알고리즘 과제(02)

201404377 진승언

1. 과제 목표 및 해결 방법

➔ 이번 과제는 병합(합병)정렬과 퀵 정렬을 구현하는 거였다. 병합정렬에는 병합하는데 삽입정렬을 사용해야하고 최대 쪼개지는 개수를 제한하는 기능을 갖게 구현해야한다는 조건이 있었다. 퀵정렬은 pivot기준을 가장 마지막에 있는 값으로 해야하고 임의의 값을 기준으로 정렬 할 수 있는 Randomize_Partition을 따로 구현해야한다는 조건이 있었다.

합병정렬은 분할 정복 방법을 사용하는데 분할 하고 정복(정렬) 그리고 마지막에 결합을 하는 방법이다. 이 과정을 재귀를 이용해 구현했다. 추가 기능중 하나인 쪼개는 개수를 제한하는 것은 정렬 할려는 배열인덱스 개수가 해당 개수보다 커야만 돌아가는 코드를 추가했다. 퀵정렬도 기본적으로는 재귀를 이용해 구현했다.

합병정렬에서 merge를 할 때는 해당 쪼개진 부분배열들을 임시배열에 복사해주고 그 임시배열을 삽입정렬 시킨 다음에 원래의 배열에 원래의 인덱스에 넣어주었다. (임시배열을 사용 안 하면 기존 배열의 다른 값들도 전부 한번에 삽입정렬 되므로 임시배열을 사용하였다.)

퀵정렬에서는 pivot을 기준으로 좌우로 정렬되게 하였다.

둘의 차이점 중 하나는 합병정렬은 1/2로 쪼개지고 정렬되고 퀵정렬은 피벗을 기준으로 쪼개지고 정렬된다는 차이점이 있다.

2. 주요 부분 코드 설명(알고리즘 부분 코드 캡쳐)

```java
//끝까지 쪼개질때까지 sort
public void sort(int[] A, int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2; // 절반기준
        sort(A, left, mid); // 기준 왼쪽 뜨 쪼갬
        sort(A, mid + 1, right); // 기준 오른쪽 뜨 쪼갬
        merge(A, left, mid, right); // 쪼갠배열 합침
    }
}

//해당개수될때까지 sort
public void divided_sort(int[] A, int left, int right, int divided_num) { //뒤에 최대 쪼개는 개수 매개변수 추가
    if (left < right && right-left >= divided_num) {
        int mid = (left + right) / 2; // 절반기준
        divided_sort(A, left, mid, divided_num); // 기준 왼쪽 뜨 쪼갬
        divided_sort(A, mid + 1, right, divided_num); // 기준 오른쪽 뜨 쪼갬
        merge(A, left, mid, right); // 쪼갠배열 합침

        //System.out.println(right-left);    // 이 출력문으로 정해진 개수까지만 쪼개지는지 확인가능
    }
}
```

분할 정복 결합 하는 과정을 재귀로 이용한 알고리즘이다. sort메소드는 1/2로 쪼개질 수 있을 때까지(1개될때까지) 분할하고 분할된 것들을 정렬하고 merge하는 것이다. divided_sort는 divided_num 개수 만큼까지 분할하고 정렬하는건 위와 같다.

```java
//merge
public void merge(int[] arr, int left, int mid, int right) {
    int k = left; // 배열 시작위치
    int merged[] = new int[right + 1]; // left에서 right인덱스 까지 저장할 임시배열을 알맞은 크기로 생성 (right인덱스까지 넣어야하므로 right+1)

    for (k = left; k <= right; k++) { // 임시배열에 쪼개진 두 배열을 그대로 합쳐서 넣어줌(정렬X)
        merged[k] = arr[k];
    }

    insertion_sort(merged); // 두개의 배열을 합친 배열을 삽입정렬해줌(이번과제에서 합병정렬을 삽입정렬을 이용하라 했으므로)

    for (k = left; k <= right; k++) { // 삽입 정렬한 임시배열을 원래 배열에 알맞은 위치에 값을 넣어줌 (정렬한값들을 복사).
        arr[k] = merged[k];
    }
}

// 삽입정렬
void insertion_sort(int[] list) {
    int i, j, key;
    for (i = 1; i < list.length; i++) {
        key = list[i];
        for (j = i - 1; j >= 0 && list[j] > key; j--) {
            list[j + 1] = list[j];
        }
        list[j + 1] = key;
    }
}
```

병합정렬의 merge와 삽입정렬의 알고리즘이다. 일반적인 병합정렬과 다르게 삽입정렬을 하는게 이번과제의 조건이었으므로 쪼개진 배열의 값들을 임시배열에 복사하고 임시배열의 값을 삽입정렬하여 그 값들을 기존배열

