알고리즘

정렬 알고리즘 성능평가

소속 : 통계학과
학번 : 21410785
이름 : 황 희

<제목 차례>

1. 서론

가. 정렬 알고리즘에 대한 이해

1) 선택정렬

- 선택 정렬은 첫 번째 자료를 두 번째 자료부터 마지막 자료까지 차례대로 비교하여 가장 작은 값을 찾아 첫 번째에 놓고, 두 번째 자료를 세 번째 자료부터 마지막 자료까지와 차례대로 비교하여 그 중 가장 작은 값을 찾아 두 번째 위치에 놓는 과정을 반복하며 정렬을 수행한다.
- 1회전을 수행하고 나면 가장 작은 값의 자료가 맨 앞에 오게 되므로 그 다음 회전에서는 두 번째 자료를 가지고 비교한다. 마찬가지로 3회전에서는 세 번째 자료를 정렬한다.

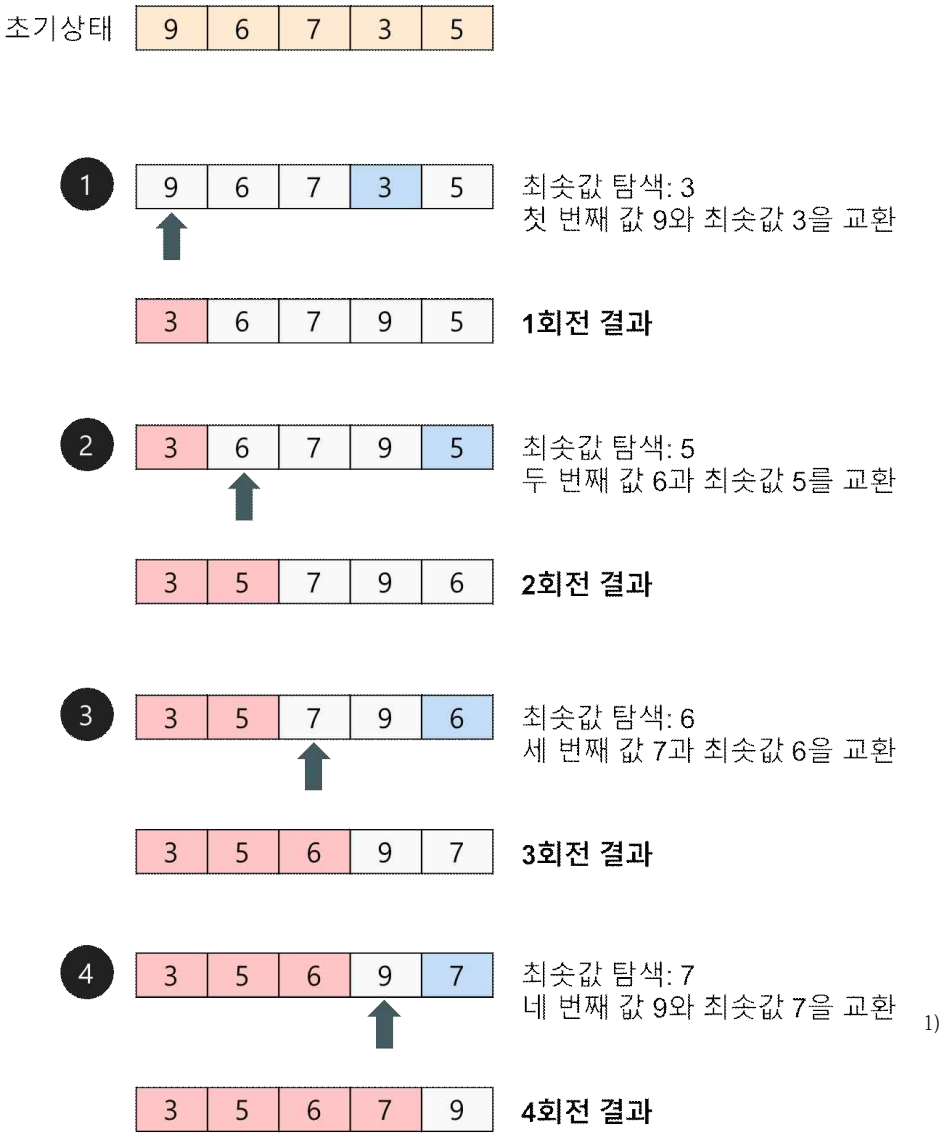

그림 1 선택 정렬 알고리즘 예제

1) 출처 : https://gmlwjd9405.github.io/2018/05/06/algorithm-selection-sort.html

2) 삽입정렬

- 삽입 정렬은 두 번째 자료부터 시작하여 그 앞(왼쪽)의 자료들과 비교하여 삽입할 위치를 지정한 후 자료를 뒤로 옮기고 지정한 자리에 자료를 삽입하여 정렬하는 알고리즘이다.
- 즉, 두 번째 자료는 첫 번째 자료, 세 번째 자료는 두 번째와 첫 번째 자료, 네 번째 자료는 세 번째, 두 번째, 첫 번째 자료와 비교한 후 자료가 삽입될 위치를 찾는다. 자료가 삽입될 위치를 찾았다면 그 위치에 자료를 삽입하기 위해 자료를 한 칸씩 뒤로 이동시킨다.
- 처음 Key 값은 두 번째 자료부터 시작한다.

그림 2 삽입 정렬 알고리즘 예제

3) 퀵 정렬

- 하나의 리스트를 피벗(pivot)을 기준으로 두 개의 비균등한 크기로 분할하고 분할된 부분 리스트를 정렬한 다음, 두 개의 정렬된 부분 리스트를 합하여 전체가 정렬된 리스트가 되게 하는 방법이다.
- 퀵 정렬은 다음의 단계들로 이루어진다.
  - **분할(Divide)** : 입력 배열을 피벗을 기준으로 비균등하게 2개의 부분 배열(피벗을 중심으로 왼쪽: 피벗보다 작은 요소들, 오른쪽: 피벗보다 큰 요소들)로 분할한다.
  - **정복(Conquer)** : 부분 배열을 정렬한다. 부분 배열의 크기가 충분히 작지 않으면 순환 호출 을 이용하여 다시 분할 정복 방법을 적용한다.
  - **결합(Combine)** : 정렬된 부분 배열들을 하나의 배열에 합병한다.
- 순환 호출이 한번 진행될 때마다 최소한 하나의 원소(피벗)는 최종적으로 위치가 정해지므로, 이 알고리즘은 반드시 끝난다는 것을 보장할 수 있다.

