알고리즘분석 실습 자료

13주차

Photo by Piotr Guzik on Unsplash

실습 소요 시간 100분

7장. 계산복잡도의 소개: 정렬 문제

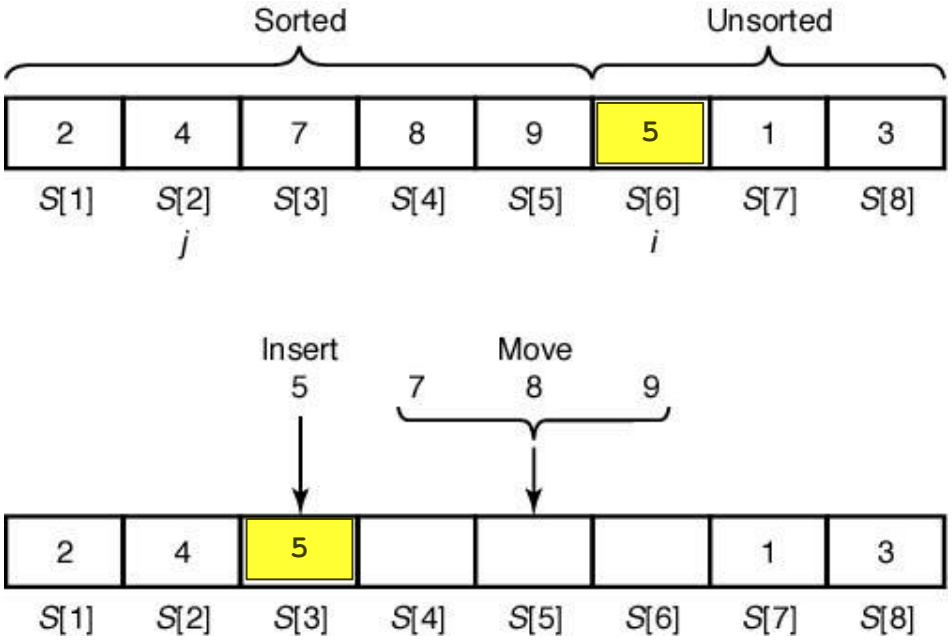
실습프로그램

- ✓ 삽입정렬
- ✓ 선택정렬
- ✓ 교환정렬
- ✓ 거품정렬
- ✓ 힙정렬
  - 방법2를 이용하여 makeHeap 구현 - 힙정렬완성
  - 방법1을 이용하여 makeHeap 구현

- ✓ 참고
  - 합병정렬
  - 빠른정렬

삽입정렬 알고리즘 (Insertion Sort)

- 이미 정렬된 배열에 항목을 끼워 넣음으로써 정렬하는 알고리즘
- **알고리즘: 삽입정렬**
  - ✓ 문제: 비내림차순으로 n개의 키를 정렬
  - ✓ 입력: 양의 정수 n; 키의 배열 S[1..n]
  - ✓ 출력: 비내림차순으로 정렬된 키의 배열 S[1..n]



삽입정렬 알고리즘

```
void insertionsort(int n, keytype S[]){
        index i,j;
        keytype x;

        for(i=2; i<=n; i++){
          x = S[i];
          j = i - 1;
          while(j>0 && S[j]>x){
            S[j+1] = S[j];
            j--;
          }
          S[j+1] = x;
        }
      }
```

[실습프로그램] 삽입정렬 알고리즘

```
s=[3,2,5,7,1,9,4,6,8]
n = len(s)

    삽입정렬 구현



print(s)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[실습프로그램 (optional)]] 객체지향방법을 이용한 삽입정렬

```
class List():

    def __init__(self, s):
        self.data = s[:]
        self.n = len(s)

    def insertion_sort(self):



       구현



s=[3,2,5,7,1,9,4,6,8]

a = List(s)
print(a.insertion_sort())
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

선택정렬 알고리즘(selection sort)

- 문제: 비내림차순으로 n개의 키를 정렬
- 입력: 양의 정수 n; 키의 배열 S[1..n]
- 출력: 비내림차순으로 정렬된 키의 배열 S[1..n]

```
void selectionsort(int n, keytype S[]){
          index i, j, smallest;

          for(i=1; i<=n-1; i++){
            smallest = i;
            for(j=i+1; j<=n; j++)
              if (S[j]<S[smallest])
                smallest = j;
            exchange S[i] and S[smallest];
          }
        }
```

[실습프로그램] 선택정렬 알고리즘(selection sort)

```
s=[3,2,5,7,1,9,4,6,8]
n = len(s)



    구현



print(s)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[실습프로그램(optional)] 객체지향방법으로 선택정렬을 구현하시오.

[실습프로그램] 교환정렬 알고리즘(Exchange Sort)

```
s=[3,2,5,7,1,9,4,6,8]
n = len(s)



     구현



print(s)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[실습프로그램 (optional)] 객체지향방법으로 교환정렬을 구현하시오.

