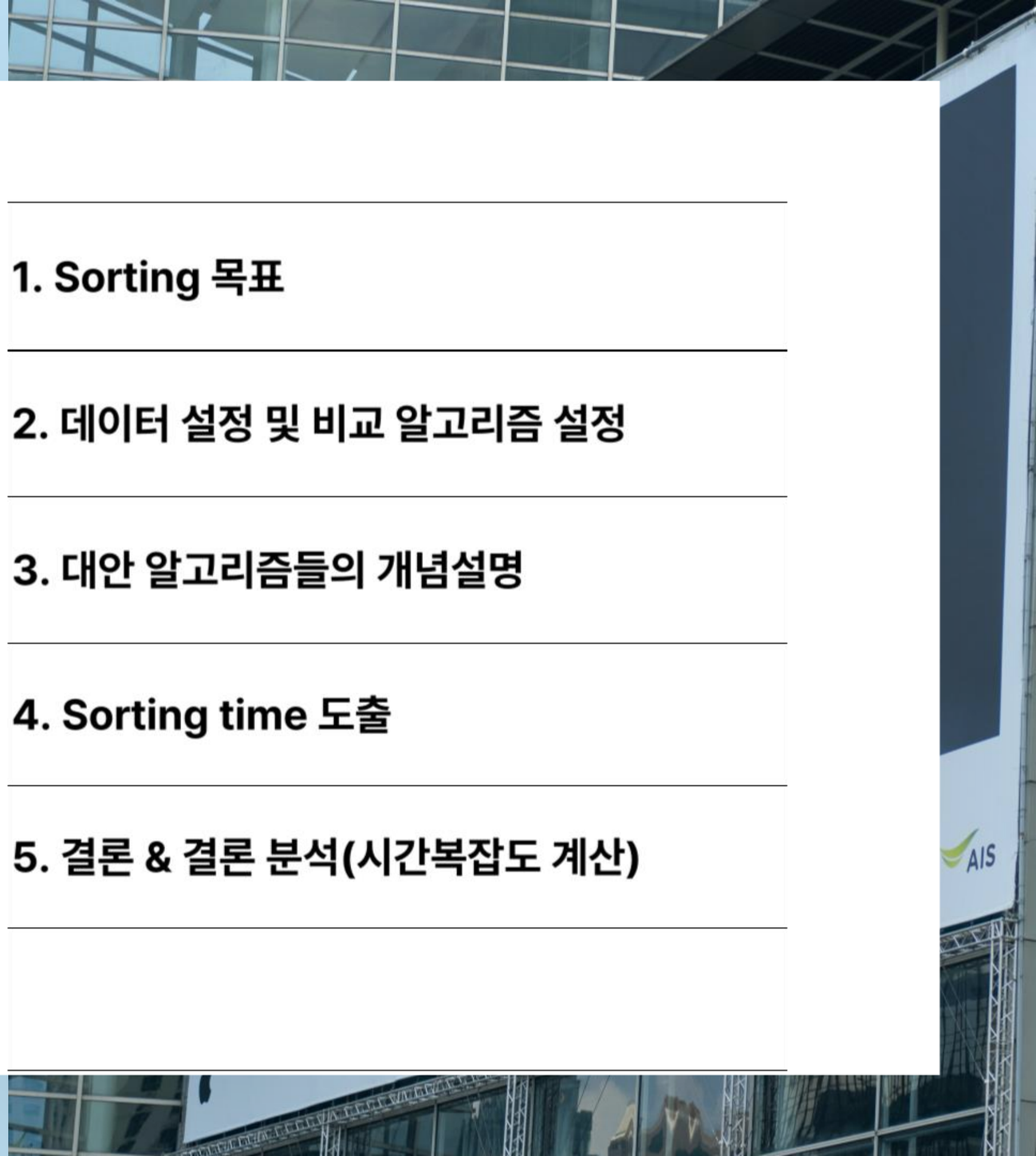
Ticker sorting 방법 개선

LIGHT 조

목차



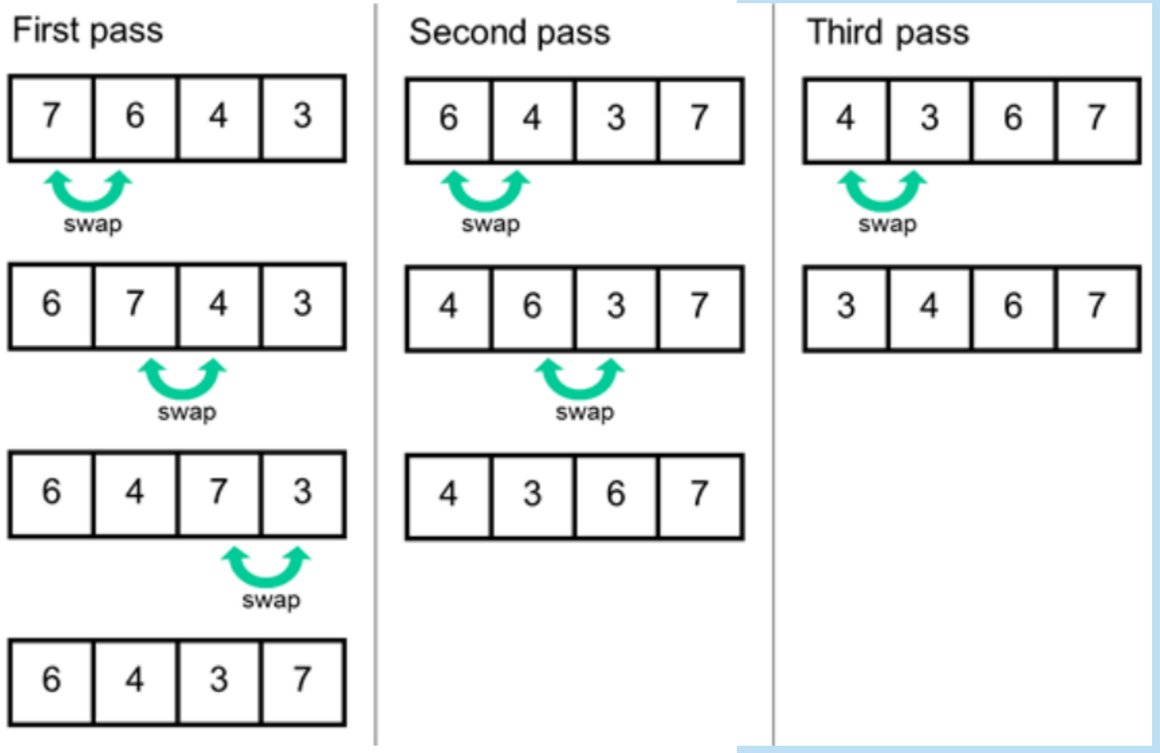
Sorting 목표

Bubble sort 방식보다 빠르며,
트레이딩하는 주식의 수가 수백 개, 수천 개로 늘어났을 때 적합한 sorting 방식 찾기

Bubble sort

- sorting 속도는 느리지만 코드가 단순하다는 이점 존재
- sorting 대상의 수가 많을 때 적합하지 않은 방식

sorting 속도는 곧 매매 속도이므로 sorting 방식 설정은
증권사의 경쟁력에 영향을 미치는 중요한 결정이다.



- **Bubble sort**

```
[ ]  def bubble_sort(arr):
         n = len(arr)
         for i in range(n):
             for j in range(0, n-i-1):
                 if arr[j] > arr[j+1]:
                     arr[j], arr[j+1] = arr[j+1], arr[j]
         return arr
```

데이터 설정

Financial Data Reader를 이용하여
한국거래소(KRX)가 상장한 국내 주식 2774개와 나스닥(NASDAQ)에 상장된 해외 주식 3958개의 주가 데이터로 설정

<국내주식>

```
stocks = fdr.StockListing('KRX')
stocks
```

	Code	ISU_CD	Name	Market
0	005930	KR7005930003	삼성전자	KOSPI
1	373220	KR7373220003	LG에너지솔루션	KOSPI
2	000660	KR7000660001	SK하이닉스	KOSPI
3	207940	KR7207940008	삼성바이오로직스	KOSPI
4	005935	KR7005931001	삼성전자우	KOSPI
...	...	...	...	...
2769	217320	KR7217320001	썬테크	KONEX
2770	245450	KR7245450002	씨앤에스링크	KONEX
2771	288490	KR7288490006	나라소프트	KONEX
2772	308700	KR7308700004	테크엔	KONEX
2773	322190	KR7322190000	베른	KONEX