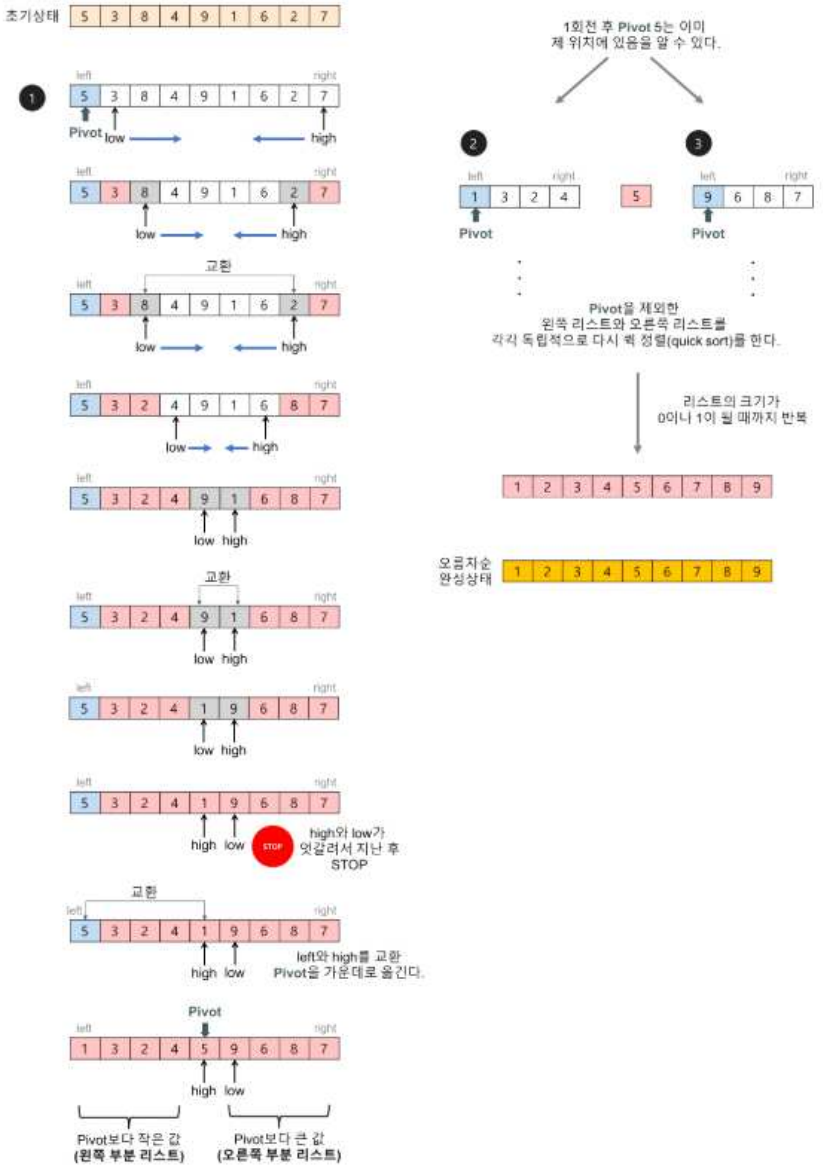

그림 3 퀵 정렬 알고리즘 예제

4) 합병 정렬

- 하나의 리스트를 두 개의 균등한 크기로 분할하고 분할된 부분 리스트를 정렬한 다음, 두 개의 정렬된 부분 리스트를 합하여 전체가 정렬된 리스트가 되게 하는 방법이다.
- 합병 정렬은 다음의 단계들로 이루어진다.
  - **분할(Divide)** : 입력 배열을 같은 크기의 2개의 부분 배열로 분할한다.
  - **정복(Conquer)** : 부분 배열을 정렬한다. 부분 배열의 크기가 충분히 작지 않으면 순환 호출 을 이용하여 다시 분할 정복 방법을 적용한다.
  - **결합(Combine)** : 정렬된 부분 배열들을 하나의 배열에 합병한다.

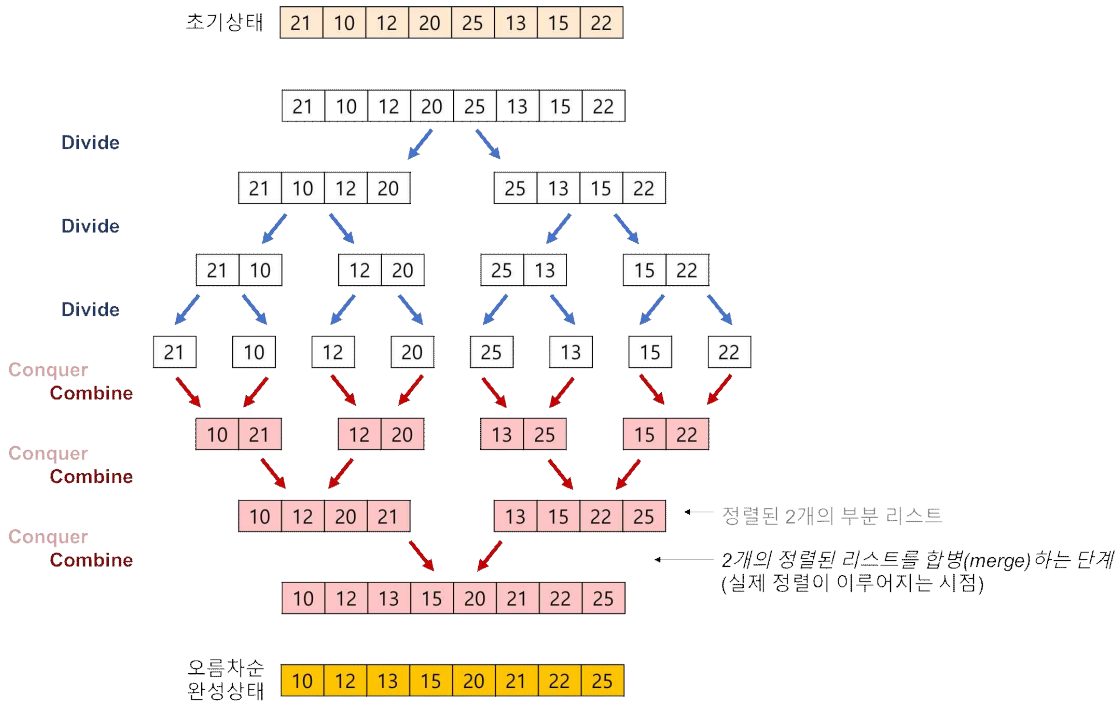

그림 4 합병 정렬 알고리즘 예제

5) 힙 정렬

- 최대 힙 트리나 최소 힙 트리를 구성해 정렬을 하는 방법
- 내림차순 정렬을 위해서는 최대 힙을 구성하고 오름차순 정렬을 위해서는 최소 힙을 구성하면 된다.

