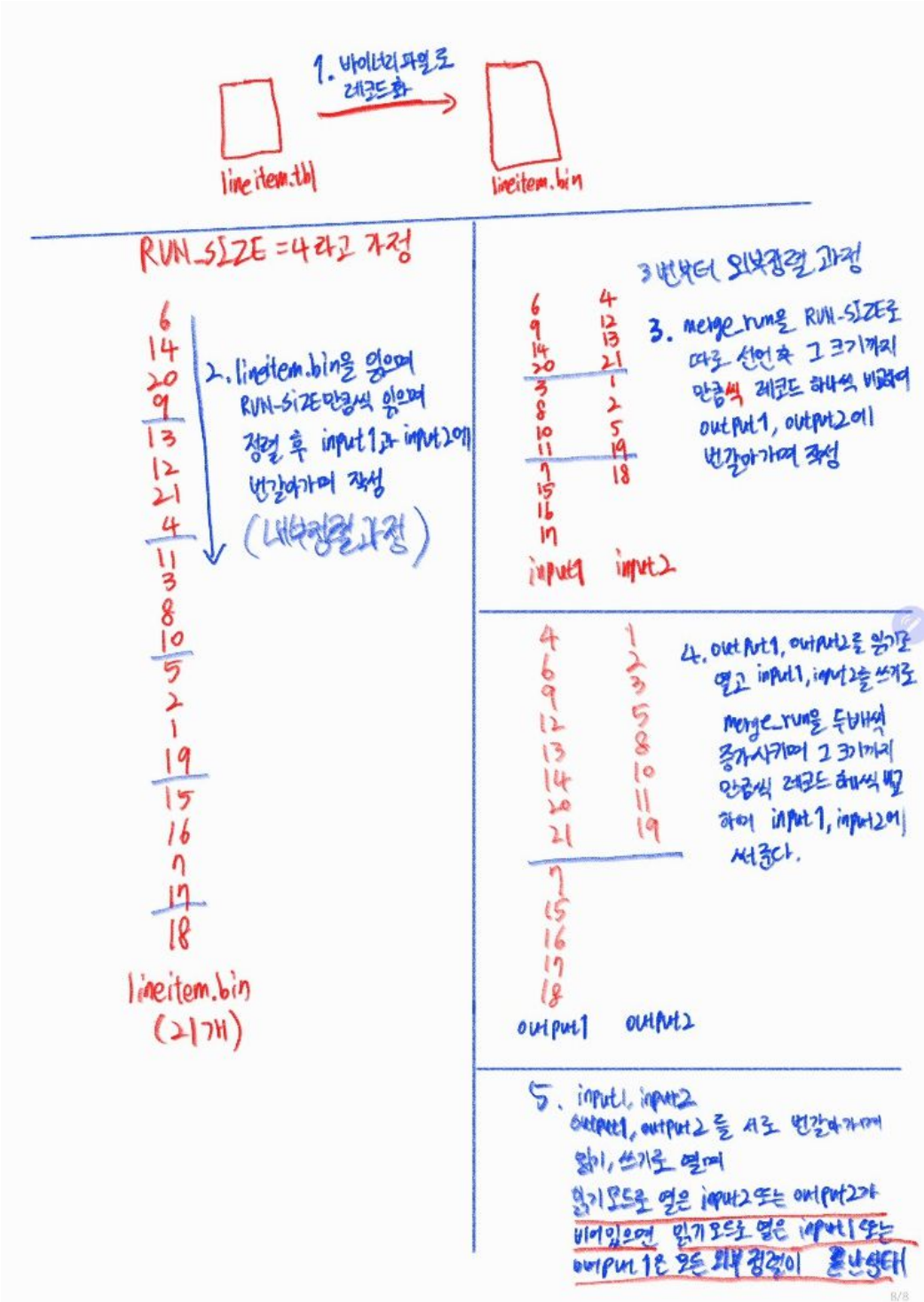
external sort 연산 구현하기

merge.cpp 코드 분석(lineitem.tbl의 레코드 수를 21개로 가정)
2원 균형합병 방식을 사용하여 알고리즘을 설계하였습니다.



