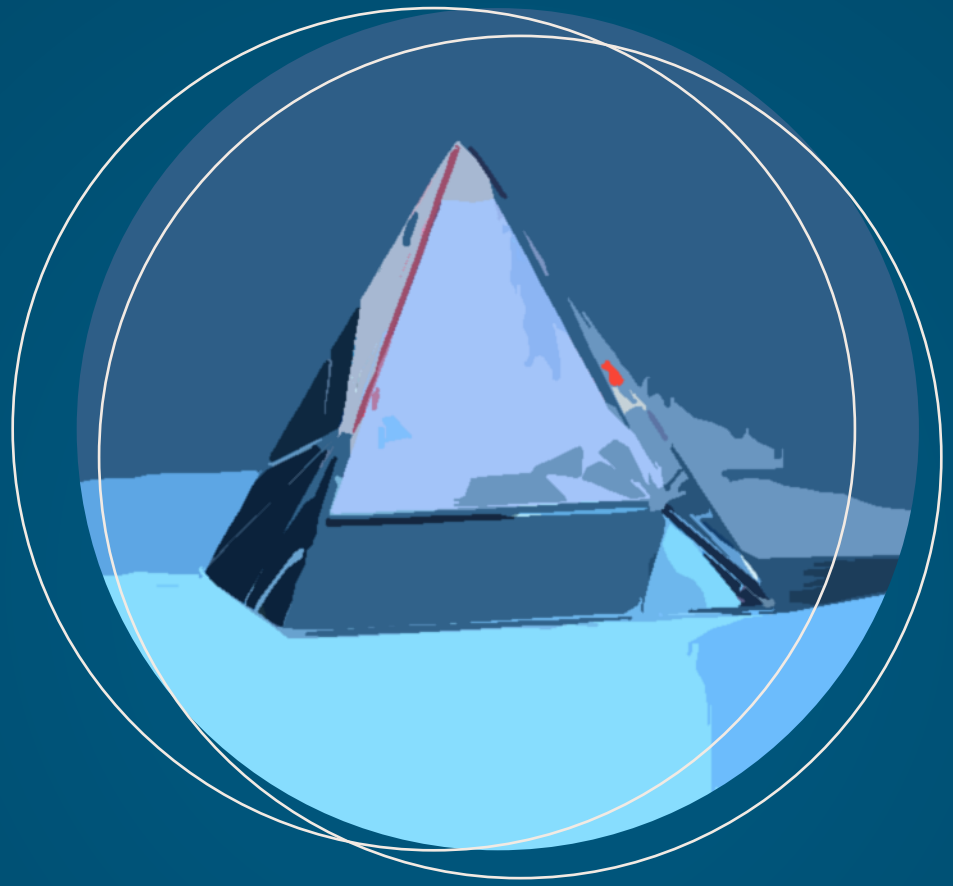
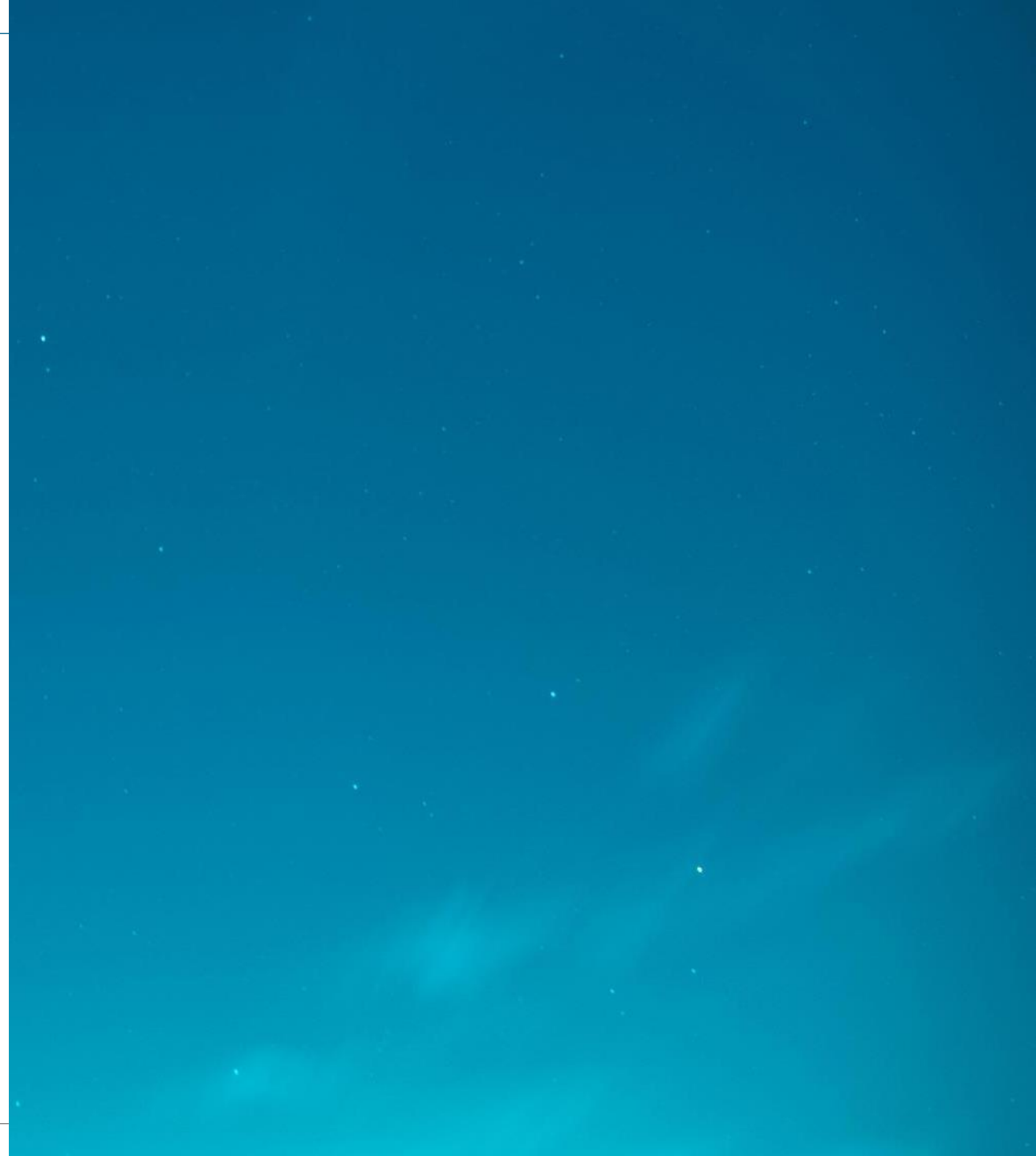
컴퓨터정보학과 2016270202
강민규



