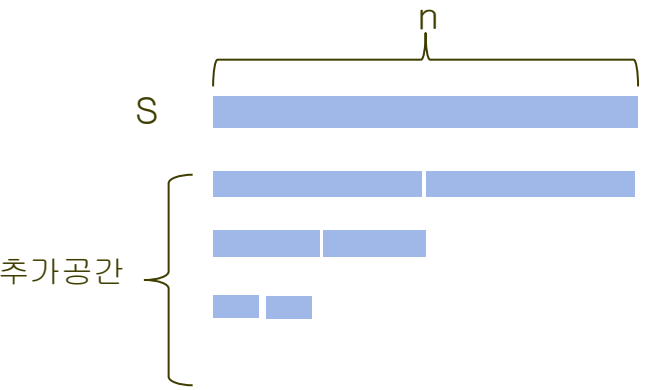
공간복잡도 분석

- 추가적인 저장장소: mergesort를 재귀호출할 때마다 크기가 S의 반이 되는 U와 V가 추가적으로 필요. (Merge 는 추가적인 저장장소 불필요).

  처음 S의 크기가 n이면, 추가적으로 필요한 U와 V의 저장장소 크기의 합은 n이 된다. 다음 재귀 호출에는 $n/2$의 추가적으로 필요한 총 저장장소의 크기는

$$n+\frac{n}{2}+\frac{n}{4}+\ldots=2n \qquad 2n\in\Theta(n)$$

- 추가적으로 필요한 저장장소가 n이 되도록, 즉, 공간복잡도가 n이 되도록 알고리즘을 향상시킬 수 있다(다음 절의 알고리즘). 그러나 합병정렬 알고리즘이 제자리정렬 알고리즘이 될 수는 없다.




```
void mergesort (int n, keytype S[]) {
                                  ……….
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);
   }
}
```

