파이썬으로 배우는 알고리즘 기초

Chap 2. 분할정복

2.1-2.2

이분 검색과 합병 정렬

2.1 이분 검색

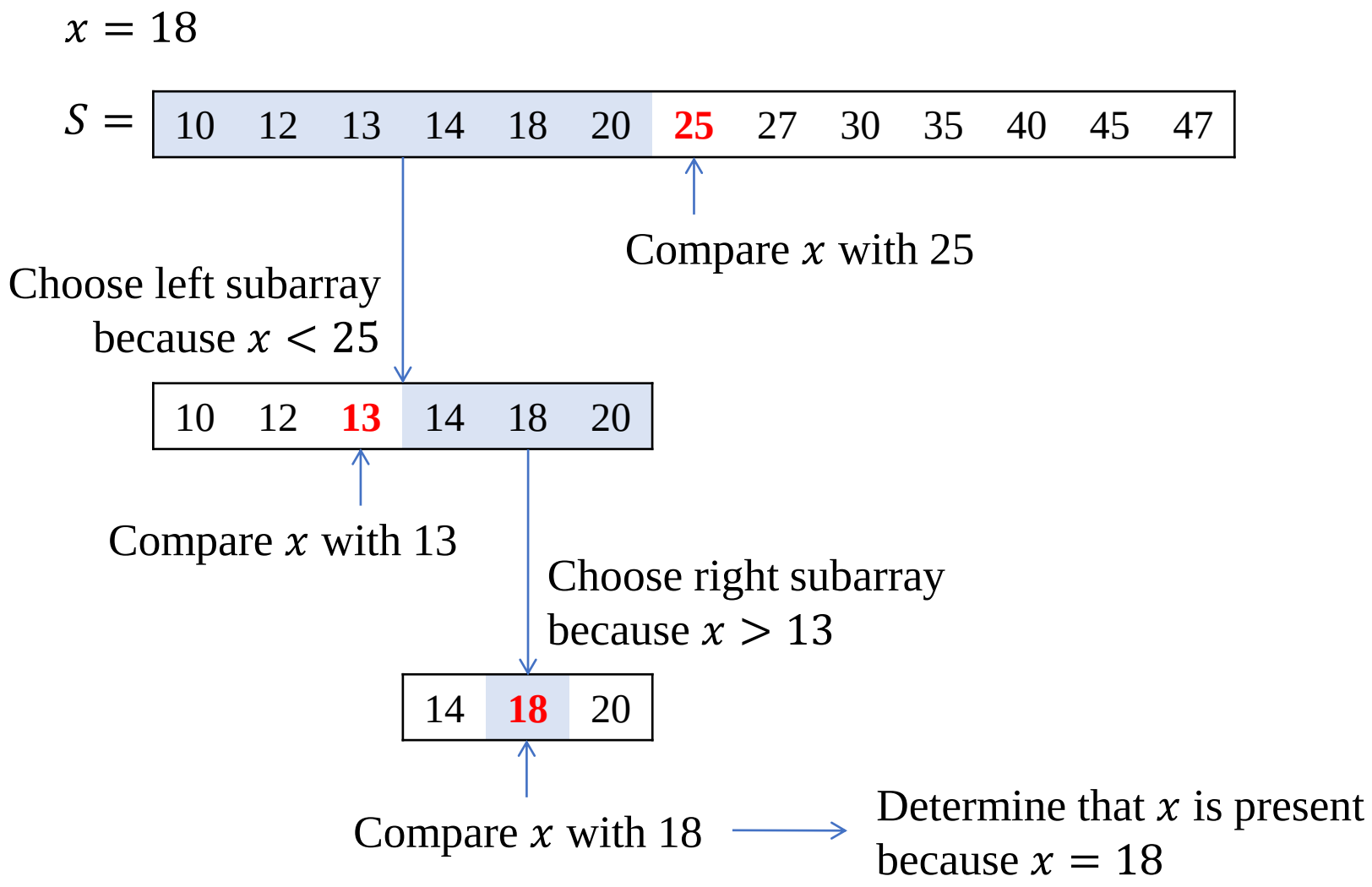
- **이분 검색 문제**
  - 정렬되지 않은 리스트에서 주어진 키가 존재하는가?
    - 순차 탐색
  - 정렬된 리스트에서 주어진 키가 존재하는가?
    - 이분 검색

  - Algorithm 1.5: **반복적인 방법**으로 이분 검색
  - Algorithm 2.1: **재귀적인 방법**으로 이분 검색

2.1 이분 검색

- 이분 검색: 분할정복(Divide-and-Conquer)
  - 문제: 정렬된 리스트 S에 어떤 키 x가 존재하는가?
  - 해답: 존재하면 S에서 x의 위치, 아니면 0을 리턴
  - 알고리즘: 분할정복
    - S의 정가운데 원소와 x를 비교하여 같으면 해당 위치를 리턴, 아니면:
    - [Divide] 정가운데 원소를 기준으로 S를 두 개의 리스트로 분할
    - [Conquer] x가 정가운데 원소보다 크면 오른쪽, 작으면 왼쪽을 재귀 호출
    - [Obtain] 선택한 리스트에서 얻은 답을 리턴

