정렬의 두 가지 아이디어

- Yo-yo Sort 와 Deduping Sticker -

작성자 : 김태형
최초 작성 일자 : 2022.11.28.
최종 작성 일자 : 2022.12.04.

작성 기록
1. 2022.11.28. : 2가지 아이디어 작성
2. 2022.11.29. : 참고 -> 참조, 사용한 Quick Sort 알고리즘 제시, 출처 명시
3. 2022.11.30. : C# 알고리즘 제작
4. 2022.12.01. : 출처 명시, 더 자세한 실험 계획 구성
5. 2022.12.02. : 실험 계획 작성
6. 2022.12.03. : 실험 결과 작성
7. 2022.12.04. : 최종 작성

1. 소개

1. 정보

정렬 알고리즘과 관련된 추가적인 제시와 새로운 정렬 알고리즘에 대한 내용을 포함하고 있다. 새 정렬의 이름은 요요 정렬 (yo-yo sort)이고, 새 추가적인 제시의 이름은 중복제거 스티커 (deduping sticker)이다. 최초 실험은 Python 한 언어로 진행되었으며, 최초 실험 일자는 2022.11.28.이다. Python은 인터프리터 언어이므로 컴파일러 언어에 비해 느리다. 그러므로 추후에 C#으로 새 실험을 진행하였다.

Python [3.10.5]

C# [.NET Core 3.1 C# 8.0]

※ N : 자료의 개수, 자료 : 정렬하고자 하는 자료

2. 실험 방식

인터프리터 언어인 Python과 컴파일러 언어인 C#으로 실험이 진행되며, 정렬, 자료 구조, 자료의 개수와 범위, 사용된 변수의 양을 비교한다.
비교될 정렬은 각각 Counting Sort, Quick Sort, Yo-yo sort이다.
비교될 자료 구조는 무작위 값, 오름차순으로 정렬된 값, 내림차순으로 정렬된 값이다.
공간복잡도와 시간복잡도를 비교한다.

2. 목차

3. 비교 소스코드

1. Python

1. Counting Sort / 출처 [6]

```
def CountingSort(Array):
        MaxValue = max(Array)
        CountingArray = [0]*(MaxValue +1)
        for i in Array:
                CountingArray[i] +=1
        for i in range(MaxValue):
                CountingArray[i+1] += CountingArray[i]
        OutputArray = [-1]*len(Array)
        for i in Array:
                OutputArray[CountingArray[i] -1] = i
                CountingArray[i] -=1
        return OutputArray
```

2. Quick Sort / 출처 [3]

```
def QuickSort(List):
        if len(List) <=1:
                return List
        Pivot = List[len(List) //2]
        LesserList, EqualList, GreaterList = [], [], []
        for num in List:
                if num < Pivot:
                        LesserList.append(num)
                elif num > Pivot:
                        GreaterList.append(num)
                else:
                        EqualList.append(num)
        return QuickSort(LesserList) + EqualList + QuickSort(GreaterList)
```

2. C#

1. Couting Sort / 출처 [5]

```
private static void CountingSort(int[] array)
{
    var size = array.Length;
    int maxElement =0;
    for (int i =0; i < size; i ++)
    {
        if (array[i] > maxElement)
        {
            maxElement = array[i];
        }
    }
    var occurrences =new int[maxElement +1];
    for (int i =0; i < maxElement +1; i ++)
    {
        occurrences[i] =0;
    }
    for (int i =0; i < size; i ++)
```

```
    {
        occurrences[array[i]]++;
    }
    for (int i =0, j =0; i <= maxElement; i ++)
    {
        while (occurrences[i] >0)
        {
            array[j] = i;
            j++;
            occurrences[i]--;
        }
    }
}
```