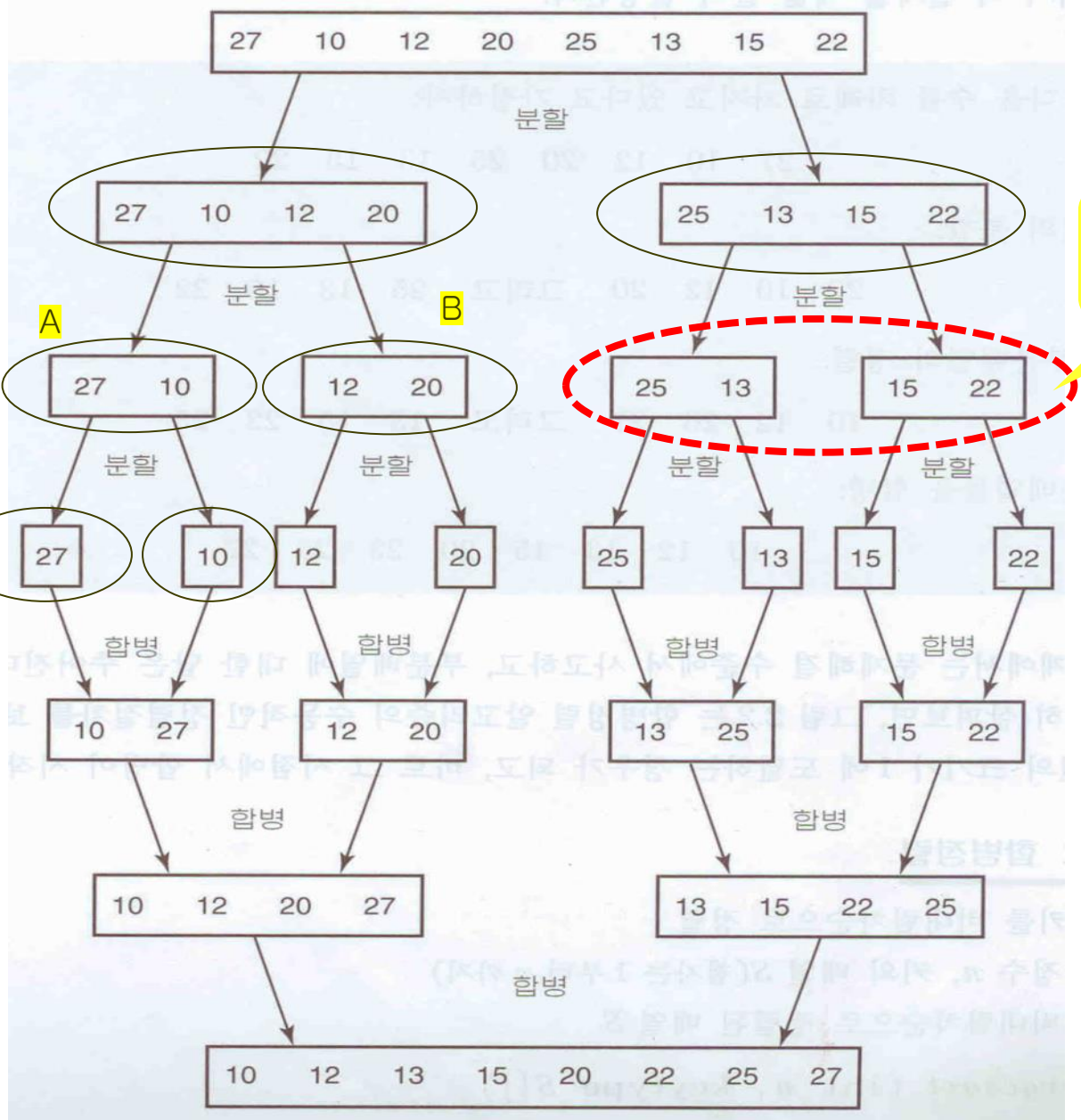
추가 공간

4+4

2+2

1+1

총추가 공간 14



-A,B 사용이 끝난 후 필요한 공간
-A,B를 재사용

- 그림 2.2 합병정렬 알고리즘의 정렬절차

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);
             merge2(low, mid, high);
         }
}
   …
   mergesort2(1, n);
   ...
```

공간복잡도가 향상된 알고리즘

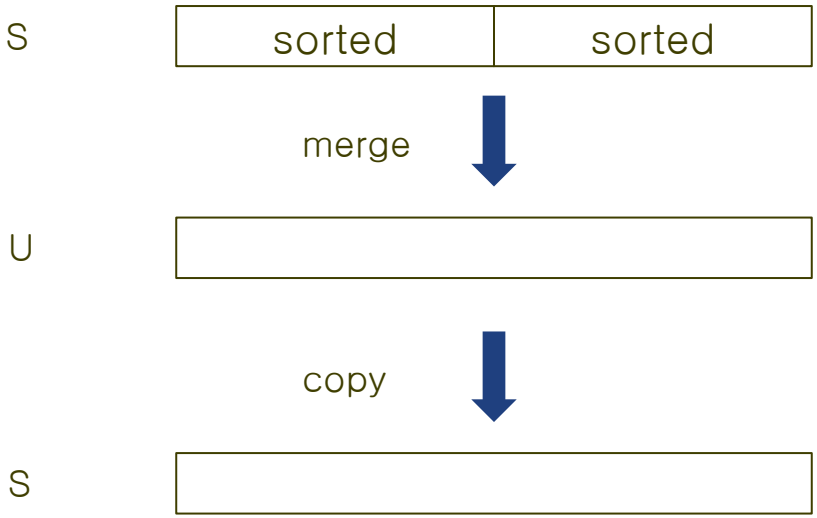
- 합병(merge2)
  - ✓ 문제: 두 개의 정렬된 배열을 하나의 정렬된 배열로 합병하시오.
  - ✓ 입력: (1) 첨자 low, mid, high,

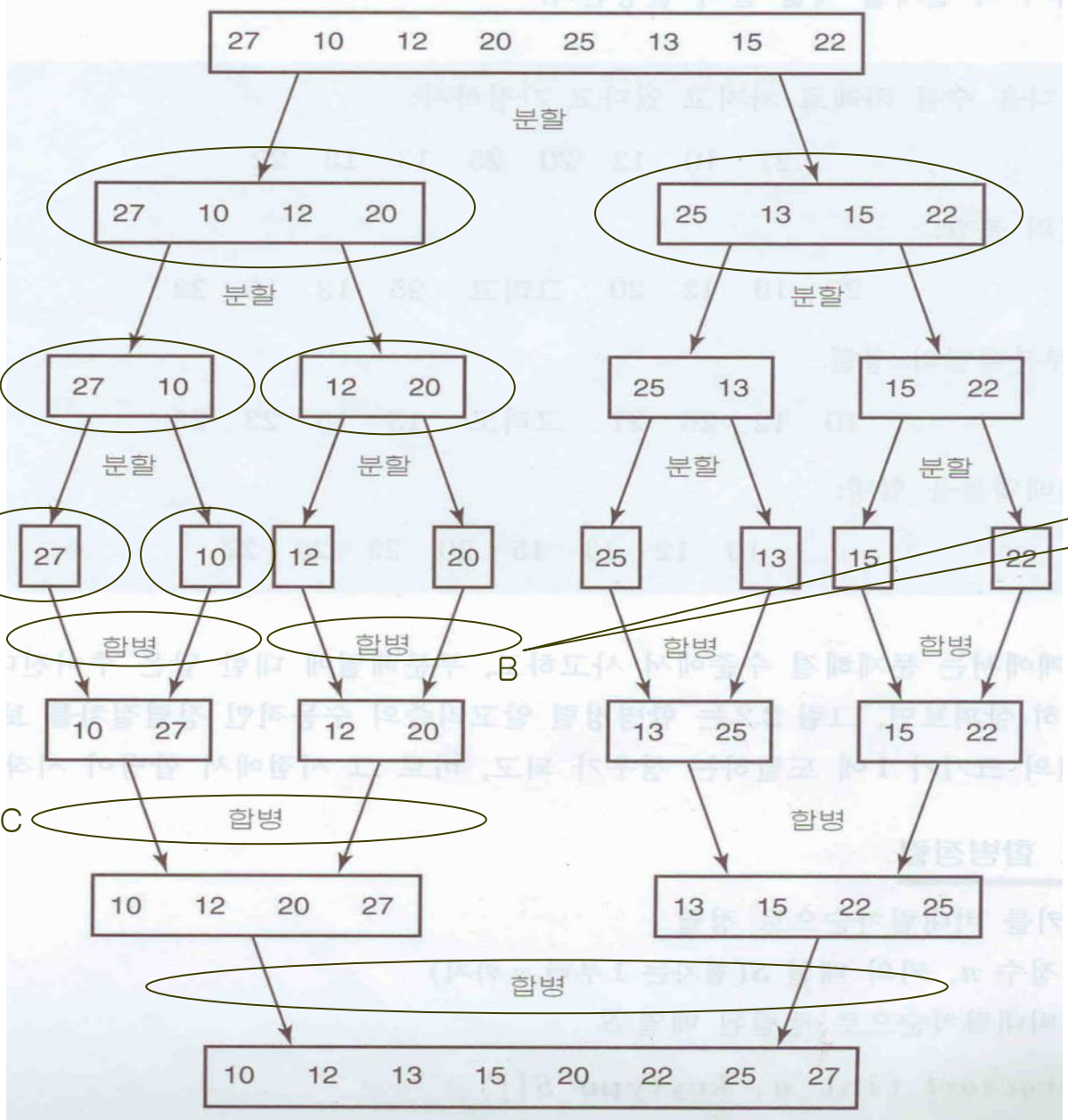
    (2) 부분 배열S[low..high], 여기서 S[low..mid]와 S[mid+1..high]는 이미 각각 정렬이 완료되어 있음.
  - ✓ 출력: 정렬이 완료된 부분배열 S[1ow..high]

- 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];
}
```

merge2





추가 공간

복사는 나중에 수행

복사는 나중에 수행

2

4 (A+B에서 사용된 공간 포함)

8(C를 포함)

총추가 공간 8

- mergesort2의 절차. Additional space is *n*.

[실습프로그램] 합병정렬2

```
def mergeSort2(s, low, high):

      구현

def merge2(s, low, mid, high):

      구현


s=[3,5,2,9,10,14,4,8]
mergeSort2(s,0,7)
print(s)
```

강의가 곧 시작됩니다.