<해외주식>

```
[ ] us_stocks = fdr.StockListing('NASDAQ')
    us_stocks
```

100%|

	Symbol	Name	IndustryCode
0	AAPL	Apple Inc	57106020
1	MSFT	Microsoft Corp	57201020
2	AMZN	Amazon.com Inc	53402010
3	NVDA	NVIDIA Corp	57101010
4	GOOGL	Alphabet Inc Class A	57201030
...	...	...	...
3953	NNAGR	99 Acquisition Group Right Exp 25 August 2028	55601010
3954	MURAV	Mural Oncology PLC	56202010
3955	ALKSV	Alkermes Plc	56202010
3956	RR	Richtech Robotics Inc	53204030
3957	MURA	Mural Oncology PLC	56202010

3958 rows × 4 columns

시간 측정 기준 설정

- time 함수 사용
- Average, Worst, Best sorting time 각각의 시간을 최대한 단축시키는 방식 찾기
- Average, Worst, Best sorting time은 각각의 sorting 방식들을 1000번씩 반복하여 도출하였다.

대안 알고리즘 설정

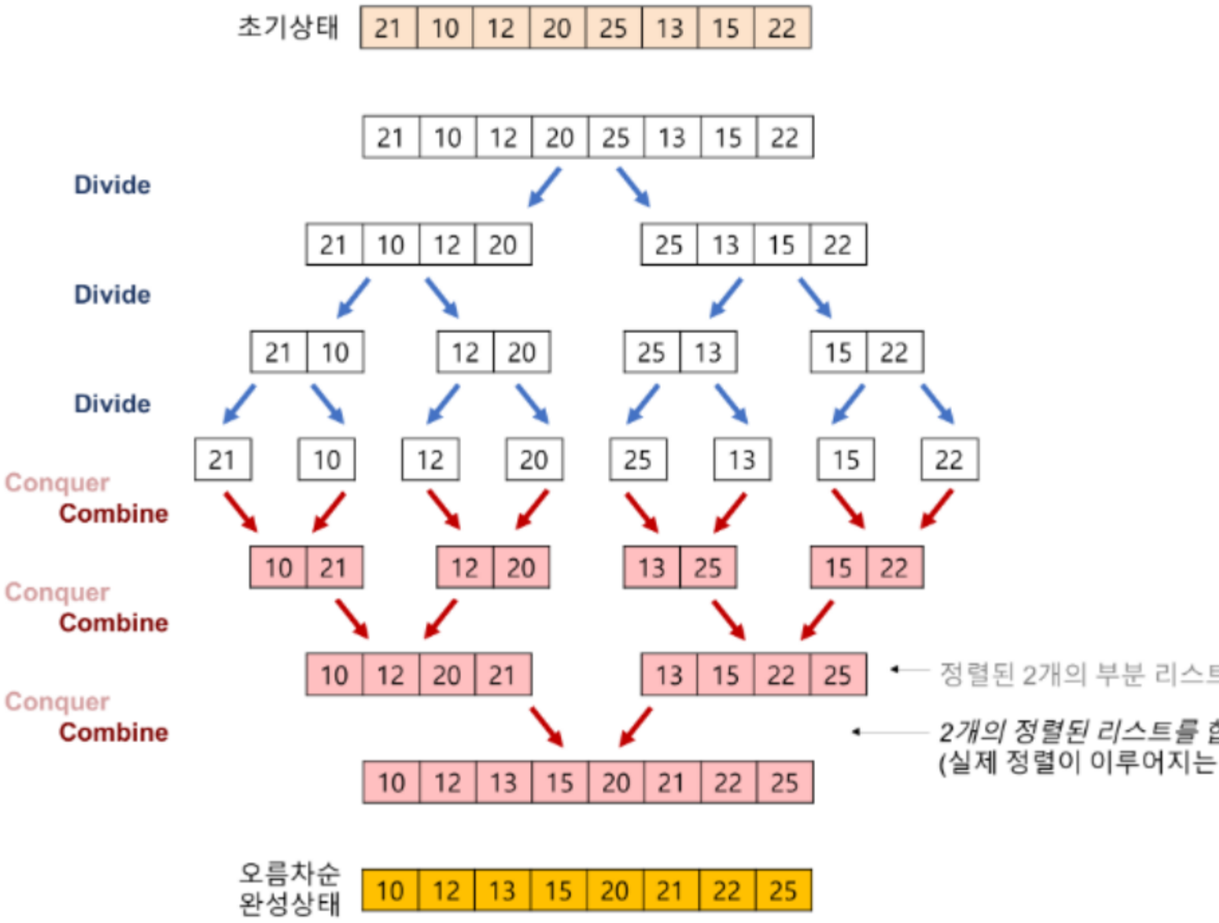
- Quick sort
- Merge sort
- Counting sort
- Radix sort