2. Quick Sort / 출처 [4]

```
static int ArryDivide(int[] Arry, int left, int right)
{
    int PivotValue, temp;
    int index_L, index_R;
    index_L = left;
    index_R = right;
    //Pivot 값은 0번 인덱스의 값을 가짐
    PivotValue = Arry[left];
    while (index_L < index_R)
    {
        //Pivot 값 보다 작은경우 index_L 증가(이동)
        while ((index_L <= right) && (Arry[index_L] < PivotValue))
            index_L++;
        //Pivot 값 보다 큰경우 index_R 감소(이동)
        while ((index_R >= left) && (Arry[index_R] > PivotValue))
            index_R--;
        //index_L 과 index_R 이 교차되지 않음
        if (index_L < index_R)
        {
            temp = Arry[index_L];
            Arry[index_L] = Arry[index_R];
            Arry[index_R] = temp;
            //같은 값이 존재 할 경우
            if (Arry[index_L] == Arry[index_R])
                index_R--;
        }
    }
    //index_L 과 index_R 이 교차된 경우 반복문을 나와 해당 인덱스값을 리턴
    return index_R;
}
private static void QuickSort(int[] arry, int left, int right)
{
    if (left < right)
    {
        int PivotIndex = ArryDivide(arry, left, right);
        QuickSort(arry, left, PivotIndex -1);
        QuickSort(arry, PivotIndex +1, right);
    }
}
```

4. 요요 정렬 (yo-yo sort)

1. 소개

이 정렬은 시간복잡도가 O(n+k)인 정렬이다. 간단히, Hash Table을 이용해 자료와 개수를 저장한 후, 반복하며 출력하는 방식이다. Hash Table을 사용하여 자료와 자료의 개수를 각각 Key와 Value에 삽입하여 저장한다. for 반복문을 사용하여 자료의 최솟값부터 자료의 최댓값까지 반복하는데, 이때, 반복하는 값이 Hash Table 내에 있다면 그 수의 개수만큼 출력 자료에 추가한다.

2. 사용 범주

이 정렬의 효과적인 사용 범주는 Quick Sort와 비교되어 작성된다.

1-1) 자료의 중복이 많이 될수록 효과적이다.

1-2) 자료의 범위(0 ~ k-1)가 작을수록 효과적이다.

2) 1-1) 또는 1-2) 가 성립할 때, 자료의 개수가 많을수록 효과적이다.

3. 소스코드

Python

```python
def FirstSort(List):
        Min = min(List)
        Max = max(List)
        Dict = {}
        for x in List:
                if Dict.get(x) ==None:
                        Dict[x] =1
                else:
                        Dict[x] +=1

        New = []
        for x in range(Min, Max +1):
                if Dict.get(x) !=None:
                        New.extend([ x for _ in range(Dict[x]) ])
        return New
```

C#

```csharp
private static void YoyoSort(int[] OriginalArray, int Length)
{
    int Min =int.MaxValue;
    for (int i =0; i < Length; i ++)
    {
        if (OriginalArray[i] < Min)
        {
            Min = OriginalArray[i];
        }
    }
    int Max =0;
```

```
    for (int i =0; i < Length; i ++)
    {
        if (OriginalArray[i] > Max)
        {
            Max = OriginalArray[i];
        }
    }
    Dictionary <int, int > CountDict =new Dictionary <int, int >();
    for (int i =0; i < Length; i ++)
    {
        if (CountDict.ContainsKey(i) ==false)
        {
            CountDict.Add(i, 1);
        }
        else
        {
            CountDict[i]++;
        }
    }
    int Index =0;
    for(int i = Min; i < Max +1; i ++)
    {
        if (CountDict.ContainsKey(i) ==true)
        {
            for(int x =0; x < CountDict[i]; x ++)
            {
                OriginalArray[Index] = i;
                Index++;
            }
        }
    }
}
```

4. 소스코드 설명

크게 3가지 범주로 나뉜다.

1) Hash Table에 값 저장

Key에는 자료의 값, Value에는 자료의 개수가 할당된다.

시간복잡도는 O(n)

2) 최솟값과 최댓값 사이 반복

자료의 최솟값과 최댓값 사이 개수인 k(Max - Min + 1)만큼 반복한다.

시간복잡도는 O(k)

3) 값 추가하기

반복하는 숫자가 Hash Table 내에 존재하면, 개수가 담겨있는 Value만큼 반복하는 숫자를 추가한다.

5. 요요 정렬의 의미

자료의 크기가 요요처럼 크게 줄었다 늘어났다 할 수 있기 때문이다.