나. 알고리즘 성능 평가 방법

- 모든 정렬은 오름차순 정렬을 기준으로 수행됩니다.

1) 오름차순으로 정렬되어있는 배열에 대한 정렬 시간 측정
   - 최선의 시간복잡도
2) 내림차순으로 정렬되어있는 배열에 대한 정렬 시간 측정
   - 최악의 시간복잡도
3) 랜덤으로 생성된 배열에 대한 정렬 시간 측정
   - 랜덤배열을 5회 생성하여 동일한 각 배열에 5가지 정렬 알고리즘을 적용하여 알고리즘 별 정렬 소요 시간 평균을 측정

다. 이론적인 알고리즘 복잡도의 비교

Name	최선	평균	최악	Memory	Stable	기타
Selection sort	n^2	n^2	n^2	1	No	•IS에 비해 Write 연산의 수가 적다(for Flash Memory)
Insertion sort	n	n^2	n^2	1	Yes	•SS에 비해 비교 연산의 수가 적다
Quick sort	$n \log n$	$n \log n$	n^2	1	No	•가장 빠른 내부 정렬 알고리즘으로 알려져 있음
Merge sort	$n \log n$	$n \log n$	$n \log n$	n	Yes	• 외부 정렬에 적합 • 연결 리스트 정렬 지원 • 병렬 처리에 적합
Heap sort	$n \log n$	$n \log n$	$n \log n$	1	No	• Quick sort보다는 늦지만, • 최악의 경우도 $n \log n$ • Memory도 적게 사용

그림 5 내부정렬 알고리즘의 비교

2)

2) 출처 : 수업 PPT 정렬(7장) 33p

2. 본론

가. main 함수 변경점

- 랜덤으로 생성된 배열에 대해 정렬 알고리즘 성능을 평가하기 위해서는 한번 랜덤으로 생성된 배열을 고정시켜 놓고, temp라는 임시 배열을 생성하여 정렬 알고리즘이 교체될 때마다 temp를 처음 생성했던 배열로 초기화시켜주는 작업이 필요 했습니다.
- 그리고 같은 배열에 대한 모든 정렬 알고리즘 소요시간이 동시에 출력하기 위하여 반복문과 case 문을 통해 실행시마다 정렬 함수를 바꿔주는 것이 아닌 한번에 모든 정렬에 대한 성능이 출력되도록 변경하였습니다.
- **아래는 변경된 메인함수에 대한 내용입니다.**

```
int i, j, position, sort_method;
        int sizelist[] = { 0, 100, 200, 300, 400, 500, 600, 700, 800,
                900, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000,
                11000, 12000, 13000, 14000, 15000, 16000 };  // 배열의 크기
        int list[ITERATIONS][MAX_SIZE];
        int temp[MAX_SIZE];

        clock_t start, stop;        // 시작 시간과 종료 시간
        double duration[5][ITERATIONS];          // 경과 시간 = stop ? start
        for (i = 0; i < ITERATIONS; i++) {
                for (j = 0; j < sizelist[i]; j++)
                        list[i][j] = rand() % sizelist[i] + 1; // 랜덤배열 생성
        }

        printf("n\tselection\tinsertion\tquick\t\tmerge\t\theap\n");
        for (sort_method = 0; sort_method < 5; sort_method++) {
                for (i = 0; i < ITERATIONS; i++) {  // 26개의 배열 크기
                        for (j = 0; j < sizelist[i]; j++) // 동일한 배열에 대한 속도비교를 위
해 list -> temp로 초기화
                                temp[j] = list[i][j];

                        switch (sort_method)
                        {
                        default:
                        case 0:
                                start = clock();   // 시작 시간 측정
                                selection_sort(temp, sizelist[i]);
                                stop = clock();   // 종료 시간 측정
                                duration[sort_method][i] = ((double)(stop - start)) /
CLK_TCK;
                                break;
```

```
                        case 1:
                                start = clock();  // 시작 시간 측정
                                insertion_sort(temp, sizelist[i]);
                                stop = clock();  // 종료 시간 측정
                                duration[sort_method][i] = ((double)(stop - start)) /
CLK_TCK;
                                break;
                        case 2:
                                start = clock();  // 시작 시간 측정
                                quick_sort(temp, 0, sizelist[i]);
                                stop = clock();  // 종료 시간 측정
                                duration[sort_method][i] = ((double)(stop - start)) /
CLK_TCK;
                                break;
                        case 3:
                                start = clock();  // 시작 시간 측정
                                merge_sort(temp, sizelist[i]);
                                stop = clock();  // 종료 시간 측정
                                duration[sort_method][i] = ((double)(stop - start)) /
CLK_TCK;
                                break;
                        case 4:
                                start = clock();  // 시작 시간 측정
                                heap_sort(temp, sizelist[i]);
                                stop = clock();  // 종료 시간 측정
                                duration[sort_method][i] = ((double)(stop - start)) /
CLK_TCK;
                                break;
                        }
                }
        }
        for (i = 0; i < ITERATIONS; i++) {
                printf("%d\t", sizelist[i]);
                for (j = 0; j < 5; j++) {
                        printf("%f\t", duration[j][i]);
                }
                printf("\n");
        }
```

- 위 코드 실행 시 출력화면은 다음과 같습니다.