[실습프로그램] 거품정렬 (Bubble Sort)

```
s=[3,2,5,7,1,9,4,6,8]
n = len(s)



     구현



print(s)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[실습프로그램 (optional)] 객체지향방법으로 거품정렬을 구현하시오.

[실습프로그램] 합병정렬 알고리즘

```
def mergeSort(n, s):
    h=int(n/2)
    m=n-h
    u=h*[0]
    v=m*[0]

    if(n>1):
        leftHalf=s[:h]
        rightHalf=s[h:]
        mergeSort(h,leftHalf)
# h=len(leftHalf)
        mergeSort(m,rightHalf)
        merge(h,m,leftHalf,rightHalf,s)

def merge(h,m,u,v,s):
    i=j=k=0
    while(i<=h-1 and j<=m-1):
        if(u[i]<v[j]):
           s[k]=u[i]
           i+=1
        else:
           s[k]=v[j]
           j+=1
        k+=1
    if(i>h-1):
           for ii in range (j,m):
                s[k+ii-j]=v[ii]
    else:
           for ii in range (i,h):
              s[k+ii-i]=u[ii]

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

```
[2, 3, 4, 5, 8, 9, 10, 14]
```

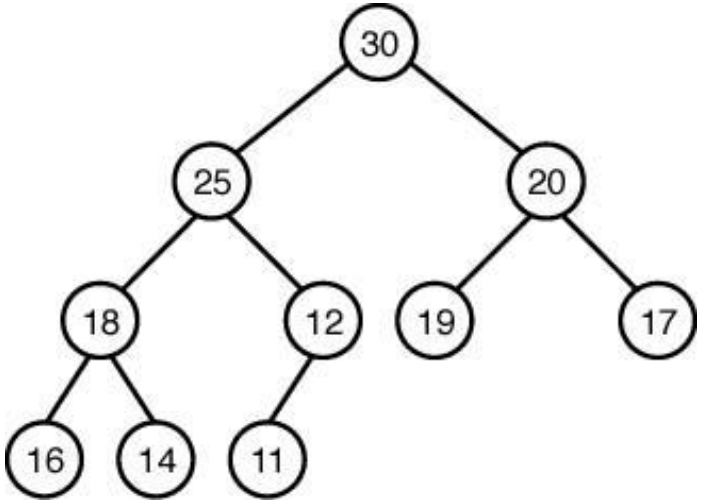

[실습프로그램] 빠른정렬 알고리즘

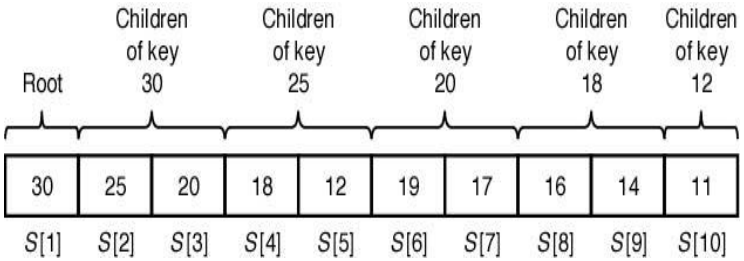
```
def quickSort(s,low, high):
    pivotPoint=-1
    if(high>low):
        pivotPoint= partition(s,low,high)
        quickSort(s,low, pivotPoint-1)
        quickSort(s,pivotPoint+1,high)


def partition(s,low,high):
    pivotItem=s[low]
    j=low
    for i in range(low+1,high+1):
        if(s[i]<pivotItem):
            j+=1
            temp=s[i]
            s[i]=s[j];
            s[j]=temp
    pivotPoint=j
    temp=s[low]
    s[low]=s[pivotPoint]
    s[pivotPoint]=temp
    return pivotPoint


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