5. 중복제거 스티커 (deduping sticker)

1. 소개

이 정렬은 자료를 전처리와 후처리를 하여 소요시간을 늦추는 정렬 추가요소다. 간단히, 정렬하기 전, 자료와 개수를 Hash Table에 저장하고 정렬 후, 이를 토대로 자료를 추가하는 방식이다. Hash Table을 사용하여 자료와 자료의 개수를 각각 Key와 Value에 삽입하여 저장한다. 자료의 중복을 제거시키고, 아무 정렬 알고리즘을 사용한다. 정렬된 자료에 따라 저장되어있던 Hash Table의 Key와 Value에 따라 중복된 값을 추가한다.

2. 사용 범주

이 중복제거 스티커는 어떤 알고리즘에도 사용될 수 있으며, 중복이 많을 때, 이를 이용해 전처리 후처리를 완료하면 나은 성능을 기대할 수 있다.

1) 중복이 2번 이상 되는 자료

2) 한 개의 자료가 많이 중복되는 자료

3. 소스코드

Python

```python
def DedupingSticker(List):
        Dict={}
        for x in List:
                if Dict.get(x) ==None:
                        Dict[x] =1
                else:
                        Dict[x] +=1
        List = QuickSort(list(Dict.keys()))
        New = []
        for x in List:
                New.extend([ x for _ in range(Dict[x]) ])
        return New
```

C#

```csharp
private static void DedupingSticker(int[] OriginalArray, int Length)
{
    Dictionary <int, int > CountDict =new Dictionary <int, int >();
    for (int i =0; i < Length; i ++)
    {
        if (CountDict.ContainsKey(OriginalArray[i]) ==false)
        {
            CountDict.Add(OriginalArray[i], 1);
        }
        else
        {
            CountDict[OriginalArray[i]]++;
        }
    }
    int[] SortedValue =new int[CountDict.Count];
    int SortedValueLen = CountDict.Count;
    int z =0;
    foreach(var Entry in CountDict)
```

```
    {
        SortedValue[z] = Entry.Key;
        z++;
    }
    QuickSort(SortedValue, 0, SortedValueLen -1);
    for(int i =0; i < SortedValueLen; i ++)
    {
        OriginalArray[Length - SortedValueLen + i] = SortedValue[i];
    }
    int Index =0;
    for(int i =0; i < SortedValueLen; i ++)
    {
        for(int x =0; x < CountDict[OriginalArray[Length - SortedValueLen + i]]; x ++)
        {
            OriginalArray[Index] = OriginalArray[Length - SortedValueLen + i];
            Index++;
        }
    }
}
```

4. 소스코드 설명

크게 3가지 범주로 나뉜다.

1) Hash Table에 값 저장

Key에는 자료의 값, Value에는 자료의 개수가 할당된다.

시간복잡도는 O(n)

2) 중복 제거시키고 정렬하기

자료를 중복 제거시키고 정렬 알고리즘을 실행시킨다.

Quick Sort의 경우 O(n log n)

3) Hash Table 토대로 값 추가하기

중복제거된 리스트의 값 순서대로 Hash Table을 토대로 값과 값의 개수를 추가시킨다.

5. 중복제거 스티커의 의미

중복제거를 스티커처럼 붙였다 뗐다 할 수 있다.

6. 실험 및 결과

1. 실험 순서 소개

1. 자료 구조

자료 구조는 다음과 같이 정의한다.

1. 무작위 값 (범위 : 10,000, 100,000, 1,000,000 | 개수 : 10,000, 100,000, 1,000,000)
2. 오름차순 (범위 : 10,000, 100,000, 1,000,000 | 오프셋 : -10,000, 0, 10,000)
3. 내림차순 (범위 : 10,000, 100,000, 1,000,000 | 오프셋 : -10,000, 0, 10,000)

2. 변수의 양

단위는 다음과 같이 정의한다.

1. 길이 : 배열의 크기
2. 범위 : 0 ~ (N - 1) 의 범위 내 숫자
3. 오프셋 : 배열 내 추가적인 값

3. 약어

Y : 요요 정렬 (Yo-yo sort)
Q : 퀵 정렬 (Quick Sort)
C : 카운팅 정렬 (Couting Sort)
D : 중복제거 스티커 (Deduping Sticker), 퀵 정렬 (Quick Sort) 사용

4. 시간 측정 소스코드

각 언어의 시간 측정은 다음과 같이 한다.