위의 각 과정을 코드로 나타낸 부분

1번 과정

```
73          while(fgets(buffer, sizeof(buffer), lineitem_file)) {           // 레코드화 이진파일로 변경
74                  record_parsing(buffer);
75                  fwrite(&record, sizeof(struct RECORD), 1, lineitem_bin_file);   // 구조체만큼 받아서 쓰는 과정
76                  // 파싱한 구조체의 주소에 접근해서 구조체 사이즈만큼 읽어 써주는 과정
77          }
```

2번 과정

```
90          // 내부정렬 과정
91          while(!feof(lineitem_bin_file)) {
92                  size_t x = 0;                                   // x : 구조체 크기를 RUN_SIZE 만큼 읽으라고 했는데 RUN_SIZE보다 작다면
93                                                                  // 딱 그만큼만 정렬하고 입력파일에 써주기 위한 변수
94                  if((x = fread(&records1[0], sizeof(struct RECORD), RUN_SIZE, lineitem_bin_file)) < RUN_SIZE) {
95                          if(x == 0)
96                                  break;
97                          sort(begin(records1), begin(records1) + x);
98                          fwrite(records1, sizeof(struct RECORD), x, input1_file);
99                          break;
100                 }
101                 else {                                                          // 초반에 x는 계속 RUN_SIZE 보다 클 것
102                         sort(begin(records1), begin(records1) + RUN_SIZE);
103                         fwrite(records1, sizeof(records1), 1, input1_file);
104                 }                                               // 2원균형합병으로 입력파일 2개에 분배 과정이기 때문에 이번엔 input2에다가 위 코드
를 반복
105                 if((x = fread(&records2[0], sizeof(struct RECORD), RUN_SIZE, lineitem_bin_file)) < RUN_SIZE) {
106                         if(x == 0)
107                                 break;
108                         sort(begin(records2), begin(records2) + x);
109                         fwrite(records2, sizeof(struct RECORD), x, input2_file);
110                         break;
111                 }
112                 else {
113                         sort(begin(records2), begin(records2) + RUN_SIZE);
114                         fwrite(records2, sizeof(records2), 1, input2_file);
115                 }
116
117                 getrusage(RUSAGE_SELF, &r_usage);                               // 메모리 넘어가는지 체크
118                 if(r_usage.ru_maxrss > MEMORY_SIZE) {
119                         printf("메모리 사용량: %ld KB\n", r_usage.ru_maxrss);
120                         printf("런 크기를 줄이거나 메모리 크기를 늘려서 선언해주세요.\n");
121                         exit(0);                                                // 정해둔 메모리 넘어가면 탈출
122                 }
123         }
```

3번 과정

```
133         // 외부정렬 과정
134         int merge_run = RUN_SIZE;               // 병합하면서 run크기는 2배씩 증가할 예정이기 때문에 따로 정의해둔 상수를 변수로 설정해준다
135         int file_index = 0;                     // file_index : input파일들과 output파일들을 서로 읽기 쓰기 모드를 번갈아가며 하기 위한 체크변수
136
137         while(fread(&record, sizeof(struct RECORD), 1, input2_file) > 0) {              // 입력파일2의 내용이 비어있지 않은동안 반복하는 알고리즘
138                 fseek(input2_file, 0, SEEK_SET);                                        // 위에서 읽어서 확인했기 때문에 다시 0번 위치로 돌아간다.
139                 int index = 1;                  // 쓸 파일을 정해주기 위한 체크변수
140                 int idx1 = 0, idx2 = 0;         // 외부 정렬시 필요한 변수
141                 RECORD a, b;
142                 fread(&a, sizeof(struct RECORD), 1, input1_file);                       // 외부정렬
143                 fread(&b, sizeof(struct RECORD), 1, input2_file);
144                 while(1) {
145                         if(index % 2 == 1)      // index가 홀수 일때는 output1_file에 써주기 위해
146                                 tmp_file = output1_file;
147                         else
148                                 tmp_file = output2_file;
149                         // 이 부분부터는 병합정렬의 알고리즘과 유사합니다.
150                         while(idx1 < merge_run && idx2 < merge_run && !feof(input1_file) && !feof(input2_file)) {
151                                 if(a.PARTKEY < b.PARTKEY) {             // PARTKEY 기준으로 내부정렬 진행했기 때문에 외부정렬도 똑같이 PARTKEY 기>
준
152                                         fwrite(&a, sizeof(struct RECORD), 1, tmp_file);
153                                         fread(&a, sizeof(struct RECORD), 1, input1_file);
154                                         idx1++;
155                                 }
156                                 else {
157                                         fwrite(&b, sizeof(struct RECORD), 1, tmp_file);
158                                         fread(&b, sizeof(struct RECORD), 1, input2_file);
159                                         idx2++;
160                                 }
161                         }
162                         // 하나의 idx가 끝나거나 파일이 끝날 시 남아있는 부분들 처리
163                         while(!feof(input1_file) && idx1 < merge_run) {
164                                 fwrite(&a, sizeof(struct RECORD), 1, tmp_file);
165                                 fread(&a, sizeof(struct RECORD), 1, input1_file);
166                                 idx1++;
167                         }
168                         while(!feof(input2_file) && idx2 < merge_run) {
169                                 fwrite(&b, sizeof(struct RECORD), 1, tmp_file);
170                                 fread(&b, sizeof(struct RECORD), 1, input2_file);
171                                 idx2++;
172                         }
173                         index++;                // index : tmp_file을 뭐로 정할지 결정해주기 위한 체크카운트 변수(쓸 파일을 정해준다)
174                         idx1 = 0, idx2 = 0;     // 블록 크기만큼의 외부정렬을 끝내면 다시 0으로하고 아래쪽의 블록 크기만큼의 외부정렬을 진행
```

