2023 Algorithm Study 4Week

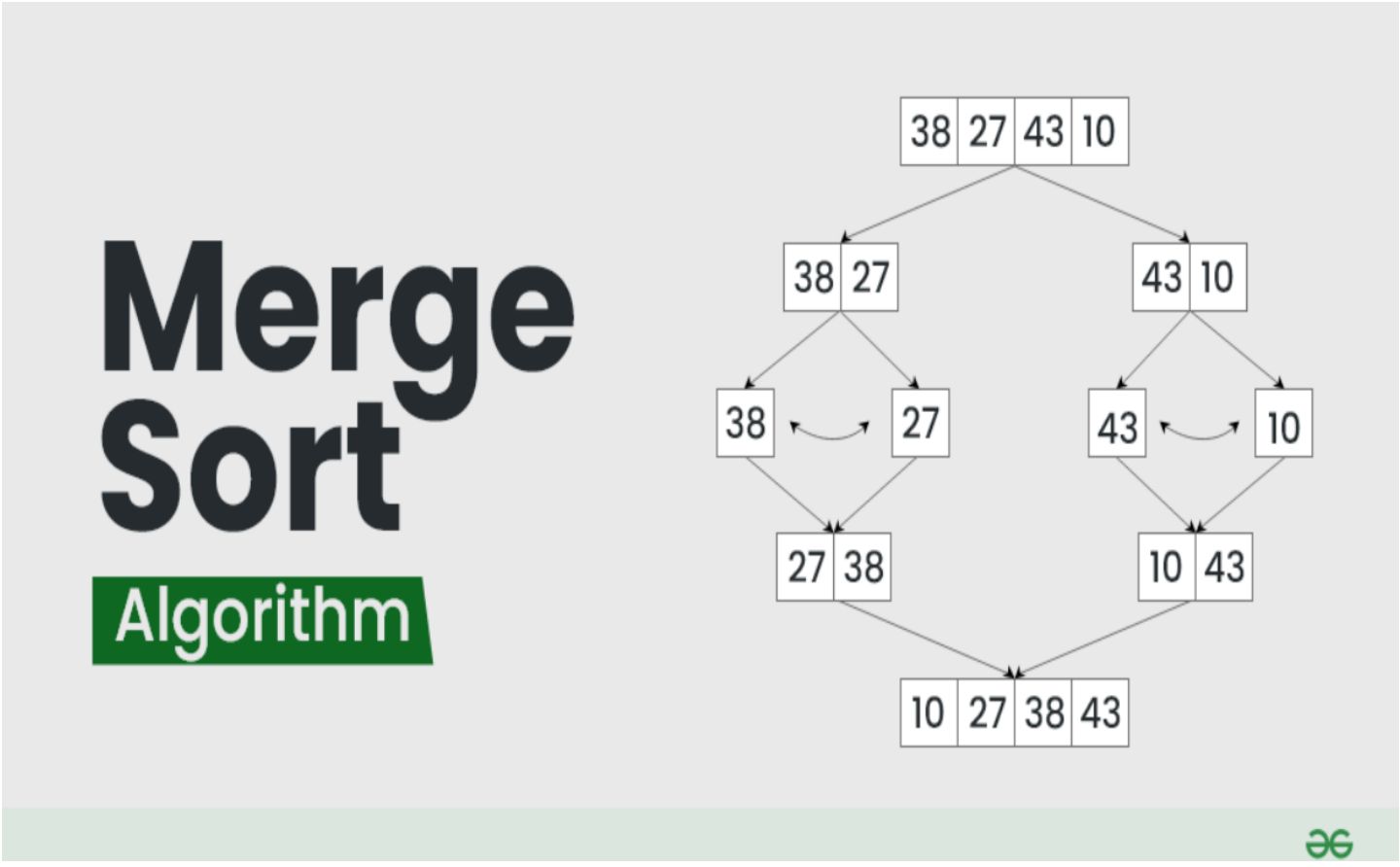
통계 수정 삭제

sookyoung0620 · 방금 전 · 비공개

0

2023 Algorithm Study

목록 보기 5/5



백준 11650

> 이번 문제 주제는 합병 정렬, 퀵 정렬이다!
> 물론 문제를 푸는 다양한 방법이 있겠지만 시험범위의 문제들을 주제별로 풀어보는 게

중요하다고 생각하기 때문에 이번 주차도 역시 그 주제로 풀어보자!