n	selection	insertion	quick	merge	heap
0	0.000000	0.000000	0.000000	0.000000	0.000000
100	0.000000	0.000000	0.000000	0.000000	0.000000
200	0.000000	0.000000	0.000000	0.000000	0.000000
300	0.000000	0.000000	0.000000	0.000000	0.000000
400	0.001000	0.001000	0.000000	0.000000	0.000000
500	0.000000	0.000000	0.000000	0.000000	0.001000
600	0.001000	0.000000	0.000000	0.001000	0.000000
700	0.001000	0.001000	0.000000	0.000000	0.000000
800	0.002000	0.001000	0.000000	0.000000	0.000000
900	0.002000	0.000000	0.001000	0.000000	0.000000
1000	0.002000	0.002000	0.000000	0.000000	0.001000
2000	0.009000	0.004000	0.000000	0.001000	0.000000
3000	0.022000	0.010000	0.001000	0.000000	0.001000
4000	0.037000	0.019000	0.001000	0.001000	0.001000
5000	0.061000	0.029000	0.001000	0.002000	0.002000
6000	0.106000	0.041000	0.002000	0.001000	0.002000
7000	0.156000	0.083000	0.001000	0.002000	0.002000
8000	0.183000	0.096000	0.003000	0.002000	0.002000
9000	0.301000	0.106000	0.002000	0.002000	0.003000
10000	0.509000	0.178000	0.002000	0.003000	0.003000
11000	0.400000	0.217000	0.003000	0.003000	0.004000
12000	0.634000	0.193000	0.003000	0.003000	0.004000
13000	0.712000	0.261000	0.003000	0.003000	0.004000
14000	0.631000	0.248000	0.028000	0.019000	0.004000
15000	0.869000	0.335000	0.004000	0.004000	0.005000
16000	0.748000	0.313000	0.006000	0.004000	0.005000

그림 6 변경된 main함수 출력화면

나. 오름차순 배열에 대한 정렬 알고리즘 성능 평가

1) 알고리즘 별 소요시간 표

n	selection	insertion	quick	merge	heap
0	0	0	0	0	0
100	0	0	0	0	0
200	0	0	0	0	0
300	0	0	0.001	0	0
400	0	0	0	0	0
500	0.001	0	0.001	0	0
600	0.001	0	0.001	0	0
700	0.001	0	0.001	0	0.001
800	0.001	0	0.001	0	0
900	0.002	0	0.002	0	0
1000	0.003	0	0.002	0	0
2000	0.011	0	0.01	0	0.001
3000	0.027	0	0.019	0	0.001
4000	0.052	0	0.034	0	0
5000	0.065	0	0.053	0.001	0.002
6000	0.098	0	0.075	0.001	0.001
7000	0.125	0	0.109	0.001	0.002
8000	0.153	0	0.132	0.001	0.002
9000	0.198	0	0.177	0.002	0.003
10000	0.24	0	0.225	0.001	0.002
11000	0.292	0	0.254	0.002	0.003
12000	0.344	0	0.322	0.001	0.003
13000	0.424	0	0.359	0.003	0.003
14000	0.465	0	0.416	0.002	0.004
15000	0.542	0	0.467	0.002	0.003
16000	0.615	0	0.277	0.002	0.005

표 1 오름차순 배열 정렬 소요시간

2) 알고리즘 별 소요시간 그래프

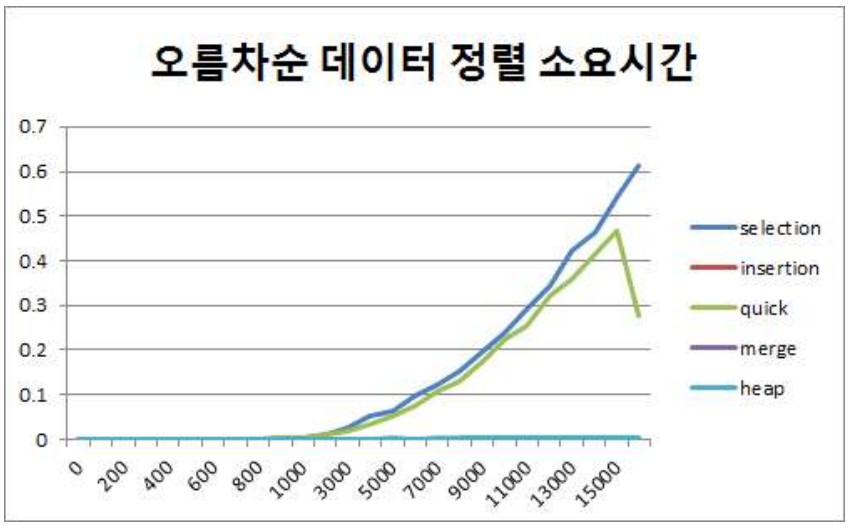

그림 7 오름차순 데이터 정렬소요시간 그래프

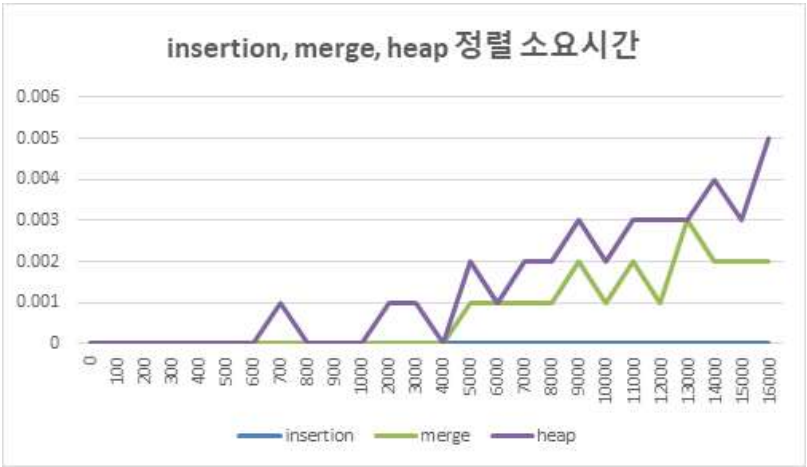

그림 8 오름차순 데이터 insertion, merge, heap 정렬 소요시간 그래프

3) 정리

- **선택 정렬** : 모든 데이터 수 구간에서 가장 높은 정렬 소요시간을 보인다. 최선의 데이터 배열에서 알고리즘 중 가장 많은 소요시간을 차지하는 것을 알 수 있다.
- **삽입 정렬** : 아래 그래프와 위의 표를 참고하시면 모든 데이터 수에서 소요시간이 0인 것을 확인 할 수 있다. 따라서 최선의 데이터 배열에서 삽입정렬이 가장 효율적임을 알 수 있다.
- **퀵 정렬** : 그래프 모양이 선택정렬과 비슷하여 데이터 수가 늘어날수록 높은 소요시간을 차지하는 것을 알 수 있다. 선택정렬보다는 소요시간이 적다.
- **합병 정렬** : 선택정렬, 퀵 정렬과 비교했을 때는 그래프에 보이지 않을 정도로 최선의 데이터 배열에서 알고리즘 효율이 뛰어난 것을 볼 수 있다.
- **힙 정렬** : 합병 정렬과 마찬가지로 최선의 데이터 배열에서 알고리즘 효율이 뛰어난 것을 볼 수 있다.
- **최선 복잡도 : 선택 > 퀵 >>> 삽입 = 합병 = 힙**