Python

```
import math
from timeit import default_timer
Start = default_timer()
#정렬
print(f "정렬 약어 {math.floor((default_timer() - Start)*1000)} ms")
```

C#

```
OriginalArray = OriginalList.ToArray();
SW.Start();
YoyoSort(OriginalArray, Length);
SW.Stop();
Console.WriteLine("정렬명 약어 "+ SW.ElapsedMilliseconds +" ms");
SW.Reset();
```

5. 리스트 제작 소스코드

Python

무작위 값

```
OriginalList = [ random.randrange(From, To) for _ in range(N) ]
```

오름차순

```
OriginalList = [ i for i in range(N) ]
```

내림차순
```
OriginalList = [ (N - i -1) for i in range(N) ]
```

C#

무작위 값
```
static List <int > SetListA(int Length, int Range)
{
    List <int > OriginalList =new List <int >();
    Random ran =new Random();
    for (int i =0; i < Length; i ++)
    {
        OriginalList.Add(ran.Next(Range));
    }
    return OriginalList;
}
```

오름차순
```
static List <int > SetListB(int Length, int Offset)
{
    List <int > OriginalList =new List <int >();
    for (int i =0; i < Length; i ++)
    {
        OriginalList.Add(i + Offset);
    }
    return OriginalList;
}
```

내림차순
```
static List <int > SetListC(int Length, int Offset)
{
    List <int > OriginalList =new List <int >();
    for (int i =0; i < Length; i ++)
    {
        OriginalList.Add(Length - i -1 + Offset);
    }
    return OriginalList;
}
```

2. 실험 진행

1. 시간 - 실험

1. 무작위 값 - C#

※ 무작위 값이니, 그 점을 이해하고 결과를 확인해주셨으면 합니다.

길이\범위	10,000	100,000	1,000,000
10,000	Y 3 ms Q 3 ms C 0 ms D 3 ms	Y 6 ms Q 3 ms C 1 ms D 5 ms	Y 34 ms Q 3 ms C 8 ms D 5 ms
100,000	Y 11 ms	Y 27 ms	Y 72 ms

	Q 39 ms C 2 ms D 13 ms	Q 35 ms C 5 ms D 46 ms	Q 31 ms C 13 ms D 55 ms
1,000,000	Y 113 ms Q 1025 ms C 11 ms D 85 ms	Y 169 ms Q 423 ms C 17 ms D 169 ms	Y 351 ms Q 413 ms C 49 ms D 515 ms

2. 오름차순 - C#

길이/오프셋	-10,000	0	10,000
10,000	Y 1 ms Q 195 ms C Error D 198 ms	Y 2 ms Q 189 ms C 0 ms D 165 ms	Y 2 ms Q 192 ms C 0 ms D 180 ms
100,000	Y 12 ms Q Stack overflow. C Error D Stack overflow.	Y 38 ms Q Stack overflow. C 3 ms D Stack overflow.	Y 11 ms Q Stack overflow. C 2 ms D Stack overflow.
1,000,000	Y 142 ms Q Stack overflow. C Error D Stack overflow.	Y 119 ms Q Stack overflow. C 16 ms D Stack overflow.	Y 106 ms Q Stack overflow. C 15 ms D Stack overflow.

3. 내림차순 - C#

길이/오프셋	-10,000	0	10,000
10,000	Y 2 ms Q 189 ms C Error D 168 ms	Y 2 ms Q 203 ms C 0 ms D 173 ms	Y 3 ms Q 210 ms C 0 ms D 187 ms
100,000	Y 14 ms Q Stack overflow. C Error D Stack overflow.	Y 17 ms Q Stack overflow. C 1 ms D Stack overflow.	Y 17 ms Q Stack overflow. C 2 ms D Stack overflow.
1,000,000	Y 142 ms Q Stack overflow. C Error D Stack overflow.	Y 160 ms Q Stack overflow. C 24 ms D Stack overflow.	Y 118 ms Q Stack overflow. C 25 ms D Stack overflow.

==========

1. 무작위 값 - Python

※ 무작위 값이니, 그 점을 이해하고 결과를 확인해주셨으면 합니다.