합병 정렬이란?

- Merge sort는 분할정복법을 사용하여 정렬하는 알고리즘
- 순서
  - Step 1 : 처음에는 배열을 두 개의 동일한 절반으로 나눈다.
  - Step 2 : 이제 더 이상 나눌 수 없는 단위 길이의 배열이 되어 항상 단위 길이의 배열이 정렬된다.
  - Step 3 : 정렬된 하위 배열은 서로 병합되어 더 큰 정렬된 하위 배열을 얻는다.
  - Step 4 : 정렬된 하위 배열을 정렬된 배열로 병합한다.

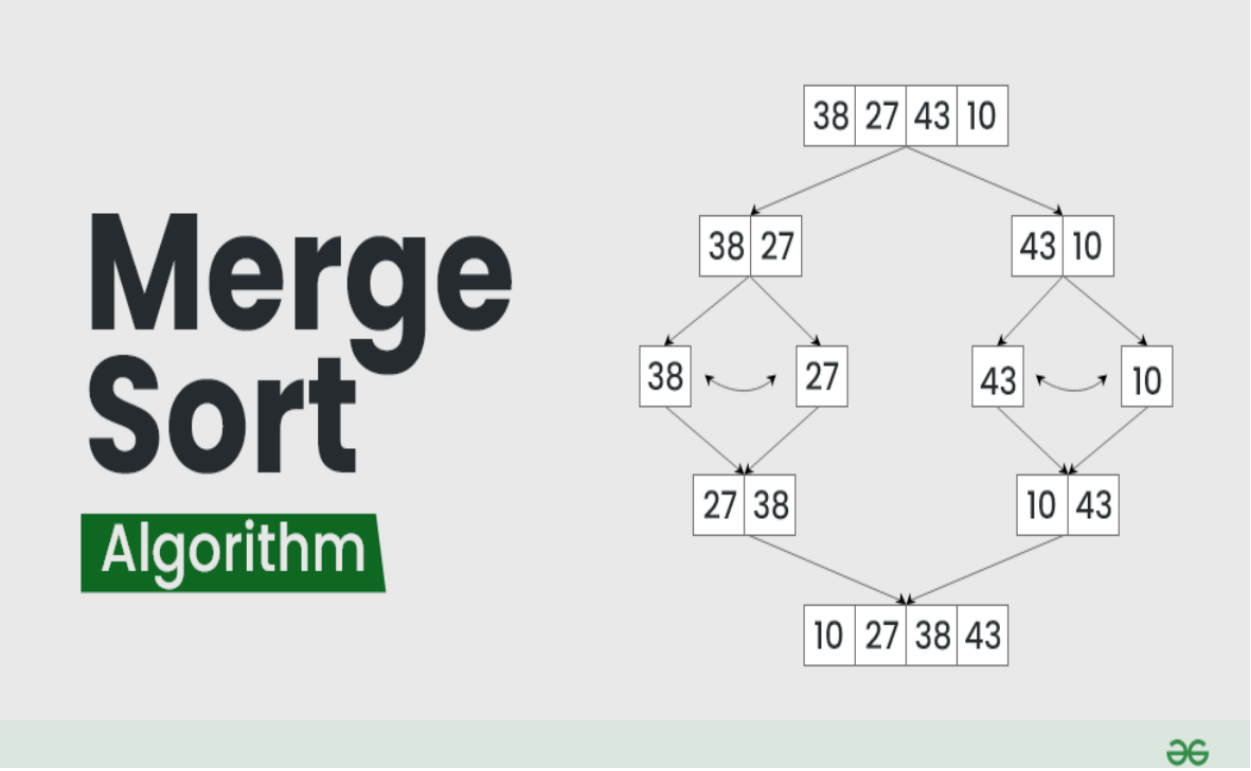


사진 출처 : https://www.geeksforgeeks.org/merge-sort/

```
// 이 코드는 합병정렬 기본코드이다. C program for Merge Sort
#include <stdio.h>
#include <stdlib.h>

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
```

```
        // Create temp arrays
        int L[n1], R[n2];

        // Copy data to temp arrays L[] and R[]
        for (i = 0; i < n1; i++)
            L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
            R[j] = arr[m + 1 + j];

        // Merge the temp arrays back into arr[l..r
        i = 0;
        j = 0;
        k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
            }
            else {
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        // Copy the remaining elements of L[],
        // if there are any
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }

        // Copy the remaining elements of R[],
        // if there are any
        while (j < n2) {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    // l is for left index and r is right index of the
    // sub-array of arr to be sorted
    void mergeSort(int arr[], int l, int r)
    {
        if (l < r) {
            int m = l + (r - l) / 2;

            // Sort first and second halves
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);

            merge(arr, l, m, r);
        }
    }

    // Function to print an array
    void printArray(int A[], int size)
    {
        int i;
```

```
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

// Driver code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

퀵 정렬이란?