4번 과정

```
178             merge_run *= 2;                                 // murge_run 크기는 두배씩 증가
179
180             file_index++;
181
182             if(file_index % 2) {            // file_index가 홀수면 output1을 읽을 파일로 열고
183                     freopen("output1", "rb", input1_file);
184                     freopen("output2", "rb", input2_file);
185                     freopen("input1", "wb", output1_file);
186                     freopen("input2", "wb", output2_file);
187             }
188             else {                          // file_index가 짝수면 input1을 읽을 파일로 연다. 왜냐하면 홀수일때 input1을 쓰기로 열었기 때문
189                     freopen("input1", "rb", input1_file);
190                     freopen("input2", "rb", input2_file);
191                     freopen("output1", "wb", output1_file);
192                     freopen("output2", "wb", output2_file);
193             }
```

5번 과정

```
137         while(fread(&record, sizeof(struct RECORD), 1, input2_file) > 0) {          // 입력파일2의 내용이 비어있지 않은동안 반복하는 알고리즘
138                 fseek(input2_file, 0, SEEK_SET);                                    // 위에서 읽어서 확인했기 때문에 다시 0번 위치로 돌아간다.
```

output.tbl 출력 결과(vi equi_join.tbl)

```
 1 5926723|1|2|2|32.00|28832.00|0.02|0.05|A|F|1992-09-27|1992-09-14|1992-10-23|COLLECT CODFOB|FOB|egular theodolites wake slyly permanen|
 2 5862498|1|2|6|38.00|34238.00|0.09|0.08|A|F|1994-08-29|1994-09-22|1994-09-07|TAKE BACK RRAIL|RAIL|ts sleep carefully carefully pending d|
 3 5618244|1|7502|5|21.00|18921.00|0.02|0.08|N|O|1998-10-17|1998-08-31|1998-10-25|NONE|REG AIR| carefully closely fi|
 4 5362629|1|5002|3|33.00|29733.00|0.00|0.07|R|F|1995-04-06|1995-02-11|1995-04-25|TAKE BACK RAIR|AIR|lar requests haggle furiously quickly|
 5 5352710|1|7502|2|40.00|36040.00|0.01|0.08|A|F|1993-03-31|1993-05-07|1993-04-10|TAKE BACK RAIR|AIR|instructions. slow |
 6 5120486|1|2|1|42.00|37842.00|0.02|0.01|A|F|1992-06-06|1992-03-26|1992-06-12|DELIVER IN SHIP|SHIP| blithely |
 7 4810753|1|5002|5|37.00|33337.00|0.03|0.07|N|O|1996-05-22|1996-06-19|1996-06-02|TAKE BACK RREG AIR|REG AIR|ffily final deposit|
 8 4912929|1|5002|5|30.00|27030.00|0.01|0.06|N|O|1996-03-21|1996-01-31|1996-04-12|TAKE BACK RTRUCK|TRUCK|eodolites sle|
 9 4076803|1|2|4|24.00|21624.00|0.00|0.06|A|F|1993-08-10|1993-08-30|1993-08-28|NONE|SHIP| along the |
10 4343170|1|7502|2|40.00|36040.00|0.09|0.06|N|O|1996-04-11|1996-04-11|1996-04-23|TAKE BACK RSHIP|SHIP|fily final packages boost furiously sil|
11 4422049|1|7502|5|11.00|9911.00|0.03|0.07|R|F|1992-11-14|1992-10-07|1992-12-02|COLLECT CODAIR|AIR|braids hin|
12 4450401|1|7502|1|20.00|18020.00|0.08|0.07|R|F|1995-05-12|1995-05-14|1995-05-18|DELIVER IN MAIL|MAIL| furiously ironic accounts wake|
13 4464900|1|2502|1|16.00|14416.00|0.02|0.06|R|F|1995-04-20|1995-03-27|1995-05-09|TAKE BACK RAIR|AIR| accounts cajole aro|
14 4168578|1|5002|3|23.00|20723.00|0.08|0.03|A|F|1995-04-16|1995-05-04|1995-05-07|COLLECT CODTRUCK|TRUCK|y pending instructions about the furious