의 원래 인덱스에 값을 저장해주었다.(주석참고) 삽입정렬은 저번시간에 했으므로 생략하겠다.

```
public void sort(int[] A, int low , int high) {
    if(low< high) {     //정렬할 범위가 2개이상 데이터라면
    int q = partition(A, low, high);  //피벗을 기준으로 2개의 리스트로 분할( 함수의 반한 값은 피벗의 위치이다)
    sort(A, low , q-1);   //low에서 피벗위치 앞까지 대상으로 순환호출
    sort(A, q+1, high);   //피벗위치부터 high까지를 대상으로 순환호출
    }
}

public void random_sort(int[] A, int low , int high) {
    if(low< high) {     //정렬할 범위가 2개이상 데이터라면
    int q = randomize_partition(A, low, high);  //피벗을 기준으로 2개의 리스트로 분할( 함수의 반한 값은 피벗의 위치이다)
    sort(A, low , q-1);   //low에서 피벗위치 앞까지 대상으로 순환호출
    sort(A, q+1, high);   //피벗위치부터 high까지를 대상으로 순환호출
    }
}
```

퀵소트의 재귀를 이용한 sorting 알고리즘이다. 정렬할 범위가 2개 이상의 데이터일 때까지 피벗을 기준으로 분할하고 sort를 순환 호출한다. (주석참고) Random_sort 메소드는 피벗을 마지막 값이 기준이 아닌 내가 정한 값으로 분할하게 하였다. (난 4번째 인덱스로 정했다.)

```
public int partition(int []A, int low, int high ) {
    int pivot = A[high];  //피벗의 기준을 가장 마지막값으로
    int left = low -1;  //피벗보다 작은값 가리킬 포인터역할
    int right = 0;      //피벗보다 큰값을 가리킬 포인터역할 및 배열처음값 부터 차례대로 불러오는 포인터역할(for문)
    for(right = low; right <= high-1; right++) {
        if(A[right] <= pivot){  //피벗보다 값이 작으면
            left++;             //left 1증가
            swap(A, left, right);  //left와 right를 swap해줌
        }
    }
    swap(A, left+1, high); //left+1과(피벗기준보다 큰 값의 시작인덱스) high를(피벗) swap해줌
    return left + 1;

}

public int randomize_partition(int []A, int low, int high ) {
    int pivot = A[4];     //피벗의 기준을 4로 잡았다.
    int left = low -1;
    int right = 0;
    for(right = low; right <= high-1; right++) {
        if(A[right] <= pivot){
            left++;
            swap(A, left, right);
        }
    }
    swap(A, left+1, high);
    return left + 1;
}
```

퀵소트의 partition 메소드이다. 피벗의 기준을 마지막 값으로 하고 for문으로 첫번째 인덱스부터 마지막값(피벗) 이전까지 찾아가서 피벗보다 작은값

은 왼쪽 left가 가리키는 곳으로 피벗보다 높은 값은 오른쪽 right가 가리키는 곳으로 swap하게 해주는 것을 반복하였다. 그리고 for문이 끝나고 마지막에는 피벗과 left+1(피벗보다 큰값의 첫번째 인덱스)와 swap하게 했다.(주석 참고)

Random_partiton은 위와 똑같고 기준을 4번째 인덱스로 잡았을 뿐이다.

3. 결과(시간 복잡도 포함)

먼저 결과 화면들이다.

<Merge Sort>

차례대로 100개짜리 일반적인 합병정렬, 4개 제한 합병정렬 ,16개 제한 합병정렬 파일출력결과 화면이다.

그 뒤는 1000개 짜리이다.

```
Below is Merge_Sort 100 result
6
1497
2131
2946
2984
3540
5684
8643
8718
9898
11264
11632
12807
13023
13880
15213
15291
17184
19849
19914
20580
24272
25228
25373
29464
31077
32871
```

```
Below is Merge_Sort 100_4 result(최대 쪼개는 개수: 4개)
6
1497
2131
2946
2984
3540
5684
8643
8718
9898
11264
11632
12807
13023
13880
15213
15291
17184
19849
19914
20580
24272
25228
25373
29464
```

```
----------------------------------------------------
Below is Merge_Sort 100_16 result(최대 쪼개는 개수: 16개)
6
1497
2131
2946
2984
3540
5684
8643
8718
9898
11264
11632
12807
13023
13880
15213
15291
17184
19849
19914
20580
24272
25228
25373
```

```
Below is Merge_Sort 1000 result
74
166
235
452
477
516
524
544
605
645
825
1105
1300
1338
1469
1490
1539
1555
1577
2178
2296
2451
2472
2592
2739
2795
2921
```

```
----------------------------------------------------
Below is Merge_Sort 1000_4 result(최대 쪼개는 개수: 4개)
74
166
235
452
477
516
524
544
605
645
825
1105
1300
1338
1469
1490
1539
1555
1577
2178
2296
2451
2472
2592
2739
2795
```

```
Below is Merge_Sort 1000_16 result(최대 쪼개는 개수: 16개)
74
166
235
452
477
516
524
544
605
645
825
1105
1300
1338
1469
1490
1539
1555
1577
2178
2296
2451
2472
2592
2739
2795
```

<Quick Sort>

차례대로 100개짜리 일반적인 퀵소트(마지막값이 피벗기준), 네번째 값이 피벗기준인 파일출력 결과 화면이다.