2.1 이분 검색



Algorithm 2.1: Binary Search (Recursive)

```
def location (S, low, high):
    if (low > high):
        return 0
    else:
        mid = (low + high) // 2
        if (x == S[mid]):
            return mid
        elif (x < S[mid]):
            return location(S, low, mid - 1)
        else:
            return location(S, mid + 1, high)
```

2.1 이분 검색


```
S = [-1, 10, 12, 13, 14, 18, 20, 25, 27, 30, 35, 40, 45]
x = 18
loc = location(S, 1, len(S) - 1)
print('S =', S)
print('x =', x)
print('loc = ', loc)
```

2.2 합병 정렬

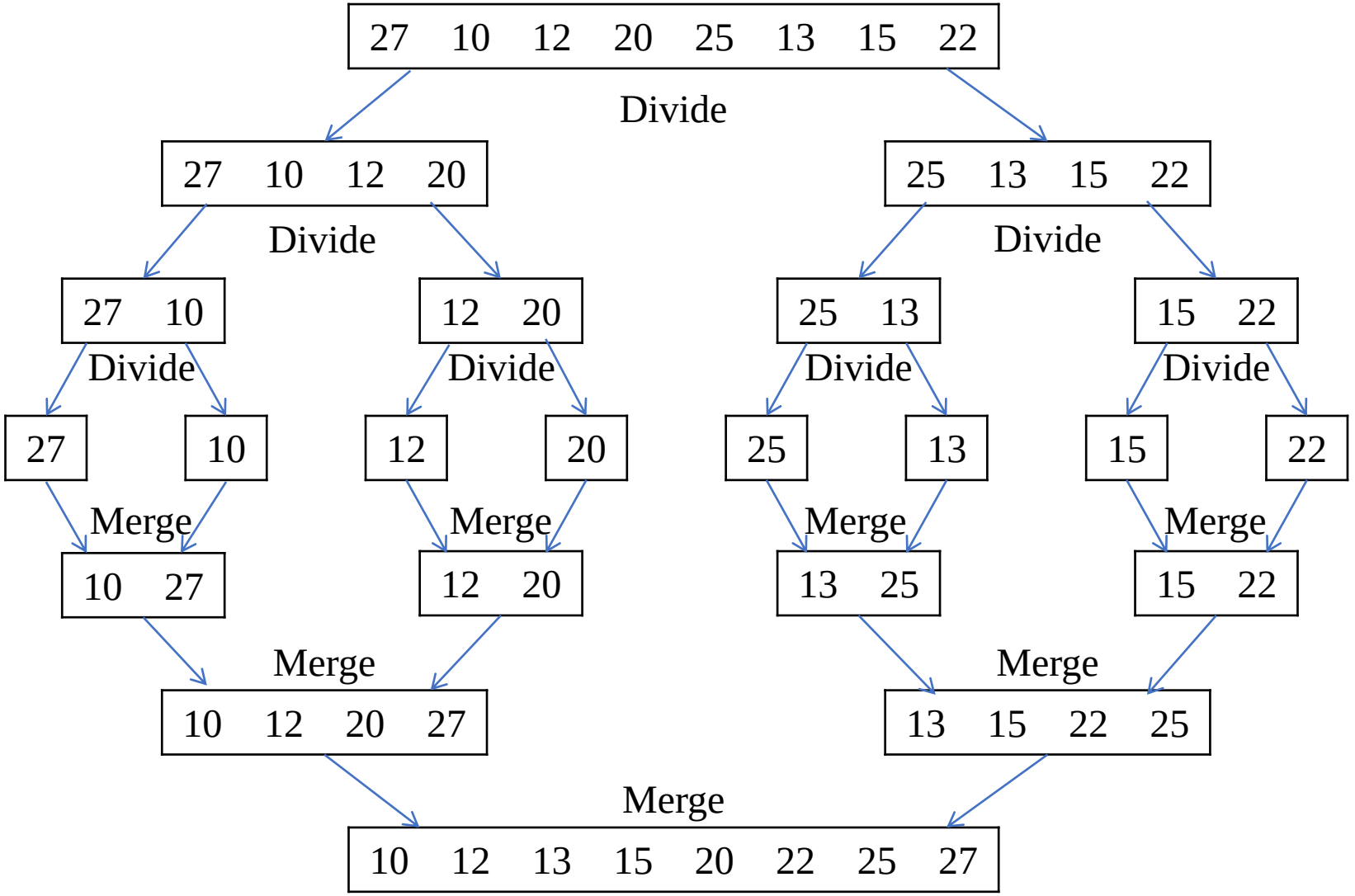
- **리스트(배열)의 정렬 문제**
  - **문제: 정렬되지 않는 리스트 S를 오름차순으로 정렬하시오.**
  - **해답: 정렬된 리스트 S'을 리턴**