   l|
15 3743746|1|5002|6|42.00|37842.00|0.05|0.04|N|O|1996-08-29|1996-11-08|1996-09-13|COLLECT CODSHIP|SHIP|accounts. ironically final requests|
16 3700610|1|5002|2|29.00|26129.00|0.04|0.01|R|F|1995-05-12|1995-05-29|1995-05-14|NONE|AIR|lly unusual deposits impress. doggedly bo|
17 3438019|1|2|6|8.00|7208.00|0.10|0.02|R|F|1992-10-25|1992-10-06|1992-11-02|DELIVER IN RAIL|RAIL|ep blithely ca|
18 2535332|1|5002|4|14.00|12614.00|0.04|0.05|R|F|1994-11-02|1994-10-18|1994-11-20|TAKE BACK RMAIL|MAIL|ns sleep quickly across the regul|
19 2698307|1|2|3|30.00|27030.00|0.06|0.08|N|O|1995-09-30|1995-09-23|1995-10-26|TAKE BACK RRAIL|RAIL|oubt slyly bold pinto beans. carefull|
20 2301222|1|2|5|36.00|32436.00|0.01|0.03|A|F|1994-09-19|1994-11-18|1994-10-06|DELIVER IN SHIP|SHIP|nt deposits use. silent package|
21 2265089|1|2502|4|36.00|32436.00|0.07|0.02|N|O|1997-10-10|1997-10-31|1997-10-27|NONE|RAIL| carefully even ideas cajole |
22 2603011|1|5002|2|41.00|36941.00|0.09|0.02|A|F|1992-08-03|1992-07-28|1992-08-21|DELIVER IN MAIL|MAIL|ts. ironically regu|
23 2485411|1|2502|6|33.00|29733.00|0.08|0.07|A|F|1994-04-13|1994-04-02|1994-05-12|NONE|REG AIR| above the car|
24 1876199|1|2502|3|19.00|17119.00|0.01|0.06|A|F|1992-02-07|1992-03-24|1992-02-16|TAKE BACK RAIR|AIR|s wake carefull|
25 1829732|1|2|3|23.00|20723.00|0.09|0.04|N|O|1997-06-06|1997-07-14|1997-07-06|COLLECT CODRAIL|RAIL|olites are|
26 1695463|1|5002|5|16.00|14416.00|0.06|0.08|N|O|1996-03-10|1996-02-09|1996-03-26|COLLECT CODFOB|FOB|re slyly carefu|
27 1225698|1|2|5|45.00|40545.00|0.02|0.00|N|O|1998-01-07|1997-12-02|1998-01-21|TAKE BACK RFOB|FOB| believe blithely ironic|
28 1139363|1|2|1|16.00|14416.00|0.03|0.08|R|F|1995-01-03|1994-12-30|1995-01-30|COLLECT CODFOB|FOB|cajole carefully: never pending acco|
29 599361|1|5002|7|1.00|901.00|0.05|0.01|N|O|1998-04-28|1998-05-23|1998-05-26|DELIVER IN AIR|AIR|lithely bold packages sleep fluffily. f|
30 545153|1|2|4|15.00|13515.00|0.10|0.07|N|O|1995-10-04|1995-08-21|1995-11-01|DELIVER IN SHIP|SHIP|g requests.|
31 504065|1|2502|6|49.00|44149.00|0.03|0.00|R|F|1992-06-13|1992-07-12|1992-06-29|TAKE BACK RRAIL|RAIL|ckly pending foxes. deposits ha|
32 5879491|2|3|4|25.00|22550.00|0.03|0.06|N|O|1996-10-20|1996-11-12|1996-10-30|COLLECT CODRAIL|RAIL|the final, regular deposi|
33 5399651|2|3|1|38.00|34276.00|0.06|0.02|N|O|1998-03-02|1997-12-24|1998-04-01|NONE|TRUCK|s cajole slyly about the ironic pac|
34 5435686|2|7503|4|48.00|43296.00|0.05|0.06|N|O|1997-02-12|1997-04-17|1997-02-21|NONE|FOB|furiously ironic, b|
35 5240129|2|7503|4|42.00|37884.00|0.06|0.01|N|O|1996-02-10|1996-02-14|1996-03-02|TAKE BACK RRAIL|RAIL| accounts against the slyly pending asympt
```

bool operator를 구조체에 선언하여 임의의 컬러 PARTKEY를 기준으로 정렬이 내부정렬이 되도록 설정하였고 외부정렬에서도 PARTKEY를 기준으로 정렬이 되도록 설정하여 위와 같이 두 번째 PARTKEY를 기준으로 잘 정렬되어 출력된 것을 확인할 수 있다.