다. 내림차순 배열에 대한 정렬 알고리즘 성능 평가

1) 알고리즘 별 소요시간 표

n	selection	insertion	quick	merge	heap
0	0	0	0	0	0
100	0	0	0.001	0	0
200	0	0	0	0	0
300	0	0	0	0	0
400	0.001	0.001	0.001	0	0
500	0.001	0.001	0.004	0	0
600	0.001	0.001	0.001	0	0
700	0.001	0.002	0.002	0.001	0
800	0.001	0.004	0.003	0	0
900	0.002	0.003	0.003	0	0
1000	0.003	0.004	0.004	0	0
2000	0.065	0.018	0.015	0	0.001
3000	0.02	0.032	0.038	0	0.001
4000	0.062	0.037	0.039	0.001	0.001
5000	0.066	0.057	0.054	0.002	0.001
6000	0.093	0.083	0.081	0.001	0.002
7000	0.112	0.109	0.105	0.001	0.002
8000	0.168	0.146	0.14	0.001	0.003
9000	0.2	0.19	0.176	0.001	0.002
10000	0.244	0.229	0.214	0.01	0.002
11000	0.319	0.288	0.273	0.003	0.003
12000	0.349	0.367	0.323	0.003	0.004
13000	0.444	0.412	0.444	0.003	0.003
14000	0.473	0.479	0.43	0.002	0.003
15000	0.545	0.532	0.487	0.003	0.004
16000	0.624	0.618	0.568	0.003	0.004

표 2 내림차순 배열 정렬 소요시간

2) 알고리즘 별 소요시간 그래프

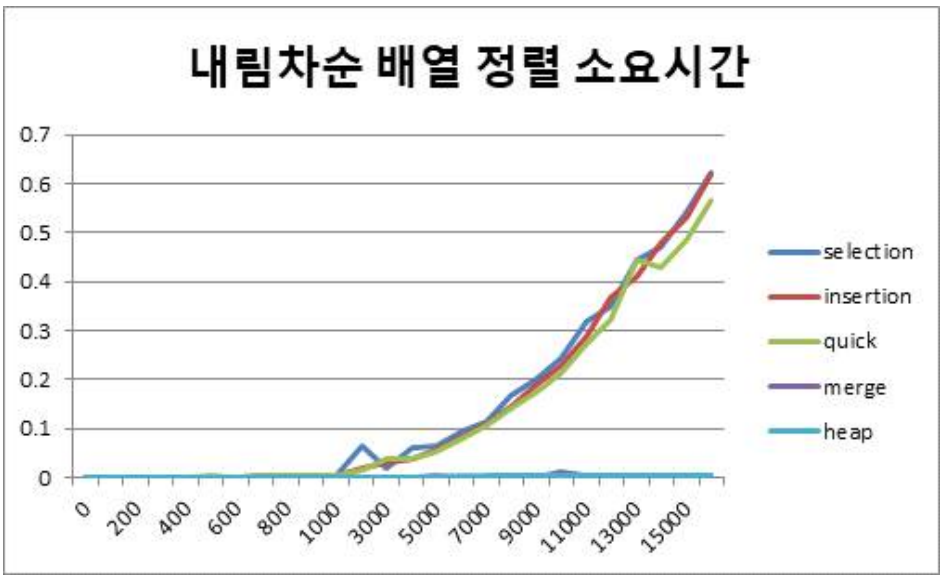

그림 9 내림차순 데이터 정렬소요시간 그래프

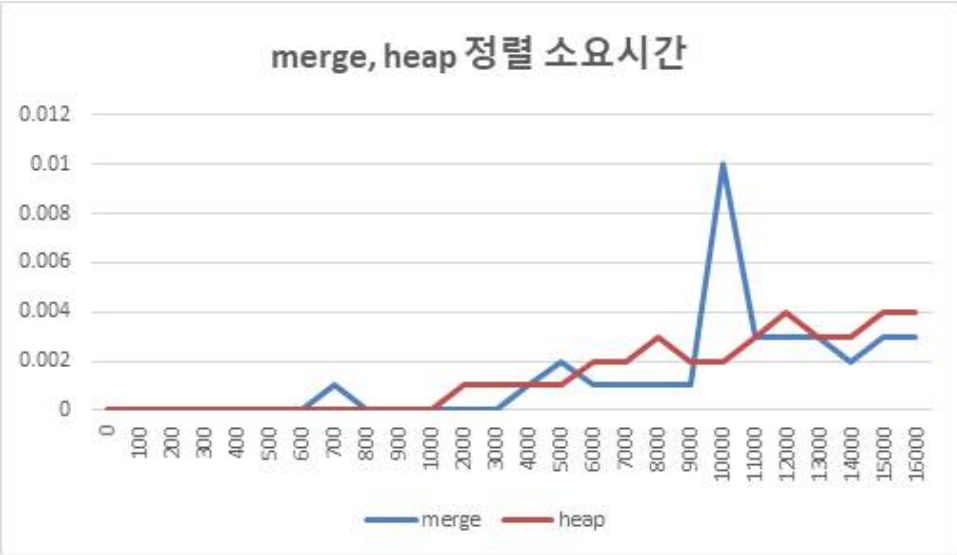

그림 10 내림차순 데이터 merge, heap 정렬 소요시간 그래프

3) 정리

- **선택 정렬** : 최선의 데이터(오름차순)의 정렬 소요시간과 최악의 데이터(내림차순) 정렬 소요시간이 크게 달라진 점이 없다.
- **삽입 정렬** : 최악의 데이터에서의 정렬 소요시간은 선택정렬의 소요시간과 비슷하게 가장 많은 시간을 소모하는 것으로 알 수 있다.
- **퀵 정렬** : 선택정렬, 삽입정렬과 비슷한 소요시간을 가지나 미세한 차이로 소요시간이 적다.
- **합병 정렬** : 선택, 삽입, 퀵 정렬에 비해 아주 적은 시간을 소요함을 알 수 있다.
- **힙 정렬** : 합병정렬과 마찬가지로 0에 가까운 소요시간을 가진다.
- **최악 복잡도 : 선택 = 삽입 = 퀵 >>> 합병 = 힙**

라. 랜덤 데이터에 대한 정렬 알고리즘 성능 평가

★ 우선 랜덤 데이터에 대한 평가는 랜덤 데이터를 5회 생성 후 평가 하였습니다.