Shell sort 와 Merge sort(1,2) 를 이용한 space-time trade-off 분석

목차 Table of Contents



2 Space -time tradeoff

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

space –time tradeoff ?

컴퓨터 분야에서 space –time tradeoff란
- 큰 메모리로 적은 시간의 문제를 품
- 작은 메모리로 큰 시간의 문제를 품

의 상관 관계를 의미함

데이터 저장 (압축)

Look up table (cache)

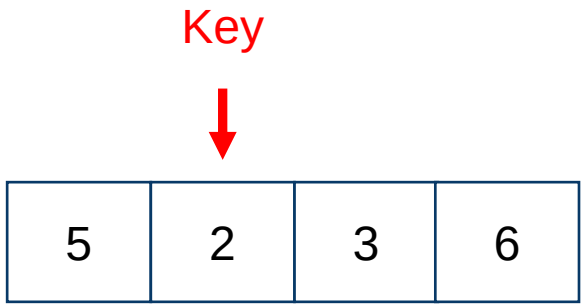
Merge sort

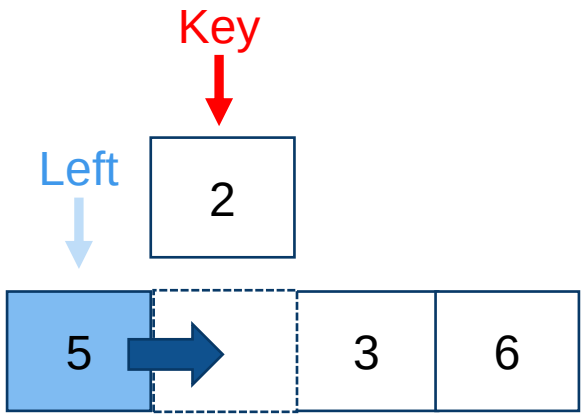
Shell sort

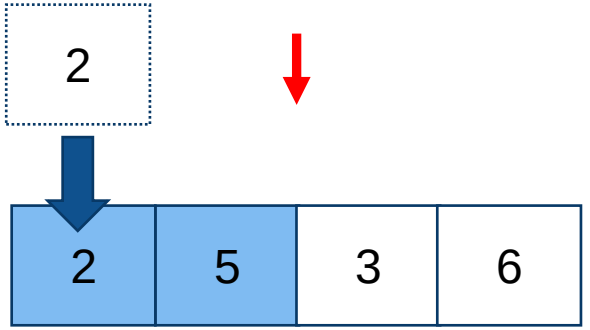
- selction sort
- shell sort
- code

2 삽입 정렬

idea







If Left > Key ? move :stop

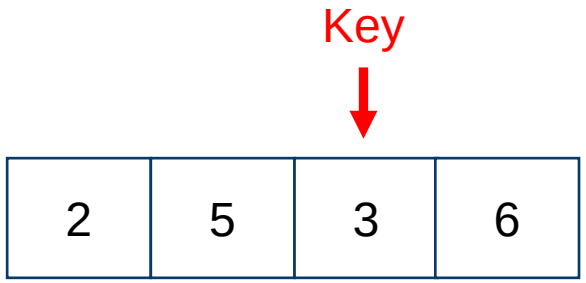
1.Key 값 선정

2.왼쪽 요소와 비교(반복)