길이\범위	10,000	100,000	1,000,000
10,000	Y 4 ms Q 14 ms C 3 ms D 17 ms	Y 17 ms Q 18 ms C 16 ms D 24 ms	Y 81 ms Q 22 ms C 118 ms D 29 ms
100,000	Y 25 ms Q 160 ms C 28 ms D 47 ms	Y 59 ms Q 222 ms C 51 ms D 221 ms	Y 163 ms Q 223 ms C 174 ms D 295 ms
1,000,000	Y 291 ms Q 2346 ms C 238 ms D 245 ms	Y 428 ms Q 2845 ms C 408 ms D 679 ms	Y 792 ms Q 3342 ms C 792 ms D 2755 ms

2. 오름차순 - Python

길이\오프셋	-10,000	0	10,000
10,000	Y 6 ms Q 19 ms C Error D 20 ms	Y 7 ms Q 13 ms C 4 ms D 26 ms	Y 6 ms Q 15 ms C 4 ms D 22 ms
100,000	Y 6 ms Q 19 ms C Error D 20 ms	Y 66 ms Q 152 ms C 31 ms D 199 ms	Y 65 ms Q 148 ms C 34 ms D 208 ms
1,000,000	Y 655 ms Q 1708 ms C Error D 2330 ms	Y 667 ms Q 1784 ms C 290 ms D 2425 ms	Y 669 ms Q 1797 ms C 291 ms D 2230 ms

3. 내림차순 - Python

길이\오프셋	-10,000	0	10,000
10,000	Y 6 ms Q 14 ms C Error D 20 ms	Y 6 ms Q 16 ms C 3 ms D 21 ms	Y 6 ms Q 12 ms C 4 ms D 23 ms
100,000	Y 69 ms Q 147 ms C Error D 217 ms	Y 68 ms Q 157 ms C 32 ms D 208 ms	Y 64 ms Q 160 ms C 37 ms D 219 ms
1,000,000	Y 738 ms Q 1908 ms C Error D 2310 ms	Y 636 ms Q 1715 ms C 285 ms D 2438 ms	Y 689 ms Q 1768 ms C 320 ms D 2271 ms

2. 시간 - 결과

Yo-yo Sort : 가장 나은 성능을 보이지는 않지만, Quick Sort 와 Counting Sort 사이에 종종 존재한다. 단, 중복이 적은 상황에서는 좋지 않은 성능을 보인다.
시간복잡도 : O(n + k) : / k : Max - Min + 1

Quick Sort : 퀵 정렬의 최악의 상황인 이미 정렬되어있는 상태의 값에서 측정은 무의미하며, 보통의 경우인 무작위 값에서만 살펴보기로 한다. 무작위 값일 때, 배열 내 자료의 범위에 상관없이 좋은 성능을 보인다.
시간복잡도 : O(n log n)

Counting Sort : 모든 상황에서 괜찮을 결과를 보인다. 단, 중복이 없을 때, 큰 값과 작은 값이 포함되어있으면 성능이 좋지 않게 나온다. 또, 코드에서는 0 이상의 상황만 가정했기에 음수의 비교를 위해서는 추가 수정이 필요하다.
시간복잡도 : O(n + k) / k : Max

Deduping Sticker : 중복이 많은 상황에서만 쓰이면 좋다. 단, 중복이 없으면 시간 낭비가 심하다. 이유는 존재하는 수와 개수를 Hash Table 에 저장하기 때문이다. 따라서, 중복의 파악이 필요하다.
시간복잡도 : O(n) + O(n log n)[Quick Sort]

3. 메모리 - 결과

기본적으로 배열을 담는 데 사용되는 O(n)의 메모리는 작성되지 않았습니다.