  - Algorithm 1.3: **교환 정렬**
  - Algorithm 2.2: **합병 정렬**

2.2 합병 정렬

- **합병 정렬: 분할정복(Divide-and-Conquer)**
  - **[Divide]**
    - **원소가 n개인 S를 $n/2$개의 원소를 가진 두 개의 리스트로 분할**
  - **[Conquer]**
    - **왼쪽의 리스트와 오른쪽의 리스트를 각각 재귀적으로 합병 정렬**
  - **[Combine]**
    - **각각 정렬된 두 개의 리스트를 정렬된 하나의 리스트로 합병하여 리턴**

2.2 합병 정렬



2.2 합병 정렬

Algorithm 2.2: Merge Sort

```
def mergesort (S):
    n = len(S)
    if (n <= 1):
        return S
    else:
        mid = n // 2
        U = mergesort(S[0 : mid])
        V = mergesort(S[mid : n])
        return merge(U, V)
```

Algorithm 2.3: Merge

```
def merge(U, V):
    S = []
    i = j = 0
    while (i < len(U) and j < len(V)):
        if (U[i] < V[j]):
            S.append(U[i])
            i += 1
        else:
            S.append(V[j])
            j += 1
    if (i < len(U)):
        S += U[i : len(U)]
    else:
        S += V[j : len(V)]
    return S
```

2.2 합병 정렬


```
S = [27, 10, 12, 20, 25, 13, 15, 22]
print('Before: ', S)
X = mergesort(S)
print(' After: ', X)
```

```
U = [27] V = [10]
U = [12] V = [20]
U = [10, 27] V = [12, 20]
U = [25] V = [13]
U = [15] V = [22]
U = [13, 25] V = [15, 22]
U = [10, 12, 20, 27] V = [13, 15, 22, 25]
```

2.2 합병 정렬

- Algorithm 2.2/2.3의 문제점
  - 입력 리스트 S 이외에 리스트 U와 V를 추가적으로 사용
  - **메모리 사용의 비효율성**: 더 효율적인 방법은 없을까?

  - 추가적으로 만들어지는 리스트 원소의 총 수
    - `mergesort()`를 호출할 때마다 U와 V를 새로 생성함
    - 첫번째 재귀 호출시 원소의 개수: U가 $n/2$개, V가 $n/2$개 (대략 n개)
    - 두번째 재귀 호출시: U가 $n/4$개, V가 $n/4$개 (대략 $n/2$개)
    - .....
    - 전체 재귀 호출시 원소의 개수: $n + \frac{n}{2} + \frac{n}{4} + \cdots = 2n$ (**대략 $2n$개 정도**)

Algorithm 2.4: Merge Sort 2 (Enhanced Merge Sort)

```
def mergesort2 (S, low, high):
    if (low < high):
        mid = (low + high) // 2
        mergesort2(S, low, mid)
        mergesort2(S, mid + 1, high)
        merge2(S, low, mid, high)
```

Algorithm 2.5: Merge2 (Enhanced Merge)

```
def merge2 (S, low, mid, high):
    U = []
    i = low
    j = mid + 1
    while (i <= mid and j <= high):
        if (S[i] < S[j]):
            U.append(S[i])
            i += 1
        else:
            U.append(S[j])
            j += 1
    if (i <= mid):
        U += S[i : mid + 1]
    else:
        U += S[j : high + 1]
    for k in range(low, high + 1):
        S[k] = U[k - low]
```

- 추가적으로 만들어지는 원소의 수를 대략 n개 정도로 절약

2.2 합병 정렬


```
S = [27, 10, 12, 20, 25, 13, 15, 22]
print('Before: ', S)
mergesort2(S, 0, len(S) - 1)
print(' After: ', S)
```

```
[27, 10, 12, 20, 25, 13, 15, 22]
[10, 27, 12, 20, 25, 13, 15, 22]
[10, 27, 12, 20, 25, 13, 15, 22]
[10, 12, 20, 27, 25, 13, 15, 22]
[10, 12, 20, 27, 13, 25, 15, 22]
[10, 12, 20, 27, 13, 25, 15, 22]
[10, 12, 20, 27, 13, 15, 22, 25]
```

주니온TV@Youtube

자세히 보면 유익한 코딩 채널

https://bit.ly/2JXXGqz

- 여러분의 **구독**과 **좋아요**는 강의제작에 큰 힘이 됩니다.
- 강의자료 및 소스코드: **구글 드라이브**에서 다운로드
  (다운로드 주소는 영상 하단 설명란 참고)

  https://bit.ly/3fN0q8t