- 분할정복(Divide and Conquer) 알고리즘

사진 출처 : https://coding-factory.tistory.com/137

- 이건 사진을 봐도 이해가 잘... 안된다.. 6번부터 이해불가.. 그래서 더 찾아보았다
  - LEFT와 RIGHT가 만나면 (5번 그림) 해당값과 PIVOT값을 비교한다.
  - PIVOT값이 크면 오른쪽에 위치시키고 PIVOT값보다 작으면 왼쪽에 위치시킨다. (이 사진의 경우 PIVOT값이 크기 때문에 6 왼쪽에 PIVOT값 3이 위치한 것이다.)
  - PIVOT값과 6 사이에 있는 값들은 한칸씩 왼쪽으로 땡긴다.
  - 이전 PIVOT값을 기준으로 5번까지 반복한다.

```
#include <stdio.h>
#include <stdlib.h> //랜덤함수 호출

void Swap(int arr[], int a, int b) // a,b 스왑 함수
{
    int temp = arr[a];
    arr[a] = arr[b];
    arr[b] = temp;
}
int Partition(int arr[], int left, int right)
{
    int pivot = arr[left]; // 피벗의 위치는 가장 왼쪽에서 시작
    int low = left + 1;
    int high = right;

    while (low <= high) // 교차되기 전까지 반복한다
    {
        while (low <= right && pivot >= arr[low]) // 피벗보다 큰 값을 찾는 과정
        {
            low++; // low를 오른쪽으로 이동
        }
        while (high >= (left+1)  && pivot <= arr[high]) // 피벗보다 작은 값을 찾는 과정
        {
            high--; // high를 왼쪽으로 이동
        }
        if (low <= high)// 교차되지 않은 상태이면 스왑 과정 실행
        {
            Swap(arr, low, high); //low와 high를 스왑
```

```
            }
        }
        Swap(arr, left, high); // 피벗과 high가 가리키는 대상을 교환
        return high;  // 옮겨진 피벗의 위치정보를 반환

    }


    void QuickSort(int arr[], int left, int right)
    {
        if (left <= right)
        {
            int pivot = Partition(arr, left, right); // 둘로 나누어서
            QuickSort(arr, left, pivot - 1); // 왼쪽 영역을 정렬한다.
            QuickSort(arr, pivot + 1, right); // 오른쪽 영역을 정렬한다.
        }
    }

    int main()
    {
        int n,i;
        int arr[100];


        printf("몇개의 숫자로 정렬하시겠습니까?\n");
        scanf("%d",&n);

        for(i = 0 ; i < n ; i++)
            arr[i]=rand()%1000;

        printf("정렬전 배열 :");
        for(i = 0 ; i < n ; i++)
            printf("%d ", arr[i]);
        printf("\n");