대안 알고리즘 실행

Merge sort

리스트를 균등한 2개의 부분 리스트로 분할하고 각각 정렬한 후 합병하여 완성하는 방식

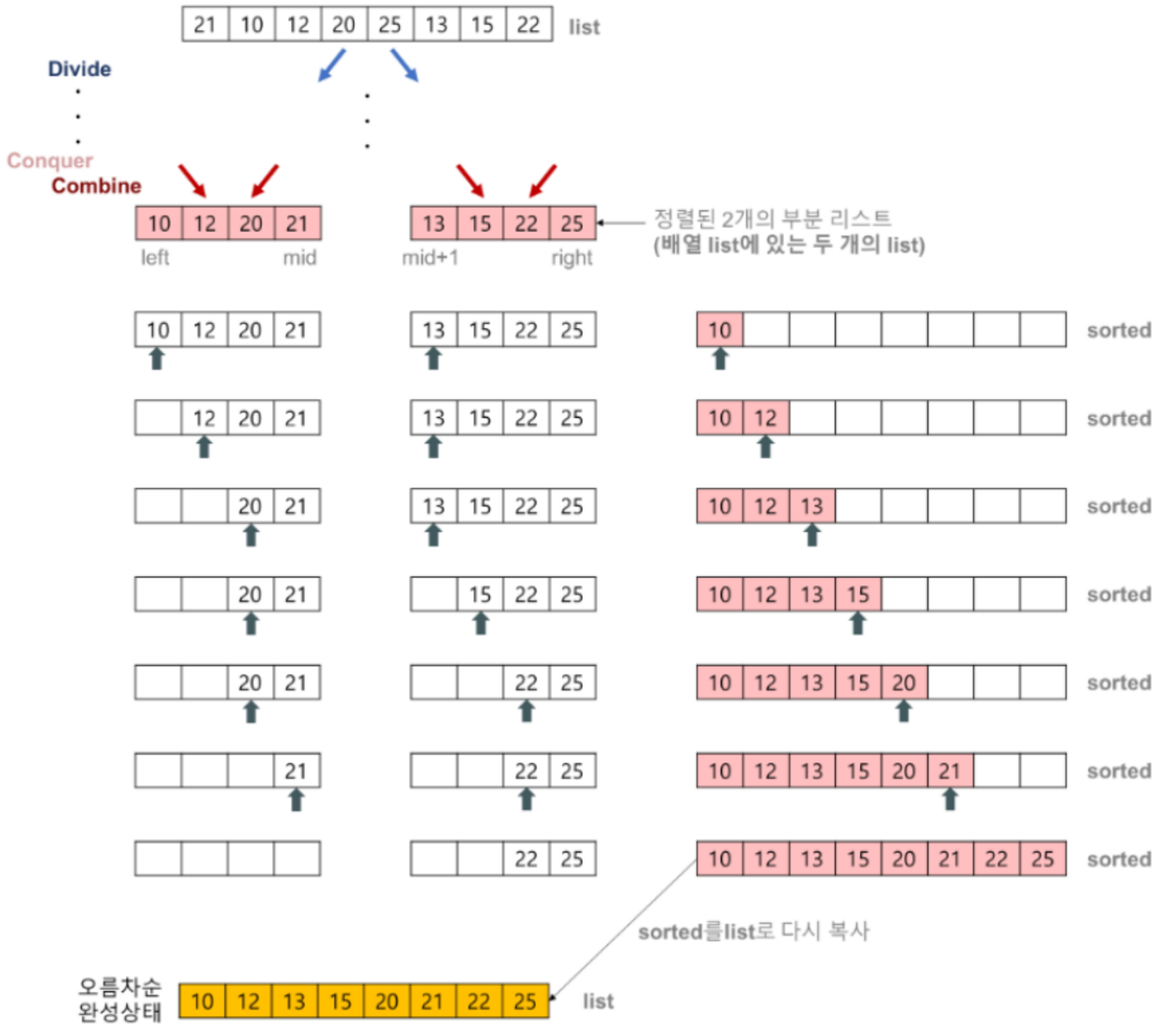
1단계. 리스트를 크기가 1이 될때까지 균등한 2개의 부분 리스트로 분할
2단계. 2개의 부분 리스트 값들을 처음부터 하나씩 비교
3단계. 2개의 리스트 값 중 더 작은 값을 새로운 리스트로 이동
4단계. 둘 중 하나가 끝날 때까지 반복하고 남은 리스트의 값들은 그대로 이동
5단계. 새로운 리스트를 원래의 리스트로 이동
.
.
.
n단계. 분할된 횟수만큼 2~5단계 반복