Yo-yo Sort : Hash Table을 사용하며, 중복이 많을 경우, 메모리 사용량이 줄어들며 반대의 경우 늘어난다.
최악 : O(n)
최상 : O(k) / k : 중복의 개수

Quick Sort : 최악의 경우 O(n)의 공간복잡도를, 평균적으로 O(log n)만큼의 메모리를 추가적으로 사용한다.
최악 : O(n)
최상 : O(log n)

Counting Sort : 반드시 배열을 최댓값의 개수만큼 생성한다. k 의 범위가 크다면, 메모리 낭비가 크다.
평균 : O(k) / k : Max

Deduping Sticker : Hash Table을 사용하며, 중복이 많을 경우, 메모리 사용량이 줄어들며 반대의 경우 늘어난다. 중복이 제거된 배열이 추가적으로 필요하다.
최악 : O(k + n) / k : 중복의 개수
최상 : O(k + j) / k : 중복의 개수, j : 중복이 제거된 배열

4. 최종 결과

Yo-yo Sort : Couting Sort보다 느리지만, 더 나은 공간복잡도를 가짐.

Quick Sort : 자료의 양에 영향받지만, 자료의 크기에는 영향받지 않음.

Couting Sort : 중복이 많은 경우, 네 알고리즘 중 가장 빠르지만, 큰 공간복잡도를 가짐.

Deduping Sticker : 중복이 많지 않은 경우, 최악의 성능을 가지지만, 가끔 좋은 성능을 보임.

7. 전체 실험 소스코드

Python

```
import random
import math
from timeit import default_timer
import sys
sys.setrecursionlimit(2147483647)
def YoyoSort(List):
        Min = min(List)
        Max = max(List)
        Dict = {}
        for x in List:
                if Dict.get(x) ==None:
                        Dict[x] =1
                else:
                        Dict[x] +=1

        New = []
        for x in range(Min, Max +1):
                if Dict.get(x) !=None:
                        New.extend([ x for _ in range(Dict[x]) ])
        return New
def QuickSort(List):
        if len(List) <=1:
                return List
        Pivot = List[len(List) //2]
        LesserList, EqualList, GreaterList = [], [], []
        for num in List:
                if num < Pivot:
                        LesserList.append(num)
                elif num > Pivot:
                        GreaterList.append(num)
                else:
                        EqualList.append(num)
        return QuickSort(LesserList) + EqualList + QuickSort(GreaterList)
def CountingSort(Array):
        MaxValue = max(Array)
        CoutingArray = [0]*(MaxValue +1)
        for i in Array:
                CoutingArray[i] +=1
        for i in range(MaxValue):
                CoutingArray[i+1] += CoutingArray[i]
        OutputArray = [-1]*len(Array)
        for i in Array:
```

```
                OutputArray[CoutingArray[i] -1] = i
                CoutingArray[i] -=1
        return OutputArray
def DedupingSticker(List):
        Dict={}
        for x in List:
                if Dict.get(x) ==None:
                        Dict[x] =1
                else:
                        Dict[x] +=1
        List = QuickSort(list(Dict.keys()))
        New = []
        for x in List:
                New.extend([ x for _ in range(Dict[x]) ])
        return New
def Run(From : int, To : int, N : int):
        """
        From : 숫자 시작
        To : 숫자 도착 - 1
        N : 리스트 개수
        """
        # OriginalList = [ random.randrange(From, To) for _ in range(N) ]
        # OriginalList = [ i + To for i in range(N) ]
        OriginalList = [ (N - i -1 + To) for i in range(N) ]
        Start = default_timer()
        YoyoSort(OriginalList)
        print(f "Y {math.floor((default_timer() - Start)*1000)} ms")
        Start = default_timer()
        QuickSort(OriginalList)
        print(f "Q {math.floor((default_timer() - Start)*1000)} ms")
        Start = default_timer()
        CountingSort(OriginalList)
        print(f "C {math.floor((default_timer() - Start)*1000)} ms")
        Start = default_timer()
        DedupingSticker(OriginalList)
        print(f "D {math.floor((default_timer() - Start)*1000)} ms")
if __name__ =="__main__":
        Run(From=0, To =10000, N =1000000)
```

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
namespace mysorttest
{
    class Program
    {
        static void Main()
        {
            int Length =1000000; int Range =10000;
            List <int > OriginalList =new List <int >();
            int[] OriginalArray;
```