        QuickSort(arr,0,n-1);

        printf("정렬후 배열 :");
        for(i = 0 ; i < n ; i++)
            printf("%d ", arr[i]);
        printf("\n");

        return 0;
    }
```

> **문제를 해결해보자...**
>
> - x축을 x[], y축을 y[]배열에 저장 후 x를 기준으로 인덱스를 움직여줄 때 y[]값도 동일하게 옮겨준다.
> - 인덱스를 옮겨주며 x[] 값이 같을 때 y[]를 확인 후 옮겨줘야하나..? 일단 해보자

> **과정**
>
> - 1st 도전: 기본 코드를 사용하여 x배열 정렬 성공

- y배열을 x배열이 정렬될 때 일단 같이 움직여줘야함..
- 어음 코드 감이 안 와서 용감하게 x,y를 구조체로 방향을 틀어봄

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include <time.h>
#pragma warning (disable:4996)

#define MAX_SIZE 100000
int sorted[MAX_SIZE];



int sorted[MAX_SIZE]; // 추가적인 공간이 필요


// i: 정렬된 왼쪽 리스트에 대한 인덱스
// j: 정렬된 오른쪽 리스트에 대한 인덱스
// k: 정렬될 리스트에 대한 인덱스
/* 2개의 인접한 배열 list[left...mid]와 list[mid+1...right]의 합병 과정
/* (실제로 숫자들이 정렬되는 과정) */
void merge(int list[], int left, int mid, int right) {
    int i, j, k, l;
    i = left;
    j = mid + 1;
    k = left;


    while (i <= mid && j <= right) {
        if (list[i] <= list[j])
            sorted[k++] = list[i++];
        else
            sorted[k++] = list[j++];
    }

    // 남아 있는 값들을 일괄 복사
    if (i > mid) {
        for (l = j; l <= right; l++)
            sorted[k++] = list[l];
    }
    // 남아 있는 값들을 일괄 복사
    else {
        for (l = i; l <= mid; l++)
            sorted[k++] = list[l];
    }

    // 배열 sorted[](임시 배열)의 리스트를 배열 list[]로 재복사
    for (l = left; l <= right; l++) {
        list[l] = sorted[l];
    }
}


// 합병 정렬
void merge_sort(int list[], int left, int right) {
    int mid;
```

```
    if (left < right) {
        mid = (left + right) / 2;// 중간 위치를 계산하여 리스트를 균등 분
            merge_sort(list, left, mid); // 앞쪽 부분 리스트 정렬 -정복
        merge_sort(list, mid + 1, right); // 뒤쪽 부분 리스트 정렬 -정
        merge(list, left, mid, right); // 정렬된 2개의 부분 배열을 합병
    }
}

int main() {
    int N;



    scanf("%d", &N);

    int* x = (int*)malloc(sizeof(int) * N);
    int* y = (int*)malloc(sizeof(int) * N);

    for (int i = 0; i < N; i++) {
        scanf("%d %d", &x[i], &y[i]);

    }



    merge_sort(x, 0, N - 1);

    for (int i = 0; i < N; i++) {
        printf("%d\n", x[i]);
    }

    return 0;
```

