파이썬으로 배우는 알고리즘 기초

Chap 1. 알고리즘: 효율, 분석, 차수

1.2 알고리즘의 효율성

1.2 알고리즘의 효율성

- 알고리즘의 효율성
  - 알고리즘의 성능: 시간과 공간 사용량의 효율성
  - 알고리즘의 성능은 컴퓨터의 실행 속도나 메모리의 가격에 무관

- 순차 탐색 .vs. 이분 검색
  - 입력 리스트의 조건에 따른 탐색 알고리즘의 선택
    - 정렬되지 않은 리스트에서 키 찾기: 순차 탐색
    - 정렬된 리스트에서 키 찾기: 이분 검색

1.2 알고리즘의 효율성

이분 검색

- 주어진 리스트 S와 키 x에 대해서,
- 먼저 x를 리스트의 중앙에 위치한 원소와 비교
- 만약 같으면, 찾았으므로 알고리즘을 종료
- 만약 x가 그 원소보다 작으면 x는 왼쪽에 있을 것이므로
  - 왼쪽 리스트에 대해서 이진 탐색 실행 (재귀 호출)
- 만약 x가 그 원소보다 크면 x는 오른쪽에 있을 것이므로
  - 오른쪽 리스트에 대해서 이진 탐색 실행 (재귀 호출)
- 더 이상 찾을 리스트가 없으면 알고리즘을 종료

Algorithm 1.5: Binary Search (Iterative)

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

1.2 알고리즘의 효율성


```
S = [-1, 5, 7, 8, 10, 11, 13]
x = 2
# x = 7
# x = 13
location = binsearch(len(S) - 1, S, x)
print('S =', S)
print('x =', x)
print('location =', location)
```

1.2 알고리즘의 효율성

- 순차 탐색과 이분 검색 알고리즘의 효율성 비교
  - 순차 탐색: 크기가 n인 리스트에서 n번의 비교를 수행
  - 이분 검색: 크기가 n인 리스트에서 $\lg n + 1$번의 비교를 수행

리스트의 크기	순차 탐색의 비교 횟수	이분 검색의 비교 횟수
128	128	8
1,024	1,024	11
1,048,576	1,048,576	21
4,294,987,296	4,294,987,296	33

1.2 알고리즘의 효율성

- **피보나치 수열의 n번째 항 구하기**
  - **피보나치 수열**: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
  - **피보나치 수열의 (재귀적) 정의**
    - $f_0 = 0, f_1 = 1$
    - $f_n = f_{n-1} + f_{n-2}\ , (n \geq 2)$
  - **문제: 피보나치 수열의 n번째 항을 구하시오**
  - **알고리즘: 재귀적 정의를 그대로 구현하면 됨**

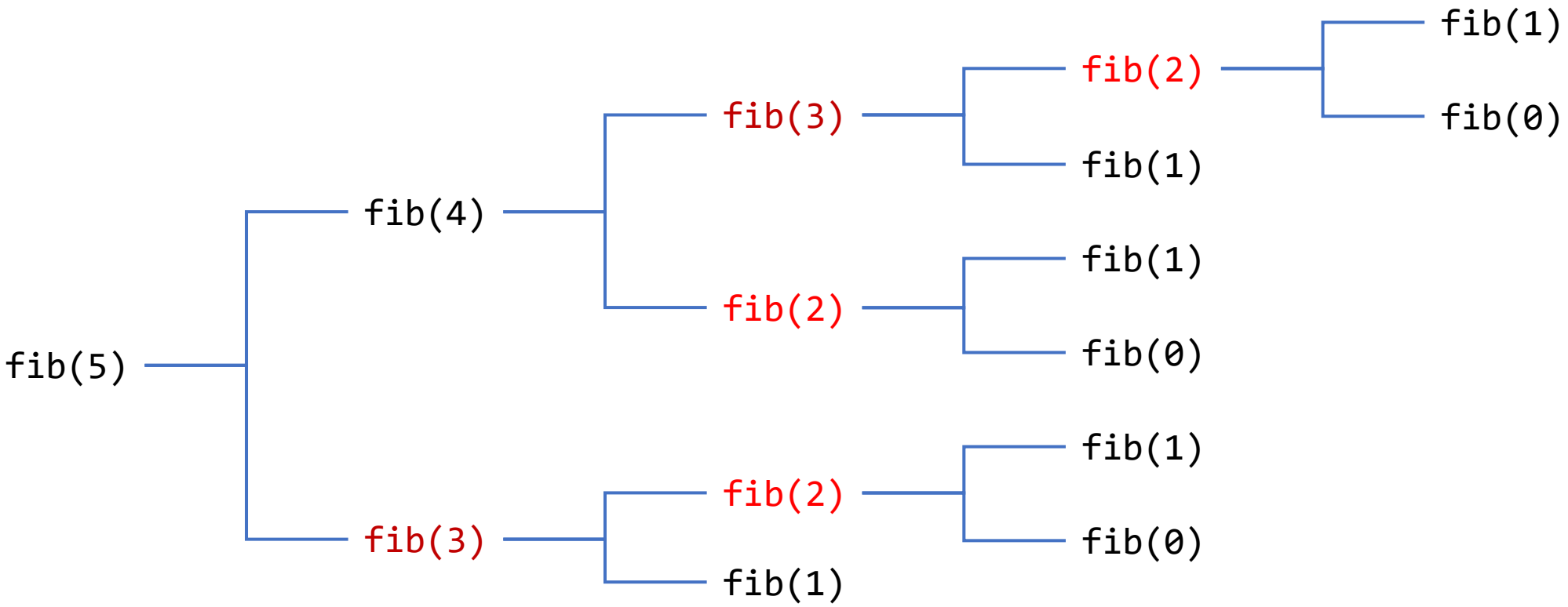
Algorithm 1.6: Finding the n-th Fibonacci Term (Recursive)

```
def fib (n):
    if (n <= 1):
        return n
    else:
        return fib(n - 1) + fib(n - 2)
```

```
for i in range(11):
    print(fib(i), end=" ")
```

1.2 알고리즘의 효율성

- Algorithm 1.6의 효율성
  - 재귀적 정의 이용: 작성하기도 쉽고 이해하기도 쉬움
  - 그러나 너무 비효율적이다. 왜 그럴까?



1.2 알고리즘의 효율성

- Algorithm 1.6의 **비효율성을 개선하려면?**
  - **같은 값을 중복해서 재귀적으로 계산하지 않도록 해야 함**
  - **아직 계산하지 않은 피보나치 항의 값은 계산을 해야 함**
  - **이미 계산한 피보나치 항의 값은 리스트에 저장**
  - **이미 계산되어 저장된 피보나치 항은 필요할 때 꺼내쓰면 된다.**

Algorithm 1.7: Finding the n-th Fibonacci Term (Iterative)

```
def fib2 (n):
    f = [0] * (n + 1)
    if (n > 0):
        f[1] = 1
        for i in range(2, n + 1):
            f[i] = f[i - 1] + f[i - 2]
    return f[n]
```

```
for i in range(11):
    print(fib2(i), end=" ")
```

1.2 알고리즘의 효율성

- **연습문제** 1.2.1:
  - Algorithm 1.7**에서 리스트** f **를 사용하지 않아도 되는가?**
  - **만약, 그렇다면** f **를 사용하지 않고 반복문으로 피보나치 항을 구하시오.**

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