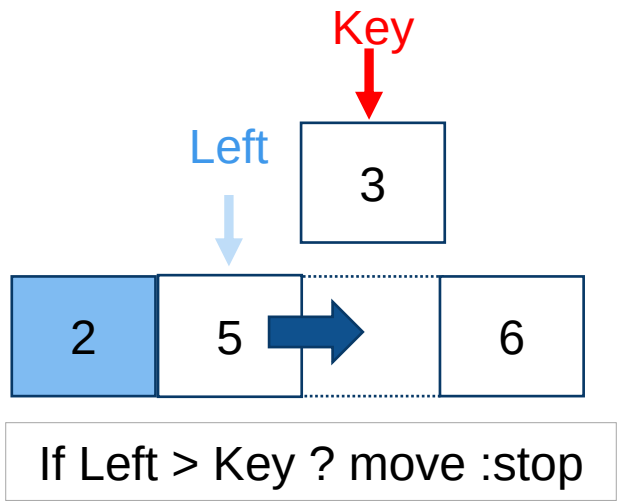
3.Stop 시 삽입

2 삽입 정렬

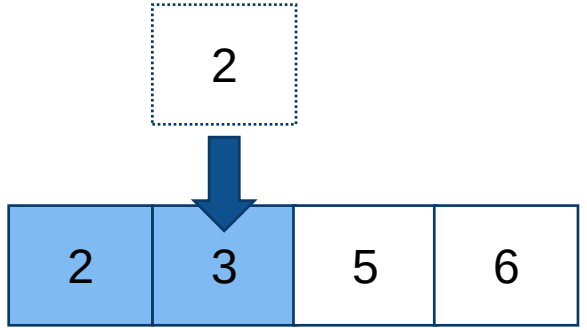
idea



1.Key 값 선정



2. 5를 옮긴 후, 2보다 크니 Stop



3.Stop후 Key 값 삽입

2 삽입정렬

Selelcion sort code

Code

```
def selection_sort1(arr):
    i=j=key =0
    length=len(arr)

    for i in range(1,length):
        key=arr[i] ## key 값 선택
        j=i-1      ## key 왼쪽과  비교
        while j>=0 and arr[j] > key:
            arr[j+1] = arr[j]
            j=j-1
        arr[j+1]=key

    return arr
```