★ 이 장에서는 1회차에 대한 표와 그래프만 첨부하였습니다.

1) 알고리즘 별 소요시간 표 (1회차)

n	selection	insertion	quick	merge	heap
0	0	0	0	0.001	0
100	0	0.001	0	0	0
200	0	0	0	0	0
300	0	0	0	0	0
400	0.001	0	0	0	0
500	0	0	0	0	0
600	0.001	0.001	0	0	0
700	0.001	0	0	0	0
800	0.002	0.001	0	0	0
900	0.002	0.001	0	0.001	0.001
1000	0.016	0.001	0.001	0	0
2000	0.024	0.006	0	0	0
3000	0.037	0.028	0.001	0.001	0.001
4000	0.154	0.02	0.001	0.001	0.001
5000	0.121	0.028	0.001	0.001	0.002
6000	0.112	0.04	0.002	0.002	0.011
7000	0.27	0.07	0.001	0.002	0.002
8000	0.427	0.127	0.002	0.002	0.003
9000	0.524	0.094	0.002	0.002	0.003
10000	0.56	0.108	0.003	0.003	0.003
11000	1.014	0.152	0.003	0.003	0.004
12000	0.686	0.243	0.003	0.003	0.004
13000	0.545	0.332	0.004	0.003	0.004
14000	0.606	0.512	0.003	0.005	0.004
15000	0.628	0.439	0.005	0.004	0.005
16000	0.697	0.316	0.005	0.005	0.005