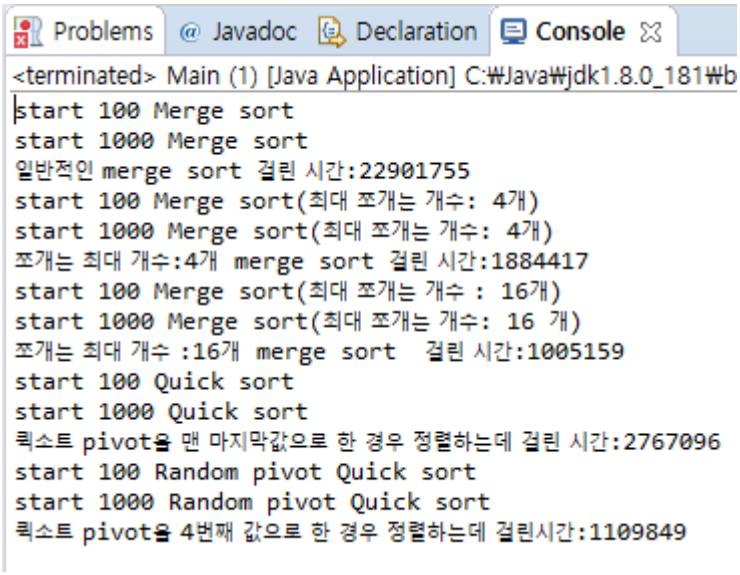
그 뒤는 1000개짜리다.

```
306 ------------------------------------------------
307 Below is Qucik_Sort 100result
308 6
309 1497
310 2131
311 2946
312 2984
313 3540
314 5684
315 8643
316 8718
317 9898
318 11264
319 11632
320 12807
321 13023
322 13880
323 15213
324 15291
325 17184
326 19849
327 19914
328 20580
329 24272
330 25228
331 25373
332 29464
```

```
408 ------------------------------------------------
409 Below is Random pivot Qucik_Sort 100 result
410 6
411 1497
412 2131
413 2946
414 2984
415 3540
416 5684
417 8643
418 8718
419 9898
420 11264
421 11632
422 12807
423 13023
424 13880
425 15213
426 15291
427 17184
428 19849
429 19914
430 20580
431 24272
432 25228
433 25373
434 29464
```

```
3006 ---------------------------------
3007 Below is Qucik_Sort 1000 result
3008 74
3009 166
3010 235
3011 452
3012 477
3013 516
3014 524
3015 544
3016 605
3017 645
3018 825
3019 1105
3020 1300
3021 1338
3022 1469
3023 1490
3024 1539
3025 1555
3026 1577
3027 2178
3028 2296
3029 2451
3030 2472
3031 2592
3032 2739
```

```
4008 ------------------------------------------------
4009 Below is Random pivot Qucik_Sort 1000 result
4010 74
4011 166
4012 235
4013 452
4014 477
4015 516
4016 524
4017 544
4018 605
4019 645
4020 825
4021 1105
4022 1300
4023 1338
4024 1469
4025 1490
4026 1539
4027 1555
4028 1577
4029 2178
4030 2296
4031 2451
4032 2472
4033 2592
4034 2739
4035 2795
```



<기준에 따른 시간차이>

위는 콘솔결과화면이다. System.nanoTime()을 이용해 정렬하는데 시간을 구해봤다.

일반적인 합병정렬보다 4개나 16개로 최대크기를 정해 논 합병정렬이 시간이 더 적게 걸림을 볼 수 있었다. 또한 4개보다 16개가 시간이 더 적게 걸림을 볼 수 있었다.

다음으로 퀵소트는 pivot을 맨 마지막값으로 기준을 한 것보다 privot을 4번째 값으로 한 경우가 더 시간이 적게 걸림을 볼 수 있었다.

` <시간복잡도>

병합정렬은(merge sort)는 데이터가 n개라면 일단 순환호출에서 평균 logn의 패스가 이루어 져야 한다. 그리고 그 각각의 패스에서 merge를 하는데, 부분 배열이 합쳐지는 merge함수에서 평균적으로 값을 2번 복사하는데 이동연산 2n이고 삽입정렬을 이용하므로 n^2의 시간복잡도가 추가로 든다. 그러므로 합하면 2n+n^2이 들으므로 총 O(n^2logn)의 시간복잡도가 된다. 삽입정렬 때문에 병합정렬이 O(nlogn)의 시간복잡도를 낼 수가 없다. 그리고 최대 부분배열 개수를 제한했을 경우는 반복연산이 줄어들기 때문에 시간복잡도가 좀 더 좋아진다.

퀵정렬의 시간복잡도도 데이터가 n개라면 병합정렬처럼 순환호출에서 평균 logn의 패스가 필요하다. 그리고 각각의 패스에서 레코드를 비교하는데 평균 n번의 비교연산이 이루어지므로 비교연산을 총 nlogn번 실행하게되어 O(nlogn)의 복잡도를 가지는 알고리즘 된다.