1.Key값 선정

2. 왼쪽 요소와 비교(반복) 후 move

3. stop 후 삽입

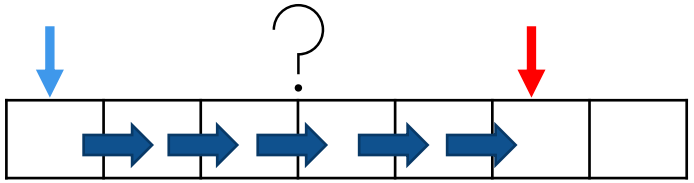
1 Shell Sort

idea

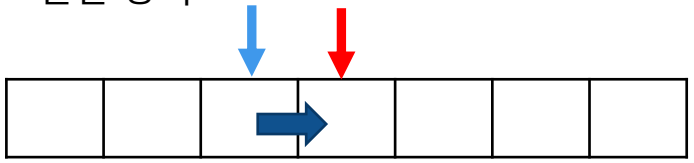
1 Donald L. Shell

-1995년 Donald L. Shell이 제안

2 삽입 정렬의 장,단점

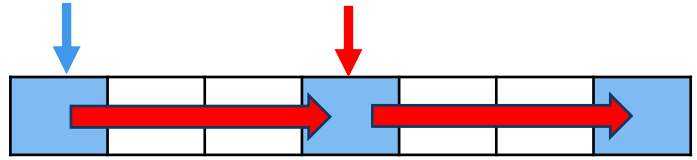


-비교 값들의 간의 거리가 멀면 이동 연산 증가

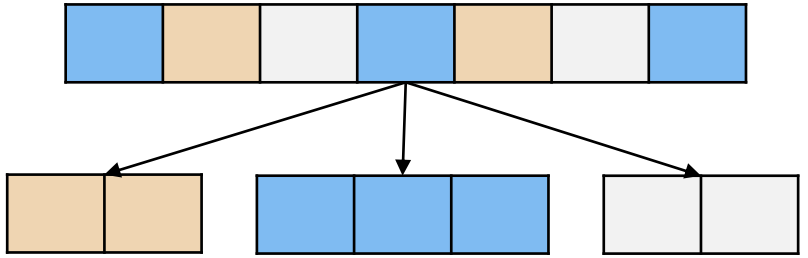


-어느 정도 정렬된 배열 내에서는 속도 빠름

3 삽입 정렬 응용



-한칸 씩 이동하던 거리를 gap만큼 이동



-Gap만큼 떨어진 부분 리스트들끼리 삽입 정렬시킨다

2 삽입정렬

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

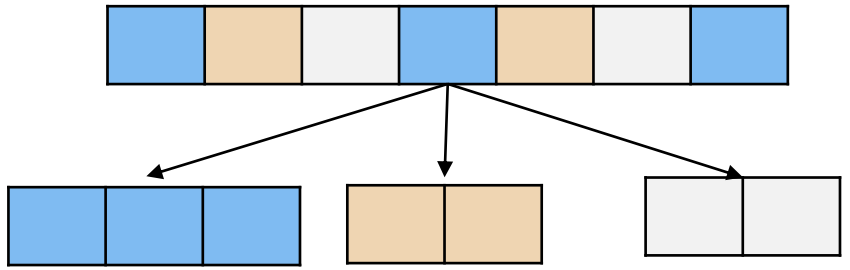
Code

```
def shell_sort(arr):
    size=len(arr)
    gap=size//2
    while gap > 0 :
        for i in range(0,gap):
            selection_sort(arr,i,size,gap)
        gap= gap//2
    return arr

def selection_sort(arr,first,size,gap):
    for i in range(first+gap,size):
        key=arr[i]
        j=i-gap
        while j>=first and arr[j] > key:
            arr[j+gap] = arr[j]
            j=j-gap
        arr[j+gap]=key
```

→ gap은 1/2로 줄여나감

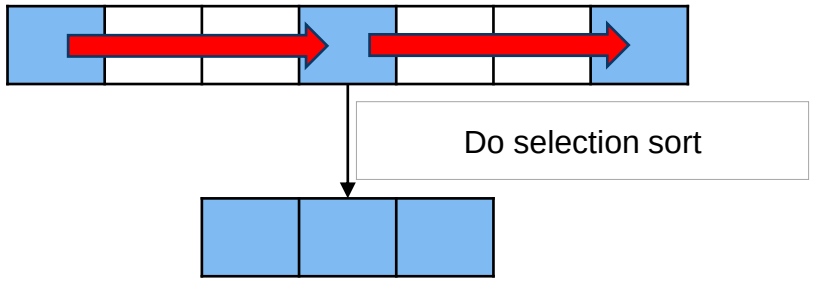
→ 부분 리스트 생성

[부분 리스트 생성]



→ 2. (시작점,size,gap)을 이용해 삽입정렬

[삽입정렬]



Merge sort

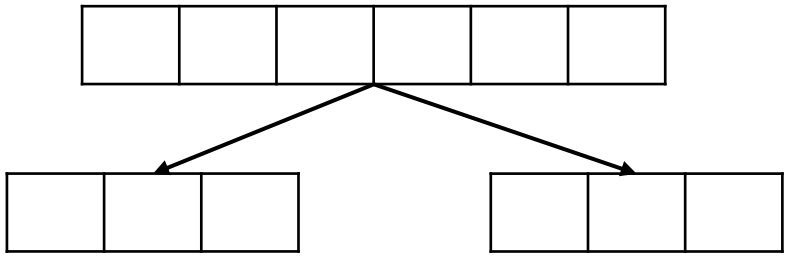
-Merge sort1,2

3 Merge sort 비교

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

1 Merge sort 1(out -of –place)

[두개의 배열 생성]



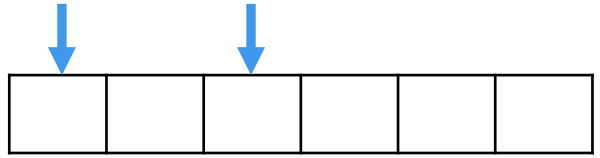
```
void mergesort (int n, keytype S[]) {
    const int h = n/2, m = n − h;
    keytype U[1..h], V[1..m];      // 추가 메모리(array)
```

-두개의 하위 배열 생성
- 추가 메모리 필요

VS

1 Merge sort 2(in–place)

[인덱스 전달]



```
void merge (h, m, U[], V[], S[]) {
    index i, j, k;      // 추가 메모리(no array)
```