merge.cpp 메모리, 시간 분석

메모리와 시간을 출력하기 위한 함수는 join.cpp와 똑같이 사용하였고, MEMORY_SIZE를 따로 선언하여 이 상수값을 넘어가면 프로그램이 종료되도록 설정하였습니다.

```
12 #define MEMORY_SIZE 180000                                    // 메모리 크기
```

```
116             getrusage(RUSAGE_SELF, &r_usage);                          // 메모리 넘어가는지 체크
117             if(r_usage.ru_maxrss > MEMORY_SIZE) {
118                     printf("메모리 사용량: %ld KB\n", r_usage.ru_maxrss);
119                     printf("런 크기를 줄이거나 메모리 크기를 늘려서 선언해주세요.\n");
120                     exit(0);                                           // 정해둔 메모리 넘어가면 탈출
121             }
```

RUN_SIZE가 2000일 경우 시간, 메모리 결과

```
vboxuser@Ubuntu:~/2018136035/dongjin$ ./merge
메모리 사용량: 3892 KB
프로그램 실행 시간: 33.147798 초
```

RUN_SIZE가 20000일 경우 시간, 메모리 결과

```
vboxuser@Ubuntu:~/2018136035/dongjin$ ./merge
메모리 사용량: 11288 KB
프로그램 실행 시간: 29.219644 초
```

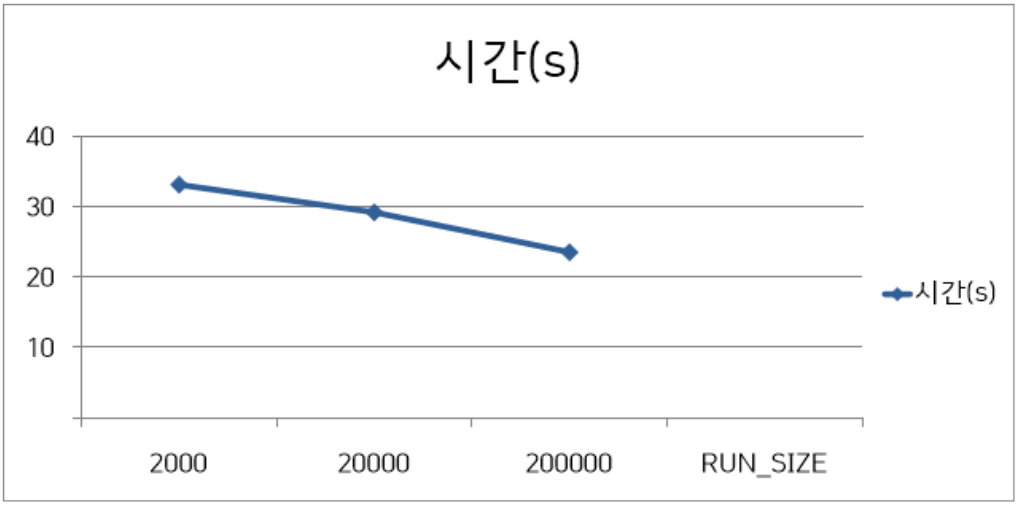
RUN_SIZE가 200000일 경우 시간, 메모리 결과

```
vboxuser@Ubuntu:~/2018136035/dongjin$ ./merge
메모리 사용량: 84512 KB
프로그램 실행 시간: 23.554692 초
```

RUN_SIZE가 500000일 경우

세그먼테이션 오류 발생한다. 따라서 RUN_SIZE가 500000에 근접한 메모리 크기인 180000KB를 초과할 시 프로그램을 종료하도록 설계

```
vboxuser@Ubuntu:~/2018136035/dongjin$ ./merge
메모리 사용량: 206472 KB
런 크기를 줄이거나 메모리 크기를 늘려서 선언해주세요.
```



RUN_SIZE가 커질수록 구조체 배열의 크기가 증가하기 때문에 메모리의 크기는 증가하지만 외부정렬의 파일입출력은 줄어들기 때문에 시간은 감소하는 것을 확인할 수 있다.