표 3 랜덤 데이터 배열 정렬 소요시간

2) 알고리즘 별 소요시간 그래프

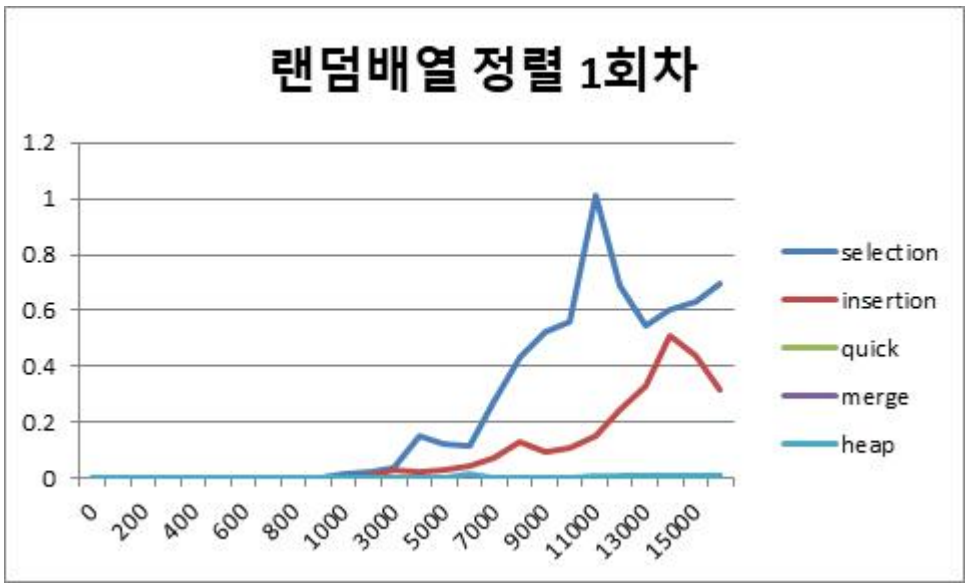

그림 11 랜덤 데이터 정렬 소요시간 그래프

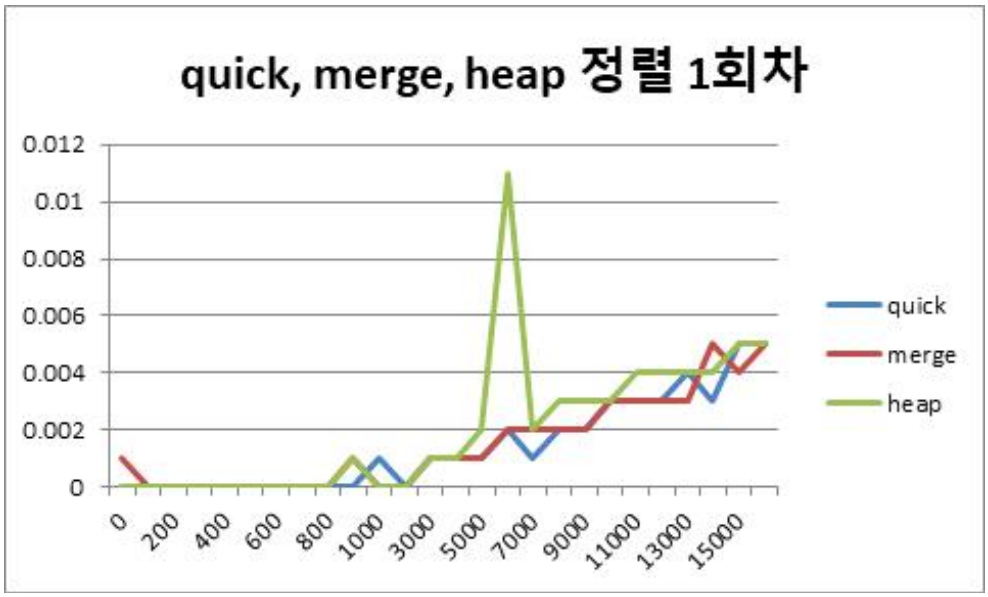

그림 12 랜덤 데이터에 대한 quick, merge, heap 정렬 소요시간 그래프

3) 선택 정렬 5회 평균 및 표준편차 표

n	1회차	2회차	3회차	4회차	5회차	평균	표준편차
0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0
200	0	0	0	0.001	0	0.0002	0.000447
300	0	0	0	0	0	0	0
400	0.001	0.001	0	0	0.001	0.0006	0.000548
500	0	0.001	0.001	0.001	0.001	0.0008	0.000447
600	0.001	0.001	0.001	0.001	0.002	0.0012	0.000447
700	0.001	0.002	0.001	0.001	0.001	0.0012	0.000447
800	0.002	0.002	0.002	0.001	0.001	0.0016	0.000548
900	0.002	0.003	0.002	0.002	0.002	0.0022	0.000447
1000	0.016	0.004	0.002	0.003	0.002	0.0054	0.005983
2000	0.024	0.015	0.01	0.021	0.01	0.016	0.006364
3000	0.037	0.039	0.091	0.028	0.024	0.0438	0.027105
4000	0.154	0.077	0.106	0.119	0.065	0.1042	0.03528
5000	0.121	0.143	0.262	0.072	0.21	0.1616	0.074909
6000	0.112	0.186	0.324	0.161	0.094	0.1754	0.090894
7000	0.27	0.223	0.337	0.152	0.116	0.2196	0.088934
8000	0.427	0.379	0.516	0.177	0.248	0.3494	0.136617
9000	0.524	0.424	0.481	0.261	0.247	0.3874	0.126934
10000	0.56	0.481	0.466	0.298	0.251	0.4112	0.130858
11000	1.014	0.923	0.395	0.439	0.287	0.6116	0.332027
12000	0.686	0.477	0.396	0.478	0.339	0.4752	0.131597
13000	0.545	0.455	0.47	0.875	0.401	0.5492	0.189257
14000	0.606	0.531	0.693	1.122	0.465	0.6834	0.259508
15000	0.628	0.612	0.684	1.113	0.547	0.7168	0.226799
16000	0.697	0.935	0.888	1.22	0.641	0.8762	0.228707

표 4 랜덤 데이터에 대한 선택 정렬

4) 삽입 정렬 5회 평균 및 표준편차 표

n	1회차	2회차	3회차	4회차	5회차	평균	표준편차
0	0	0	0	0	0	0	0
100	0.001	0	0	0	0	0.0002	0.000447
200	0	0	0	0	0	0	0
300	0	0	0	0	0	0	0
400	0	0	0	0.001	0	0.0002	0.000447
500	0	0.001	0	0	0	0.0002	0.000447
600	0.001	0	0.001	0	0.001	0.0006	0.000548
700	0	0.005	0	0.001	0	0.0012	0.002168
800	0.001	0.001	0.001	0.001	0.001	0.001	0
900	0.001	0.001	0.001	0	0.001	0.0008	0.000447
1000	0.001	0.001	0.001	0.002	0.001	0.0012	0.000447
2000	0.006	0.005	0.004	0.004	0.004	0.0046	0.000894
3000	0.028	0.011	0.011	0.01	0.01	0.014	0.007842
4000	0.02	0.044	0.017	0.075	0.018	0.0348	0.025094
5000	0.028	0.028	0.035	0.027	0.028	0.0292	0.003271
6000	0.04	0.084	0.053	0.104	0.04	0.0642	0.028604
7000	0.07	0.058	0.056	0.124	0.058	0.0732	0.028934
8000	0.127	0.137	0.075	0.222	0.088	0.1298	0.057686
9000	0.094	0.327	0.095	0.233	0.088	0.1674	0.108062
10000	0.108	0.25	0.181	0.286	0.106	0.1862	0.081555
11000	0.152	0.165	0.149	0.382	0.135	0.1966	0.104189
12000	0.243	0.167	0.163	0.502	0.157	0.2464	0.147135
13000	0.332	0.21	0.255	0.526	0.277	0.32	0.123242
14000	0.512	0.223	0.253	0.329	0.239	0.3112	0.119391
15000	0.439	0.314	0.267	0.479	0.277	0.3552	0.097392
16000	0.316	0.344	0.397	0.592	0.312	0.3922	0.116731

표 5 랜덤 데이터에 대한 삽입 정렬

5) 퀵 정렬 5회 평균 및 표준편차 표

n	1회차	2회차	3회차	4회차	5회차	평균	표준편차
0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0
200	0	0	0	0	0	0	0
300	0	0	0	0	0	0	0
400	0	0	0	0	0	0	0
500	0	0	0	0	0	0	0
600	0	0	0.001	0	0.001	0.0004	0.000548
700	0	0	0	0.001	0	0.0002	0.000447
800	0	0	0	0	0	0	0
900	0	0.004	0	0	0.001	0.001	0.001732
1000	0.001	0	0	0	0	0.0002	0.000447
2000	0	0.001	0.001	0.001	0	0.0006	0.000548
3000	0.001	0	0	0.001	0.001	0.0006	0.000548
4000	0.001	0.001	0.033	0.002	0.001	0.0076	0.014206
5000	0.001	0.001	0.002	0.001	0.001	0.0012	0.000447
6000	0.002	0.002	0.001	0.003	0.002	0.002	0.000707
7000	0.001	0.001	0.002	0.006	0.002	0.0024	0.002074
8000	0.002	0.001	0.002	0.003	0.003	0.0022	0.000837
9000	0.002	0.002	0.002	0.004	0.002	0.0024	0.000894
10000	0.003	0.002	0.003	0.004	0.002	0.0028	0.000837
11000	0.003	0.003	0.002	0.004	0.003	0.003	0.000707
12000	0.003	0.003	0.003	0.005	0.003	0.0034	0.000894
13000	0.004	0.003	0.004	0.005	0.004	0.004	0.000707
14000	0.003	0.004	0.003	0.006	0.003	0.0038	0.001304
15000	0.005	0.003	0.004	0.006	0.005	0.0046	0.00114
16000	0.005	0.009	0.041	0.009	0.006	0.014	0.015199

표 6 랜덤 데이터에 대한 퀵 정렬

6) 합병 정렬 5회 평균 및 표준편차 표

n	1회차	2회차	3회차	4회차	5회차	평균	표준편차
0	0.001	0	0	0	0	0.0002	0.000447
100	0	0	0	0	0	0	0
200	0	0	0	0	0	0	0
300	0	0	0	0	0	0	0
400	0	0.001	0	0	0	0.0002	0.000447
500	0	0	0	0.001	0.001	0.0004	0.000548
600	0	0	0	0	0	0	0
700	0	0	0	0	0	0	0
800	0	0	0	0.001	0	0.0002	0.000447
900	0.001	0	0.001	0	0	0.0004	0.000548
1000	0	0	0	0	0.001	0.0002	0.000447
2000	0	0	0	0.001	0	0.0002	0.000447
3000	0.001	0.001	0.001	0.005	0.001	0.0018	0.001789
4000	0.001	0.001	0.001	0.002	0.002	0.0014	0.000548
5000	0.001	0.001	0.001	0.002	0.001	0.0012	0.000447
6000	0.002	0.001	0.001	0.002	0.001	0.0014	0.000548
7000	0.002	0.002	0.001	0.003	0.001	0.0018	0.000837
8000	0.002	0.002	0.002	0.003	0.002	0.0022	0.000447
9000	0.002	0.002	0.002	0.003	0.002	0.0022	0.000447
10000	0.003	0.003	0.003	0.004	0.002	0.003	0.000707
11000	0.003	0.003	0.003	0.005	0.003	0.0034	0.000894
12000	0.003	0.003	0.003	0.007	0.003	0.0038	0.001789
13000	0.003	0.003	0.003	0.006	0.004	0.0038	0.001304
14000	0.005	0.004	0.003	0.005	0.005	0.0044	0.000894
15000	0.004	0.004	0.051	0.006	0.004	0.0138	0.020813
16000	0.005	0.004	0.004	0.01	0.005	0.0056	0.00251