- 2nd 도전 : 뭔가 x,y 연결하는 건 성공한 거 같기도..하지만 출력이 안 됨

```
#define MAX_SIZE 100000

typedef struct Point {
    int x, y;
} Point;

int sorted[MAX_SIZE]; // 추가적인 공간이 필요

// i: 정렬된 왼쪽 리스트에 대한 인덱스
// j: 정렬된 오른쪽 리스트에 대한 인덱스
// k: 정렬될 리스트에 대한 인덱스
/* 2개의 인접한 배열 list[left...mid]와 list[mid+1...right]의 합병 과정 */
/* (실제로 숫자들이 정렬되는 과정) */
void merge(Point list[], int left, int mid, int right) {
    int i, j, k, l;
    i = left;
    j = mid + 1;
    k = left;

    /* 분할 정렬된 list의 합병 */
    while (i <= mid && j <= right) {
        if (list[i].x < list[j].x || (list[i].x == list[j].x && list[i].y < list[j].y)) {
            sorted[k++] = i++;
        }
```

```
            else {
                sorted[k++] = j++;
            }
        }

        // 남아 있는 값들을 일괄 복사
        while (i <= mid) {
            sorted[k++] = i++;
        }
        while (j <= right) {
            sorted[k++] = j++;
        }

        // 배열 sorted[](임시 배열)의 리스트를 배열 list[]로 재복사
        for (l = left; l <= right; l++) {
            list[l] = list[sorted[l]];
        }
    }

    // 합병 정렬
    void merge_sort(Point list[], int left, int right) {
        int mid;

        if (left < right) {
            mid = (left + right) / 2; // 중간 위치를 계산하여 리스트를 균등 분할 -분할(Divide)
            merge_sort(list, left, mid); // 앞쪽 부분 리스트 정렬 -정복(Conquer)
            merge_sort(list, mid + 1, right); // 뒤쪽 부분 리스트 정렬 -정복(Conquer)
            merge(list, left, mid, right); // 정렬된 2개의 부분 배열을 합병하는 과정 -결합(Combine)
        }
    }

    int main() {
        int N;

        scanf("%d", &N);

        Point* points = (Point*)malloc(sizeof(Point) * N);

        for (int i = 0; i < N; i++) {
            scanf("%d %d", &points[i].x, &points[i].y);
        }

        merge_sort(points, 0, N - 1);

        for (int i = 0; i < N; i++) {
            printf("%d %d\n", points[i].x, points[i].y);
        }

        free(points);

        return 0;
    }
```

참고로 출력은 이렇다...

```
1 -1
1 -1
1 -1
2 2
3 3
```

- 3rd 도전 : sorted 배열을 포인터로 바꿈.. 배열로 하니까 헷갈리고 생각해보니까 x,y니까 바꿨음

```
 #define MAX_SIZE 100000

typedef struct Point {
    int x, y;
} Point;

Point sorted[MAX_SIZE]; // sorted 배열을 Point 형태로 변경


void merge(Point list[], int left, int mid, int right) {
    int i, j, k;
    i = left;
    j = mid + 1;
    k = left;

    while (i <= mid && j <= right) {
        if (list[i].x < list[j].x || (list[i].x == list[j].x && list[i].y < list[j].y)) {
            sorted[k++] = list[i++];
        }
        else {
            sorted[k++] = list[j++];
        }
    }

    while (i <= mid) {
        sorted[k++] = list[i++];
    }
    while (j <= right) {
        sorted[k++] = list[j++];
    }

    // sorted 배열을 list 배열로 복사
    for (int l = left; l <= right; l++) {
        list[l] = sorted[l];
    }
}

// 합병 정렬
void merge_sort(Point list[], int left, int right) {
    int mid;

    if (left < right) {
        mid = (left + right) / 2;
        merge_sort(list, left, mid);
        merge_sort(list, mid + 1, right);
        merge(list, left, mid, right);
    }
}

int main() {
    int N;

    scanf("%d", &N);

    Point* points = (Point*)malloc(sizeof(Point) * N);

    for (int i = 0; i < N; i++) {
        scanf("%d %d", &points[i].x, &points[i].y);
    }
```

```
    merge_sort(points, 0, N - 1);

    for (int i = 0; i < N; i++) {
        printf("%d %d\n", points[i].x, points[i].y);
    }

    free(points);

    return 0;
}
```

> 문제 푼 후기...
> 솔직히 기본 코드 없이는 못 풀겠다... 기본코드를 좀 외워야겠다

윤수경

이전 포스트
2023 Algorithm Study 3Week

0개의 댓글

댓글을 작성하세요

댓글 작성