```
[2, 3, 4, 5, 8, 9, 10, 14]
```

힙(heap)

- 힙의 성질(heap property): 어떤 마디에 저장된 값은 그 마디의 자식마디에 저장된 값보다 크거나 같다. – max heap
- 힙(heap): 힙의 성질을 만족하는 실질적인 완전이진트리



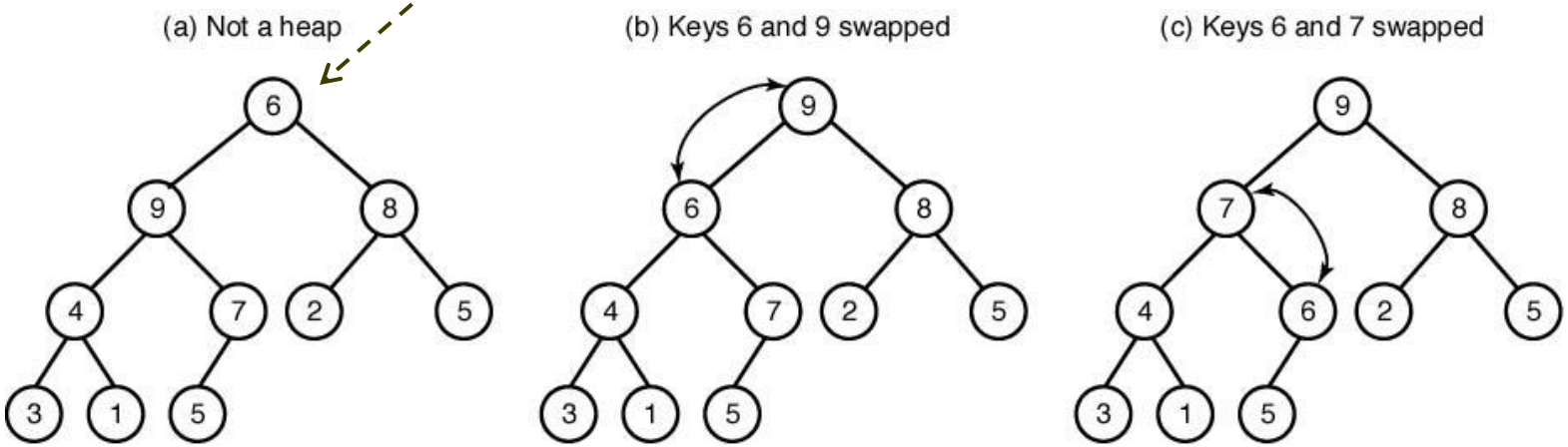


힙의 자료구조(배열)

Siftdown

sift: 채로 치다

- 힙 성질을 만족하도록 재구성 방법
  - ✓ 루트에 있는 키가 힙성질을 만족하지 않음.



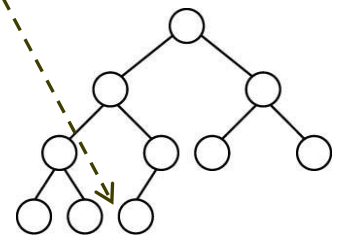
- ✓ 교체하는 child node를 결정하기 위해 2회의 비교 필요

- 힙성질을 만족하도록 조정

```
void siftdown(heap& H){
        node parent, largerchild;

        parent = root of H;
        largerchild = parent's child containing larger key;

        while(key at parent is smaller than key at largerchild){
            exchange key at parent and key at largerchild;
            parent = largerchild;
            largerchild = parent's child containing larger key;
          }
}
```

- 루트에서 키를 추출하고 힙 성질을 회복하는 의사코드

```
keytype root(heap& H){

     keytype keyout;

        keyout = key at the root;
        move the key at the bottom node to the root;
        delete the bottom node;
        siftdown(H);
        return keyout;
     }
```



힙정렬

- 힙 정렬 아이디어

1. n개의 키를 이용하여 힙을 구성한다.
2. 루트에 있는 제일 큰 값을 제거한다. → 힙 재구성
3. step 2를 $n-1$번 반복한다.

힙정렬

```
void removekeys(int n, heap H, keytype S[]){
  index i;
  for(i=n; i>=1; i--)
    S[i] = root(H);
}
```

```
void makeheap(int n, heap& H){
  index i;
  heap Hsub;
  for(i=d-1; i>=0; i--)
    for(all subtree Hsub whose roots have depth i)
      siftdown(Hsub);
}
```

d=H의 높이, i는 depth의 index