대안 알고리즘 실행

Merge sort

리스트를 균등한 2개의 부분 리스트로 분할하고 각각 정렬한 후 합병하여 완성하는 방식



2단계. 2개의 부분 리스트 값들을 처음부터 하나씩 비교

3단계. 2개의 리스트 값 중 더 작은 값을 새로운 리스트로 이동

4단계. 둘 중 하나가 끝날 때까지 반복하고 남은 리스트의 값들은 그대로 이동

5단계. 새로운 리스트를 원래의 리스트로 이동

Merge sort

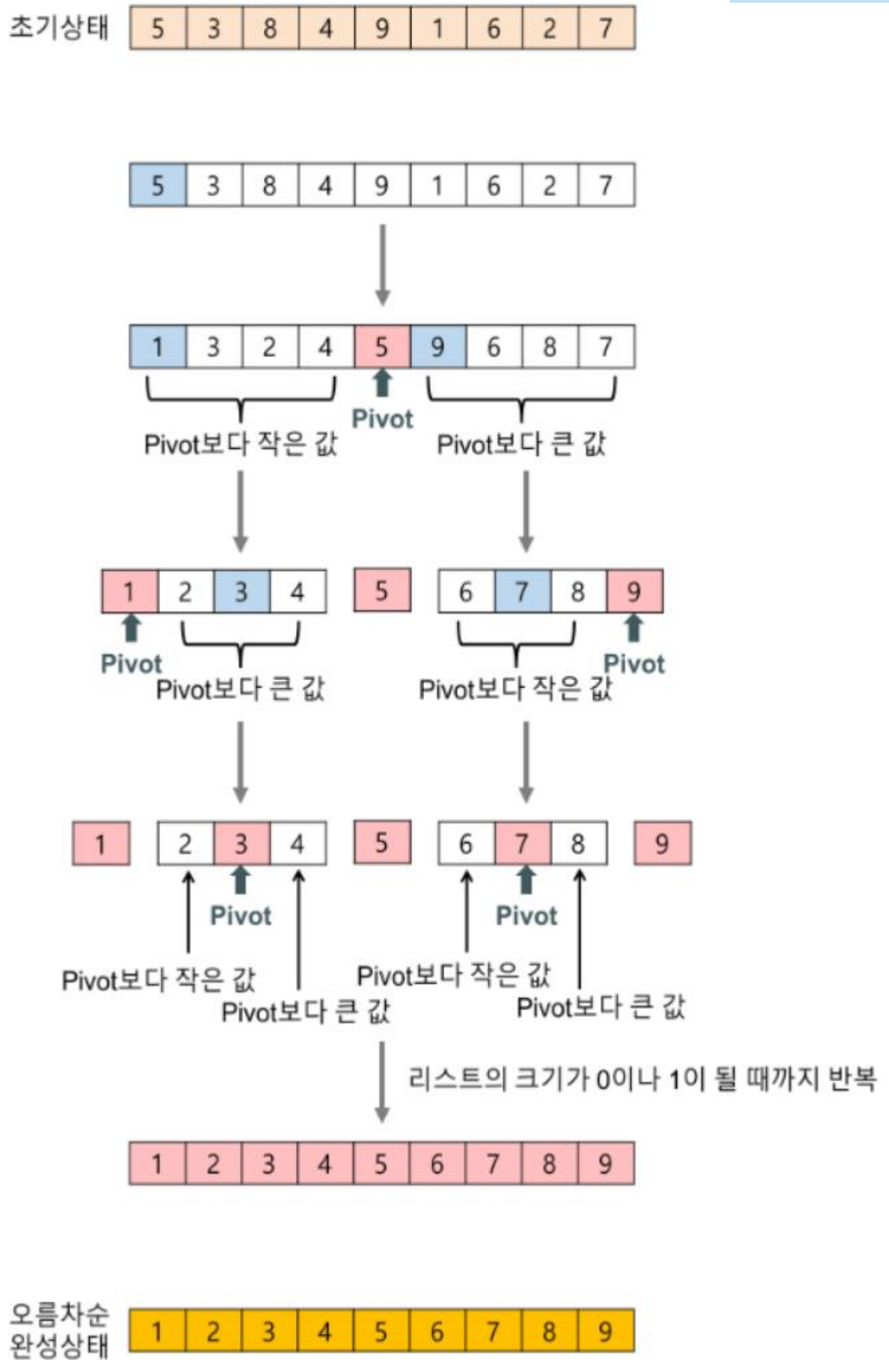
```python
def merge_sort(arr):
    if len(arr) < 2:
        return arr

    mid = len(arr) // 2
    left_arr = merge_sort(arr[:mid])
    right_arr = merge_sort(arr[mid:])

    merged_arr = []

    i = j = 0
    while i < len(left_arr) and j < len(right_arr):
        if left_arr[i] < right_arr[j]:
            merged_arr.append(left_arr[i])
            i += 1
        else:
            merged_arr.append(right_arr[j])
            j += 1
    merged_arr += left_arr[i:]
    merged_arr += right_arr[j:]
    return merged_arr
```