```
        Stopwatch SW =new Stopwatch();
        //OriginalList = SetListA(Length, Range);
        //OriginalList = SetListB(Length, Range);
        OriginalList = SetListC(Length, Range);
        /*OriginalArray = OriginalList.ToArray();
        SW.Start();
        BubbleSort(OriginalArray, Length);
        SW.Stop();
        Console.WriteLine("BubbleSort " + SW.ElapsedMilliseconds + " ms");
        SW.Reset();*/
        OriginalArray = OriginalList.ToArray();
        SW.Start();
        YoyoSort(OriginalArray, Length);
        SW.Stop();
        Console.WriteLine("Y "+ SW.ElapsedMilliseconds +" ms");
        SW.Reset();

        OriginalArray = OriginalList.ToArray();
        SW.Start();
        //QuickSort(OriginalArray, 0, Length - 1);
        SW.Stop();
        Console.WriteLine("Q "+ SW.ElapsedMilliseconds +" ms");
        SW.Reset();
        OriginalArray = OriginalList.ToArray();
        SW.Start();
        CountingSort(OriginalArray);
        SW.Stop();
        Console.WriteLine("C "+ SW.ElapsedMilliseconds +" ms");
        SW.Reset();
        OriginalArray = OriginalList.ToArray();
        SW.Start();
        //DedupingSticker(OriginalArray, Length);
        SW.Stop();
        Console.WriteLine("D "+ SW.ElapsedMilliseconds +" ms");
        SW.Reset();
        /*for(int i = 0; i < 100; i++)
        {
            Console.Write(OriginalArray[i * Length / 100] + " ");
        }*/
    }
    static List <int > SetListA(int Length, int Range)
    {
        List <int > OriginalList =new List <int >();
        Random ran =new Random();
        for (int i =0; i < Length; i ++)
        {
            OriginalList.Add(ran.Next(Range));
        }
        return OriginalList;
    }
    static List <int > SetListB(int Length, int Offset)
    {
        List <int > OriginalList =new List <int >();
        for (int i =0; i < Length; i ++)
```

```
        {
            OriginalList.Add(i + Offset);
        }
        return OriginalList;
    }
    static List <int > SetListC(int Length, int Offset)
    {
        List <int > OriginalList =new List <int >();
        for (int i =0; i < Length; i ++)
        {
            OriginalList.Add(Length - i -1 + Offset);
        }
        return OriginalList;
    }
    private static void CountingSort(int[] array)
    {
        var size = array.Length;
        int maxElement =0;
        for (int i =0; i < size; i ++)
        {
            if (array[i] > maxElement)
            {
                maxElement = array[i];
            }
        }
        var occurrences =new int[maxElement +1];
        for (int i =0; i < maxElement +1; i ++)
        {
            occurrences[i] =0;
        }
        for (int i =0; i < size; i ++)
        {
            occurrences[array[i]]++;
        }
        for (int i =0, j =0; i <= maxElement; i ++)
        {
            while (occurrences[i] >0)
            {
                array[j] = i;
                j++;
                occurrences[i]--;
            }
        }
    }
    private static void DedupingSticker(int[] OriginalArray, int Length)
    {
        Dictionary <int, int > CountDict =new Dictionary <int, int >();
        for (int i =0; i < Length; i ++)
        {
            if (CountDict.ContainsKey(OriginalArray[i]) ==false)
            {
                CountDict.Add(OriginalArray[i], 1);
            }
            else
```

```
                {
                    CountDict[OriginalArray[i]]++;
                }
            }
            int[] SortedValue =new int[CountDict.Count];
            int SortedValueLen = CountDict.Count;
            int z =0;
            foreach(var Entry in CountDict)
            {
                SortedValue[z] = Entry.Key;
                z++;
            }
            QuickSort(SortedValue, 0, SortedValueLen -1);
            for(int i =0; i < SortedValueLen; i ++)
            {
                OriginalArray[Length - SortedValueLen + i] = SortedValue[i];
            }
            int Index =0;
            for(int i =0; i < SortedValueLen; i ++)
            {
                for(int x =0; x < CountDict[OriginalArray[Length - SortedValueLen +
i]]; x ++)
                {
                    OriginalArray[Index] = OriginalArray[Length - SortedValueLen + i];
                    Index++;
                }
            }
        }
        private static void YoyoSort(int[] OriginalArray, int Length)
        {
            int Min =int.MaxValue;
            for (int i =0; i < Length; i ++)
            {
                if (OriginalArray[i] < Min)
                {
                    Min = OriginalArray[i];
                }
            }
            int Max =0;
            for (int i =0; i < Length; i ++)
            {
                if (OriginalArray[i] > Max)
                {
                    Max = OriginalArray[i];
                }
            }
            Dictionary <int, int > CountDict =new Dictionary <int, int >();
            for (int i =0; i < Length; i ++)
            {
                if (CountDict.ContainsKey(OriginalArray[i]) ==false)
                {
                    CountDict.Add(OriginalArray[i], 1);
                }
                else
```