```
void heapsort(int n, heap H, keytype S[]){
  makeheap(n,H);
  removekeys(n,H,S);
}
```


heap 정렬

```
struct heap{
  keytype S[1..n];
  int heapsize;};

void siftdown(heap& H, index i){
  index parent, largerchild;
  keytype siftkey;
  bool spotfound;

  siftkey = H.S[i];
  parent = i;
  spotfound = false;
  while( 2*parent ≤ H.heapsize && !spotfound){
    if(2*parent < H.heapsize &&
          H.S[2*parent] < H.S[2*parent+1])
       largerchild = 2*parent + 1;
    else
       largerchild = 2*parent;
    if(siftkey < H.S[largerchild]){
       H.S[parent] = H.S[largerchild];
       parent = largerchild;
    }
    else
       spotfound = true;
  }
  H.S[parent] =siftkey;
 }
```

```
keytype root(heap& H) {
   keytype keyout;
   keyout = H.S[1];
   H.S[1] = H.S[heapsize];
   H.heapsize = H.heapsize -1;
   siftdown(H,1);
   return keyout;
   }

void removekeys(int n, heap& H, keytype S[]){
    index i;
    for (i=n; i≥1; i--)
       S[i] = root(H);
     }

void makeheap(int n, heap& H){
   index i;

   H.heapsize=n;
   for(i=⌊n/2⌋; i ≥ 1, i--)
      siftdown(H,i);
   }
```

i는 노드번호의 index

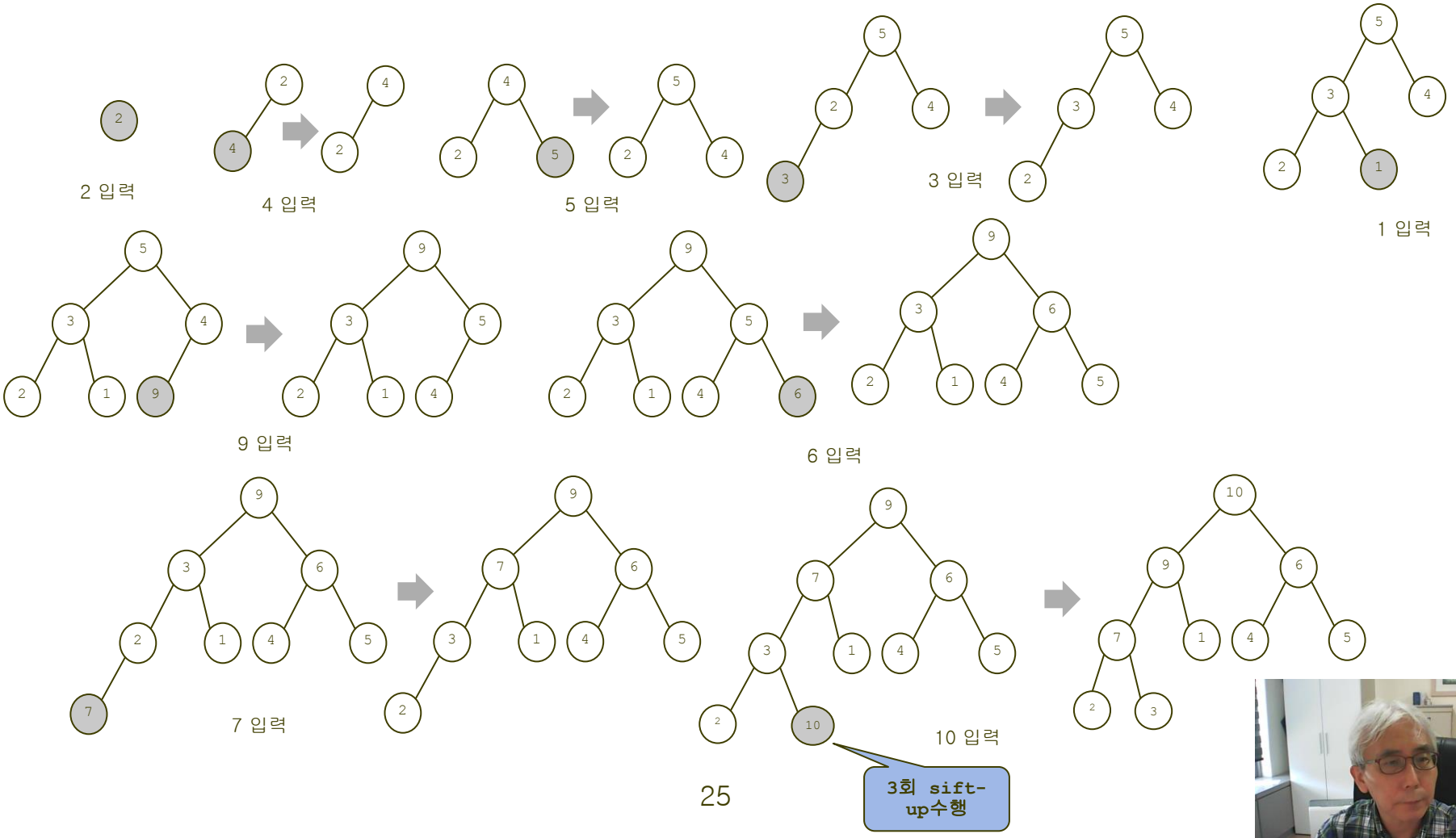
- make heap 방법

- 방법1: 데이터가 입력되는 순서대로 heap을 매번 구성

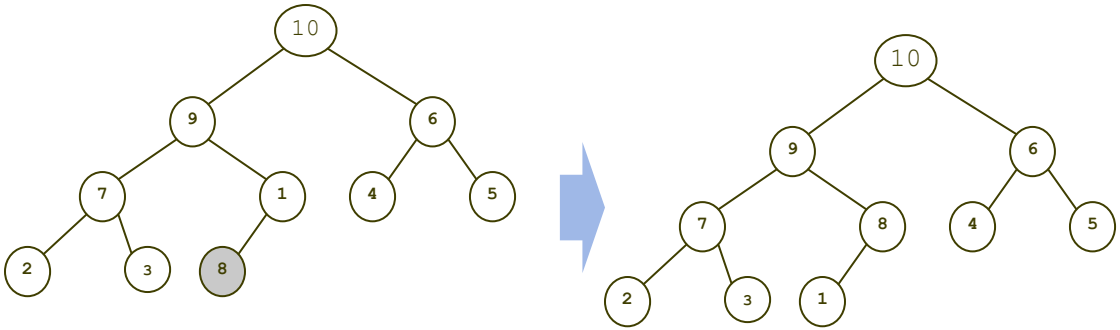
- 방법2: 모든 데이터를 트리에 넣은 상태에서 heap 구성

데이터 : 2 4 5 3 1 9 6 7 10 8

(방법1) sift-up 수행, 데이터가 입력되는 순서대로 heap을 매번 구성



데이터 : 2 4 5 3 1 9 6 7 10 8



8 입력

[실습프로그램] 방법2를 이용하여 makeHeap 구현하고 힙정렬 구현