-전역 배열의 인덱스를 넘겨줌
- merge1에 비해 약간 느림

2 Merge sort 1(extra array)

Merge sort1 code

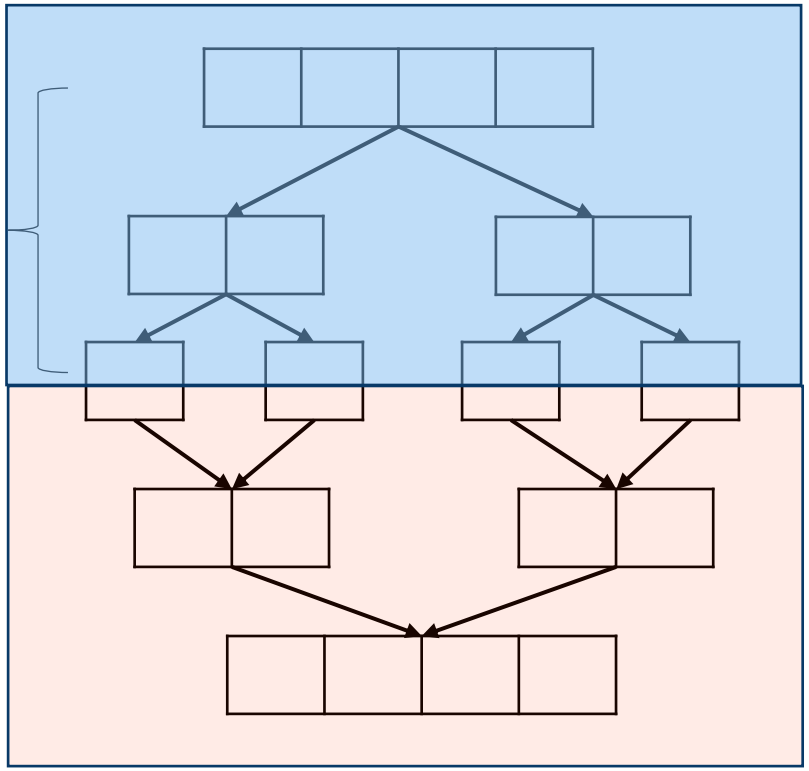
Code

```
def merge_sort1(arr):
    half=len(arr)//2
    if half==0:
        return arr
    arr_left=arr[:half]
    arr_right=arr[half:]
    left_result=merge_sort1(arr_left)
    right_result=merge_sort1(arr_right)
    return merge(left_result,right_result)

def merge(left_arr,right_arr):
    i=0;j=0
    sum_arr=[]

    while(i<len(left_arr) and j<len(right_arr)):
        if (left_arr[i] > right_arr[j]):
            sum_arr.append(right_arr[j])
            j+=1
        else:
            sum_arr.append(left_arr[i])
            i+=1
    if (i==len(left_arr)):
        sum_arr.extend(right_arr[j:])
    if(j==len(right_arr)):
        sum_arr.extend(left_arr[i:])
    return sum_arr
```

1.배열 생성 후, **분할**

2. 좌우 배열 비교

3. 남은 배열 삽입



2 Merge sort 2(in-place)

Merge sort 2 (in-place)

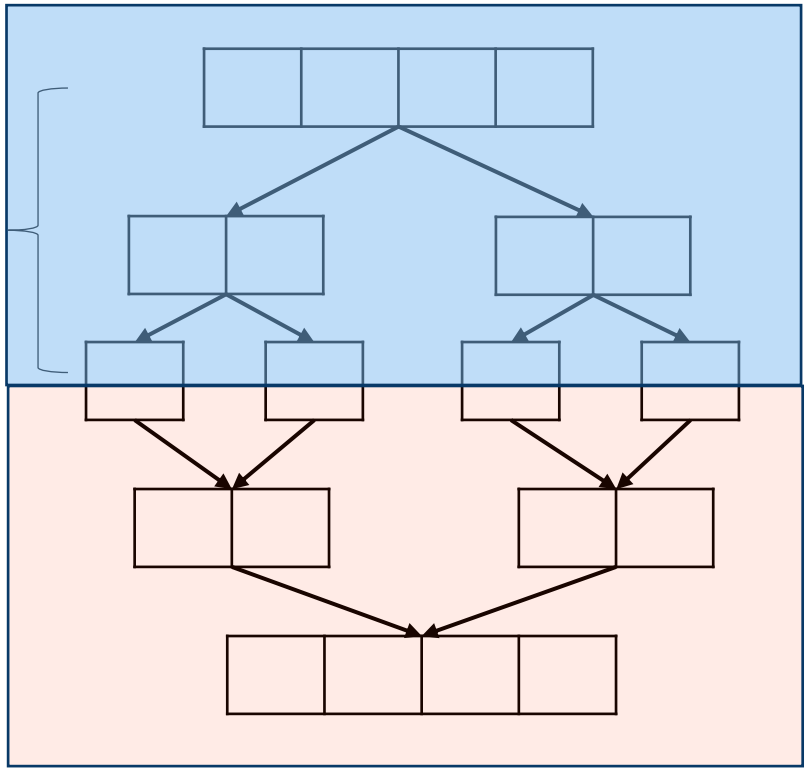
Code

```
def mergesort2(arr,left,right):
    if(right-left>=1):
        mid=(right+left)//2
        mergesort2(arr,left,mid)
        mergesort2(arr,mid+1,right)
        return merge_def(arr,left,mid,right)
    else:
        return

def merge_def(arr,left,mid,right):
    sum_arr = []
    i = left
    j = mid+1
    while (i <= mid and j <= right):
        if (arr[i] < arr[j]):
            sum_arr.append(arr[i])
            i+=1
        else:
            sum_arr.append(arr[j])
            j+=1
    if (i>mid):
        sum_arr.extend(arr[j:right+1])
    if (j>right):
        sum_arr.extend(arr[i:mid+1])
    arr[left:right+1] = sum_arr
    return arr
```

1.분할 과정에서 **인덱스** 전달

2. 좌우 배열 비교

3. 남은 배열 삽입



성능비교

- 시간복잡도
- 공간복잡도

3 사양

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

개발환경,언어	PyCharm Community Edition 2020.1.2 Python 3.6.4
CPU	Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz 3.60 GHz
RAM	8.00GB

4 시간,공간 복잡도

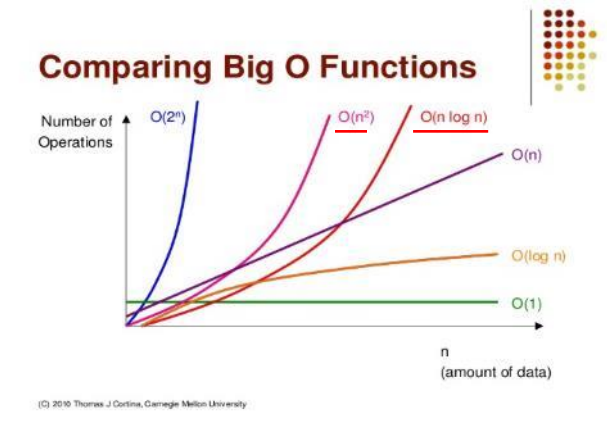
Lorem Ipsum is simply dummy text of the printing and typesetting industry.