대안 알고리즘 실행

Quick sort

리스트를 비균등한 2개의 부분 리스트로 나누어 각각 정렬한 뒤 합쳐 완성하는 방식

1단계. 리스트 안에 있는 요소 하나 선택
2단계. 고른 요소를 기준으로 작은 요소들은 모두 왼쪽으로, 큰 요소들은 모두 오른쪽으로 자리이동
3단계. 고른 요소를 기준으로 분할된 2개의 부분 리스트에서도 다시 요소 하나씩 선택
4단계. 부분 리스트에서도 고른 요소들을 기준으로 자리이동하고 다시 2개의 부분 리스트로 분할
.
.
.
n단계. 부분 리스트들의 크기가 0이나 1이여서 더 이상 분할이 불가능할 때까지 반복



Quick sort

```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    lesser_arr, equal_arr, greater_arr = [], [], []
    for num in arr:
        if num < pivot:
            lesser_arr.append(num)
        elif num > pivot:
            greater_arr.append(num)
        else:
            equal_arr.append(num)
    return quick_sort(lesser_arr) + equal_arr + quick_sort(greater_arr)
```

대안 알고리즘 실행

Counting sort

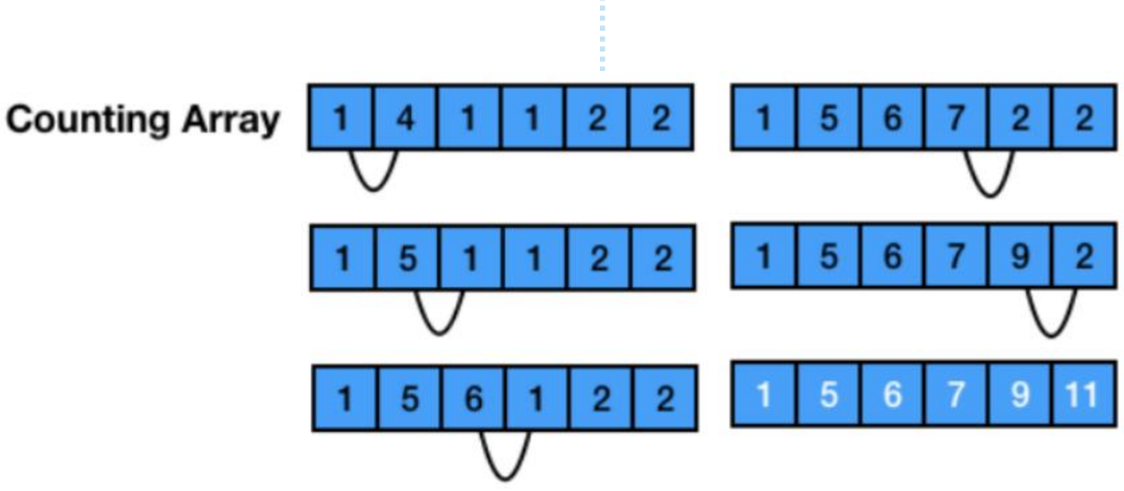
리스트를 균등한 2개의 부분 리스트로 분할하고 각각 정렬한 후 합병하여 완성하는 방식

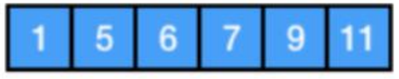
1단계. 리스트 안 각각의 값들이 중복되는 횟수를 계산

2단계. 카운팅 리스트를 만들어 중복 횟수를 저장

3단계. 카운팅 리스트 각각의 값에 대해 직전의 값을 더한다.