```
            {
                CountDict[OriginalArray[i]]++;
            }
        }
        int Index =0;
        for(int i = Min; i < Max +1; i ++)
        {
            if (CountDict.ContainsKey(i) ==true)
            {
                for(int x =0; x < CountDict[i]; x ++)
                {
                    OriginalArray[Index] = i;
                    Index++;
                }
            }
        }
    }
    private static void BubbleSort(int[] OriginalArray, int Length)
    {
        for (int x =0; x < Length; x ++)
        {
            for (int y =0; y < Length; y ++)
            {
                if (OriginalArray[x] > OriginalArray[y])
                {
                    int temp = OriginalArray[x];
                    OriginalArray[x] = OriginalArray[y];
                    OriginalArray[y] = temp;
                }
            }
        }
    }
    static int ArryDivide(int[] Arry, int left, int right)
    {
        int PivotValue, temp;
        int index_L, index_R;
        index_L = left;
        index_R = right;
        //Pivot 값은 0번 인덱스의 값을 가짐
        PivotValue = Arry[left];
        while (index_L < index_R)
        {
            //Pivot 값 보다 작은경우 index_L 증가(이동)
            while ((index_L <= right) && (Arry[index_L] < PivotValue))
                index_L++;
            //Pivot 값 보다 큰경우 index_R 감소(이동)
            while ((index_R >= left) && (Arry[index_R] > PivotValue))
                index_R--;
            //index_L 과 index_R 이 교차되지 않음
            if (index_L < index_R)
            {
                temp = Arry[index_L];
                Arry[index_L] = Arry[index_R];
                Arry[index_R] = temp;
```

```
                    //같은 값이 존재 할 경우
                    if (Arry[index_L] == Arry[index_R])
                        index_R--;
                }
            }
            //index_L 과 index_R 이 교차된 경우 반복문을 나와 해당 인덱스값을 리턴
            return index_R;
        }
        private static void QuickSort(int[] arry, int left, int right)
        {
            if (left < right)
            {
                int PivotIndex = ArryDivide(arry, left, right);
                QuickSort(arry, left, PivotIndex -1);
                QuickSort(arry, PivotIndex +1, right);
            }
        }
    }
}
```

8. 자료 출처

[1] 위키백과 - 퀵 정렬 / 시간복잡도, 공간복잡도 부분 2022.11.28.
https://ko.wikipedia.org/wiki/%ED%80%B5_%EC%A0%95%EB%A0%AC
[2] 구글 번역기 - 요요, 중복제거 스티커 -> yo-yo, deduping sticker 2022.11.28.
https://translate.google.co.kr/
[3] [알고리즘] 퀵 정렬 - Quick Sort (Python, Java) / Python 퀵 정렬 알고리즘 부분 2022.11.28.
https://www.daleseo.com/sort-quick/
[4] [알고리즘] 퀵 정렬 (Quick Sort) / C# 퀵 정렬 알고리즘 부분 2022.12.01.
https://dnmaxi.tistory.com/26

[5] Counting Sort in C# / C# 카운팅 정렬 알고리즘 부분 2022.12.01.
https://code-maze.com/counting-sort-in-c/
[6] 카운팅 정렬(counsort) - 정렬 알고리즘, 파이썬 / Python 카운팅 정렬 알고리즘 부분 2022.12.01.
https://elrion018.tistory.com/37
[7] 위키백과 - 계수정렬 / 시간복잡도, 공간복잡도 부분 2022.12.01.
https://ko.wikipedia.org/wiki/%EA%B3%84%EC%88%98_%EC%A0%95%EB%A0%AC