	Best	Avg	Worst	Space complexity
Shell sort	O(n)	$O(n^{1.5})$	$O(n^2)$	O(1)
Merge sort(out-of-place)	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Merge sort(in-place)	O(nlogn)	O(nlogn)	O(nlogn)	O(n)

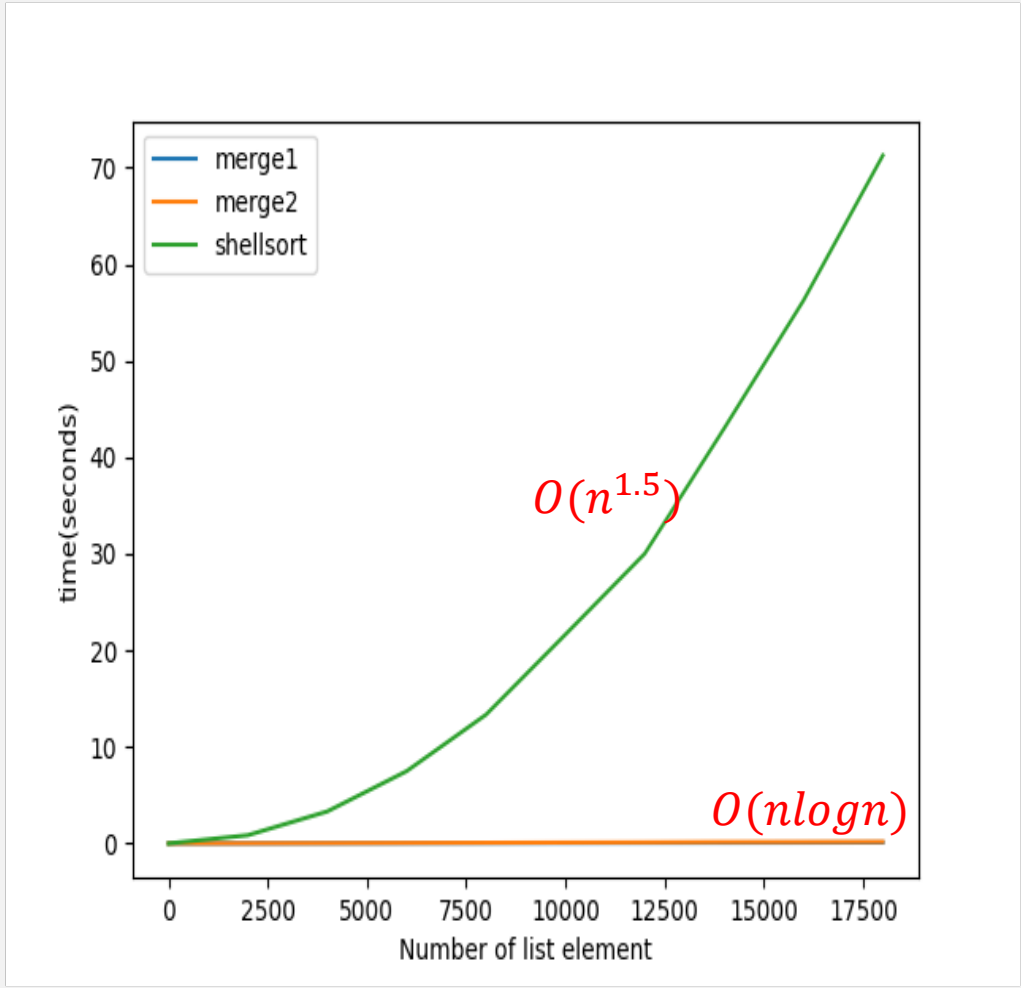
4 시간 복잡도

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

- $O(n^{1.5})$인 shell sort가 가장 느린 모습을 보인다
- 같은 시간 복잡도인 merge sort는 비슷한 시간 소요를 보인다



	Best	Avg	Worst	Space complexity
Shell sort	O(n)	$O(n^{1.5})$	$O(n^2)$	O(1)
Merge sort1	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Merge sort2	O(nlogn)	O(nlogn)	O(nlogn)	O(n)