4단계. 입력 리스트와 동일한 크기의 출력 리스트를 만들어 입력 값들을 역순으로 출력 리스트에 채운다.





c[0] = 1 -> b1, 한 자리를 0이 차지한다.

c[1] = 5 -> b2 ~ b5, 네 자리를 1가 차지한다.

c[2] = 6 -> b6, 한 자리를 2이 차지한다.

c[3] = 7 -> b7, 한 자리를 3가 차지한다.

c[4] = 9 -> b8 ~ b9, 두 자리를 4가 차지한다.

c[5] = 11 -> b10 ~ b11, 두 자리를 5이 차지한다.

Counting sort

```python
def counting_sort(arr):
    max_length = max(len(s) for s in arr)
    sorted_arr = arr[:]

    # 오른쪽 끝을 기준으로 정렬합니다 (문자열의 앞자리부터 정렬하기 위해)
    for i in range(max_length - 1, -1, -1):
        count = [0] * 256  # ASCII 문자셋 기준
        output = [''] * len(arr)

        for string in sorted_arr:
            if i < len(string):
                index = ord(string[i])
                count[index] += 1
            else:
                count[0] += 1

        for j in range(1, 256):
            count[j] += count[j - 1]

        for string in reversed(sorted_arr):
            if i < len(string):
                index = ord(string[i])
                output[count[index] - 1] = string
                count[index] -= 1
            else:
                output[count[0] - 1] = string
                count[0] -= 1

        sorted_arr = output[:]

    return sorted_arr
```

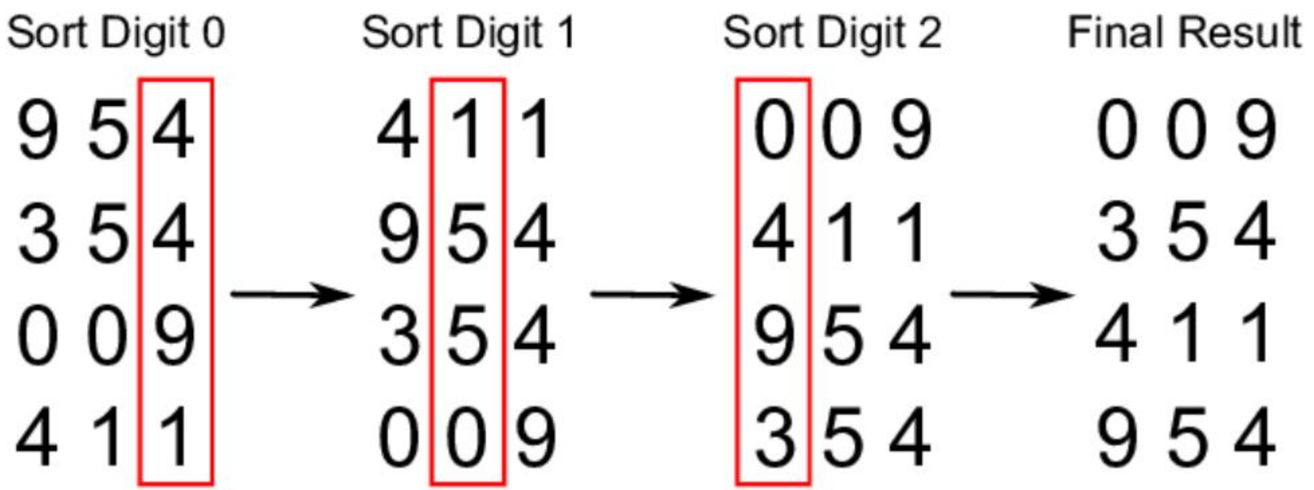
대안 알고리즘 실행

Radix sort

리스트 값들을 낮은 자릿수부터 정렬해가는 방식

1단계. 리스트의 값들 중 가장 큰 값의 자릿수 구하기

2단계. 1의 자릿수부터 리스트 안 가장 큰 값의 자랏수까지 해당 자릿수만을 보고 정렬 반복



Radix sort

```python
def counting(arr, place):
    n = len(arr)
    output = [0] * n
    count = [0] * 256  # ASCII 문자셋 기준

    for i in range(n):
        index = ord(arr[i][place]) if place < len(arr[i]) else 0
        count[index] += 1

    for i in range(1, 256):
        count[i] += count[i - 1]

    i = n - 1
    while i >= 0:
        index = ord(arr[i][place]) if place < len(arr[i]) else 0
        output[count[index] - 1] = arr[i]
        count[index] -= 1
        i -= 1

    for i in range(n):
        arr[i] = output[i]
```

```python
def radix_sort(arr):
    max_length = len(max(arr, key=len))

    for i in range(max_length - 1, -1, -1):
        counting(arr, i)

    return arr
```