```
import math
class Heap(object):
    n=0

    def __init__(self, data):
        self.data=data
# heap size를 하나 줄여야 한다. 인덱스는 1부터. 인덱스의 2* 연산 가능하도록.
        self.n=len(self.data)-1

    def addElt(self,elt):

            구현


    def siftUp(self, i):
        while(i>=2):

             구현
```

```
def siftDown(self,i):

     구현


def makeHeap2(self):

     구현

def root(self):

        구현

        if(self.n>0):
# 추가 하였음. 힙 이 더 이상없을 때는 down 없음

             구현
        return keyout

def removeKeys(self):


          구현
def heapSort(a):

          구현
```


```
# 인덱스를 간단히 하기 위해 처음 엘리먼트 0 추가
a=[0,11,14,2,7,6,3,9,5]
b=Heap(a)
b.makeHeap2()
print(b.data)
b.addElt(50)
print(b.data)
s=heapSort(a)
print(s)
```

```
[0, 14, 11, 9, 7, 6, 3, 2, 5]
[0, 50, 14, 9, 11, 6, 3, 2, 5, 7]
[50, 14, 11, 9, 7, 6, 5, 3, 2]
>>>
```

[실습프로그램] 방법1을 이용하여 makeHeap 구현

```
def makeHeap1(self):

      구 현

a=[0,11,14,2,7,6,3,9,5]
b=Heap(a)
b.makeHeap1()
print(b.data)
s=heapSort(a)
print(s)
```

```
[0, 14, 11, 9, 7, 6, 2, 3, 5]
[14, 11, 9, 7, 6, 5, 3, 2]
>>>
```