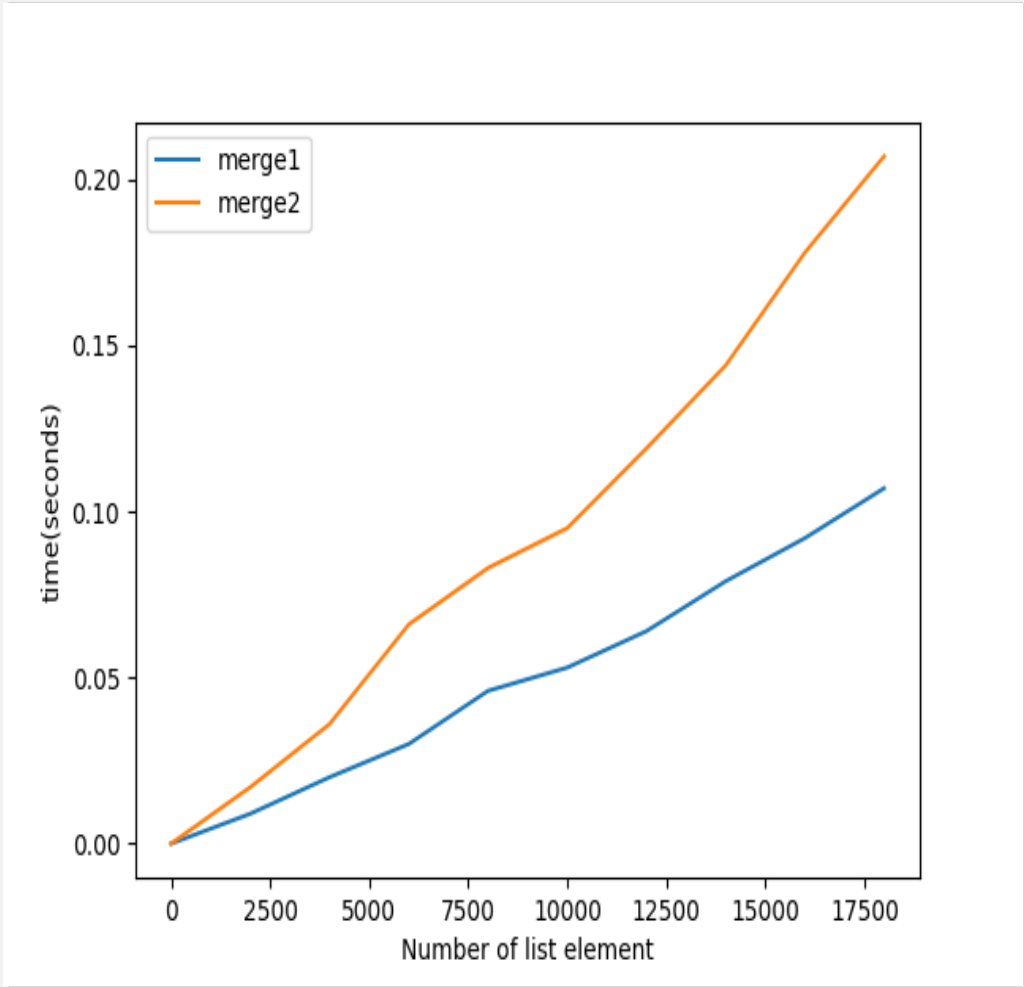


4 시간 복잡도

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

- 같은 시간 복잡도인 O(nlogn)
- In-place 인 merge sort 2 가 조금 느린 모습 (마지막 배열 복사 과정)

	Best	Avg	Worst	Space complexity
Shell sort	O(n)	$O(n^{1.5})$	$O(n^2)$	O(1)
Merge sort1	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Merge sort2	O(nlogn)	O(nlogn)	O(nlogn)	O(n)