시간 복잡도 계산하기 (n개의 데이터가 있다고 가정할때)

Bubble sort 평균 시간 복잡도: O(n^2)

1)

Quick sort

평균 시간 복잡도:
O(nlogn)

2)

Merge sort

평균 시간 복잡도:
O(nlogn)

3)

Radix sort

평균 시간 복잡도:
O(k*n)

4)

Counting sort

평균 시간 복잡도:
O(n+k)

결론 - 해외주식, 1000번 반복

```
Average Merge Sort Time: 0.009820643663406372 seconds
Average Quick Sort Time: 0.006682946681976319 seconds
Average Radix Sort Time: 0.004589187860488892 seconds
Average Counting Sort Time: 0.0067949244976043 7 seconds
```

```
Max Merge Sort Time: 0.020055294036865234 seconds
Max Quick Sort Time: 0.017874002456666504 seconds
Max Radix Sort Time: 0.010386705398559 57 seconds
Max Counting Sort Time: 0.013203620910644531 seconds
```

```
Min Merge Sort Time: 0.008576869964599 61 seconds
Min Quick Sort Time: 0.005311727523803711 seconds
Min Radix Sort Time: 0.003263711929321289 seconds
Min Counting Sort Time: 0.006313800811767578 seconds
```

결론 - 국내주식, 1000번 반복

```
Average Merge Sort Time: 0.012363755941390992 seconds
Average Quick Sort Time: 0.007466354131698608 seconds
Average Radix Sort Time: 0.007202440023422242 seconds
Average Counting Sort Time: 0.011763966798782348 seconds
```

```
Max Merge Sort Time: 0.03331136703491211 seconds
Max Quick Sort Time: 0.018541574478149414 seconds
Max Radix Sort Time: 0.015247344970703125 seconds
Max Counting Sort Time: 0.024762868881225586 seconds
```

```
Min Merge Sort Time: 0.010327577590942383 seconds
Min Quick Sort Time: 0.0060086250305175781 seconds
Min Radix Sort Time: 0.004429101943969726 seconds
Min Counting Sort Time: 0.008980751037597656 seconds
```

결론 - 해외주식, 1000번 반복

```
Average Merge Sort Time: 0.02156732892990112 seconds
Average Quick Sort Time: 0.010938447952270508 seconds
Average Radix Sort Time: 0.01683957552909851 seconds
Average Counting Sort Time: 0.014003098964691162 seconds


Max Merge Sort Time: 0.044050931930541990 seconds
Max Quick Sort Time: 0.03339028358459473 seconds
Max Radix Sort Time: 0.038945913314819336 seconds
Max Counting Sort Time: 0.034499168395996094 seconds


Min Merge Sort Time: 0.015910863876342773 seconds
Min Quick Sort Time: 0.009169101715087890 seconds
Min Radix Sort Time: 0.013146400451660156 seconds
Min Counting Sort Time: 0.010874986648559570 seconds
```

결론 - 국내주식, 1000번 반복

```
Average Merge Sort Time: 0.013671107530593873 seconds
Average Quick Sort Time: 0.007370582103729248 seconds
Average Radix Sort Time: 0.015186766386032105 seconds
Average Counting Sort Time: 0.012467087984085082 seconds


Max Merge Sort Time: 0.04915261268615723 seconds
Max Quick Sort Time: 0.04102134704589844 seconds
Max Radix Sort Time: 0.09124922752380371 seconds
Max Counting Sort Time: 0.04228925704956055 seconds


Min Merge Sort Time: 0.01027369499206543 seconds
Min Quick Sort Time: 0.005987405776977539 seconds
Min Radix Sort Time: 0.011082410812377930 seconds
Min Counting Sort Time: 0.009035587310791016 seconds
```

결론

해외주식: radix > quick > counting > merge

국내주식: radix > quick > counting > merge

결론

해외주식: quick > counting > radix > merge

국내주식: quick > counting > merge > radix

감사합니다

LIGHT 조