

Radix Sort (기수 정렬)

1. 기수 정렬이란?

기수 정렬(Radix Sort)는 자릿수를 기준으로 차례대로 데이터를 정렬하는 알고리즘입니다. 각 데이터를 자릿수를 기준으로 분류하므로 가장 큰 자릿수를 D 라고 했을 때 $O(DN)$ 의 시간 복잡도를 가집니다.

기수정렬은 낮은 자리수부터 비교하여 정렬해 간다는 것을 기본 개념으로 하는 정렬 알고리즘입니다. 기수정렬은 비교 연산을 하지 않으며 정렬 속도가 빠르지만 데이터 전체 크기에 기수 테이블의 크기만한 메모리가 더 필요합니다.

원소	7	84	25	341	65	30	34			
결과 배열										
자릿수 배열	0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0

	자릿수: 1의 자리									
원소	7	84	25	341	65	30	34			
결과 배열										
자릿수 배열	0	1	2	3	4	5	6	7	8	9
	1	1	0	0	2	2	0	1	0	0

기수정렬과 비슷

기수 정렬

자릿수: 1의 자리

원소	7	84	25	341	65	30	34
----	---	----	----	-----	----	----	----

결과 배열							
-------	--	--	--	--	--	--	--

자릿수 배열 (누적)	0	1	2	3	4	5	6	7	8	9
	1	2	0	0	4	6	0	7	0	0

영수치를 구한다
therefore
처음 포함 밑에 수 다함
ex) 5 Index
Size: 2개
 $\Rightarrow (0)1 + (1)1 + (4)2 + (5)2$
 $\therefore 6$
size 4
Index

기수 정렬

자릿수: 1의 자리

전행 순서

원소	7	84	25	341	65	30	34
----	---	----	----	-----	----	----	----

Idx -- 한 칸씩 배치

결과 배열				34			
-------	--	--	--	----	--	--	--

자릿수 배열 (누적)	0	1	2	3	4	5	6	7	8	9
	1	2	0	0	3	6	0	7	0	0

자릿수: 1의 자리

원소	7	84	25	341	65	30	34
----	---	----	----	-----	----	----	----

결과 배열	30	341	84	34	25	65	7
-------	----	-----	----	----	----	----	---

1의자리까지 정렬 완료!

자릿수 배열 (누적)	0	1	2	3	4	5	6	7	8	9
	0	1	0	0	2	4	0	6	0	0

```

#define _CRT_SECURE_NO_WARNINGS
#include "stdio.h"
#include "stdlib.h"
#include <vcruntime_string.h>

#define MAX 10000

void RadixSort(int *pDataSet)
{
    int nSize = _msize(pDataSet) / sizeof(int);
    int nMaxValue = 0;

    // Step 1 : 최대값을 찾는다.
    for (int i = 0; i < nSize; ++i)
    {
        if (pDataSet[i] >= nMaxValue)
            nMaxValue = pDataSet[i];
    }

    int arrRes[MAX]          ;
    int arrBucket[10]       ; // 0 ~ 9
    int nExp = 1             ; // 자릿수

    // Step 2 : Radix Sort
    while ((nMaxValue / nExp) > 0)
    {
        memset(arrRes, 0, sizeof(int) * MAX);
        memset(arrBucket, 0, sizeof(int) * 10);
        {
            // Step 2.1 : 자릿수 배열 처리

```

```

        for (int i = 0; i < nSize; ++i)
        {
            arrBucket[pDataSet[i] / nExp %
10]++;

        }

        // Step 2.2 : 누적합 계산
        for (int i = 1; i < 10; ++i)
        {
            arrBucket[i] += arrBucket[i -
1];

        }

        // Step 2.3 : 정렬 (같은 자릿수 끼리는 순서
를 유지)

        for (int i = (nSize - 1); i >= 0; --i)
        {
            int nIDX = --arrBucket[pDataSe
t[i] / nExp % 10];

            arrRes[nIDX] = pDataSet[i];

        }

        // Step 3 : 기본 배열 갱신
        for (int i = 0; i < nSize; ++i)
            pDataSet[i] = arrRes[i];

    }

    nExp *= 10;

}

}

```

```
int main(void)
{
    int nSize;
    printf("Size : ");
    scanf("%d", &nSize);

    int *pDataSet = (int*)malloc(sizeof(int) * nSize);
    for (int i = 0; i < nSize; ++i)
    {
        printf("[%d] : ", i + 1);
        scanf("%d", pDataSet + i);
    }
    RadixSort(pDataSet);

    printf("After -----\\n");
    for (int i = 0; i < nSize; ++i)
    {
        printf("[%d] : ", i + 1);
        printf("%d\\n", *(pDataSet + i));
    }

    system("pause");
    return 0;
}
```