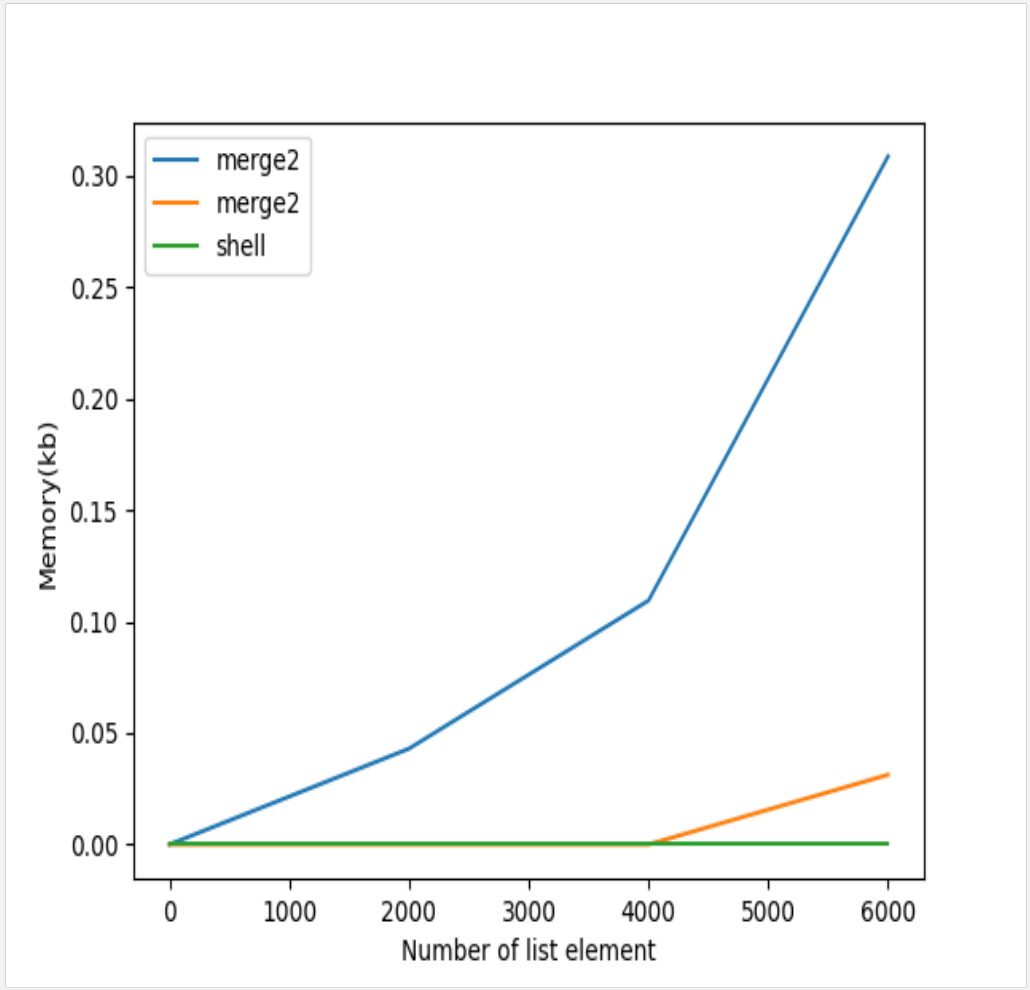


4 공간 복잡도

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

- 공간복잡도는 shell sort – *O(1)* / merge sort – *O(n)*
- 실제로는 추가 배열이 필요한 merge1은 더 많은 메모리 사용량을 보임

	Best	Avg	Worst	Space complexity
Shell sort	O(n)	$O(n^{1.5})$	$O(n^2)$	O(1)
Merge sort1	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Merge sort2	O(nlogn)	O(nlogn)	O(nlogn)	O(n)



1 결론

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

1 결론

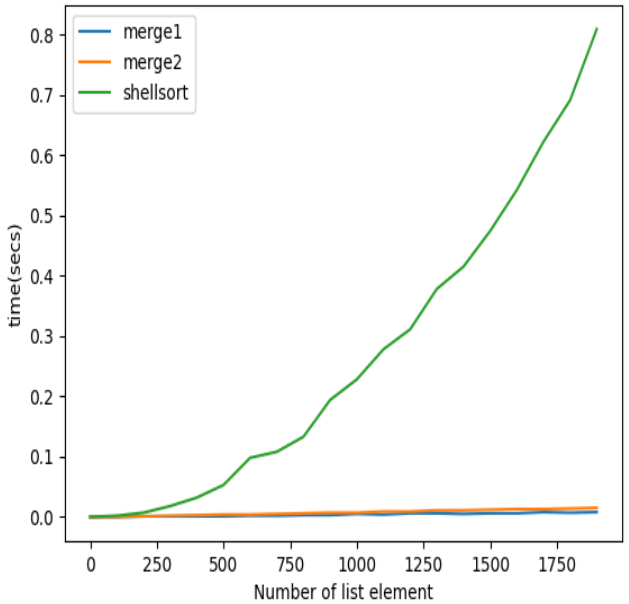
-시간 복잡도:
shell>merge2(in-place)>=merge1

-공간복잡도:
merge1>=merge2(in-place)>shell

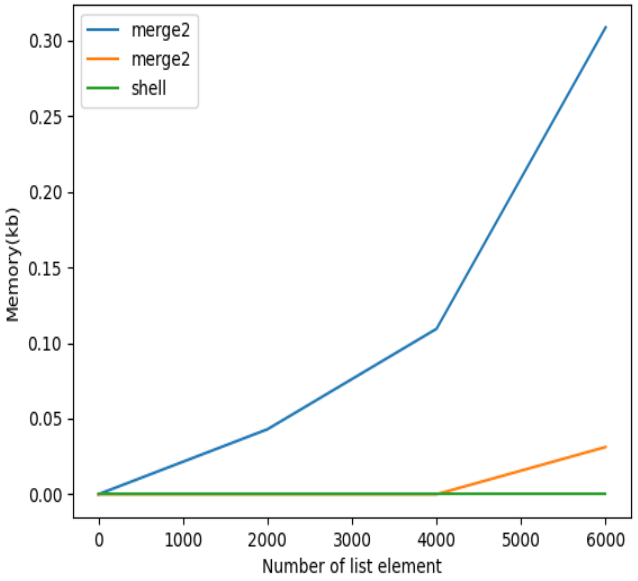
→적당히 작은 n에서 merge,shell sort는 space –time trade-off 가 존재함을 보여준다.

.

2 소요 시간 관찰 결과



3 메모리 사용량 관찰 결과



THANK YOU