표 7 랜덤 데이터에 대한 합병 정렬

7) 힙 정렬 5회 평균 및 표준편차 표

n	1회차	2회차	3회차	4회차	5회차	평균	표준편차
0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0
200	0	0	0	0	0	0	0
300	0	0	0	0	0	0	0
400	0	0	0.001	0	0	0.0002	0.000447
500	0	0.001	0	0	0	0.0002	0.000447
600	0	0	0	0.001	0	0.0002	0.000447
700	0	0	0	0	0.001	0.0002	0.000447
800	0	0	0	0	0	0	0
900	0.001	0	0	0.001	0	0.0004	0.000548
1000	0	0.001	0.001	0	0	0.0004	0.000548
2000	0	0	0	0.001	0	0.0002	0.000447
3000	0.001	0.001	0.002	0.001	0.001	0.0012	0.000447
4000	0.001	0.001	0.001	0.002	0.001	0.0012	0.000447
5000	0.002	0.002	0.002	0.002	0.002	0.002	0
6000	0.011	0.001	0.001	0.004	0.001	0.0036	0.004336
7000	0.002	0.003	0.002	0.003	0.002	0.0024	0.000548
8000	0.003	0.002	0.003	0.004	0.003	0.003	0.000707
9000	0.003	0.003	0.003	0.004	0.003	0.0032	0.000447
10000	0.003	0.007	0.003	0.005	0.003	0.0042	0.001789
11000	0.004	0.003	0.003	0.006	0.004	0.004	0.001225
12000	0.004	0.004	0.004	0.006	0.02	0.0076	0.006986
13000	0.004	0.005	0.011	0.007	0.004	0.0062	0.00295
14000	0.004	0.004	0.005	0.009	0.006	0.0056	0.002074
15000	0.005	0.005	0.005	0.008	0.005	0.0056	0.001342
16000	0.005	0.005	0.005	0.008	0.005	0.0056	0.001342

표 8 랜덤 데이터에 대한 힙 정렬

8) 소요시간 평균에 대한 정리

- 랜덤 데이터에 대한 정렬 소요시간 **평균**을 정리하면 다음의 표와 같다.

n	selection	insertion	quick	merge	heap
0	0	0	0	0.0002	0
100	0	0.0002	0	0	0
200	0.0002	0	0	0	0
300	0	0	0	0	0
400	0.0006	0.0002	0	0.0002	0.0002
500	0.0008	0.0002	0	0.0004	0.0002
600	0.0012	0.0006	0.0004	0	0.0002
700	0.0012	0.0012	0.0002	0	0.0002
800	0.0016	0.001	0	0.0002	0
900	0.0022	0.0008	0.001	0.0004	0.0004
1000	0.0054	0.0012	0.0002	0.0002	0.0004
2000	0.016	0.0046	0.0006	0.0002	0.0002
3000	0.0438	0.014	0.0006	0.0018	0.0012
4000	0.1042	0.0348	0.0076	0.0014	0.0012
5000	0.1616	0.0292	0.0012	0.0012	0.002
6000	0.1754	0.0642	0.002	0.0014	0.0036
7000	0.2196	0.0732	0.0024	0.0018	0.0024
8000	0.3494	0.1298	0.0022	0.0022	0.003
9000	0.3874	0.1674	0.0024	0.0022	0.0032
10000	0.4112	0.1862	0.0028	0.003	0.0042
11000	0.6116	0.1966	0.003	0.0034	0.004
12000	0.4752	0.2464	0.0034	0.0038	0.0076
13000	0.5492	0.32	0.004	0.0038	0.0062
14000	0.6834	0.3112	0.0038	0.0044	0.0056
15000	0.7168	0.3552	0.0046	0.0138	0.0056
16000	0.8762	0.3922	0.014	0.0056	0.0056

표 9 랜덤 데이터 정렬 평균 소요시간 표

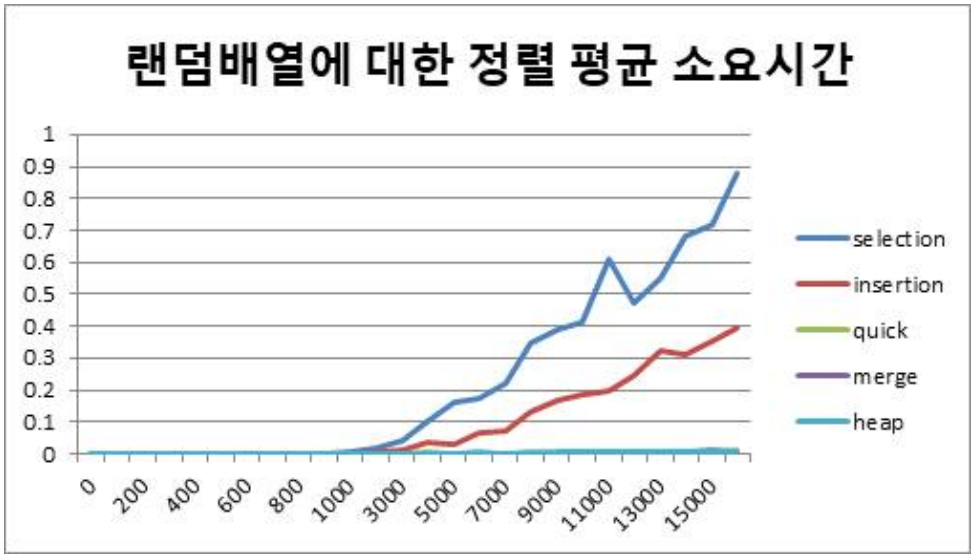

그림 13 랜덤 데이터 정렬 평균 소요시간

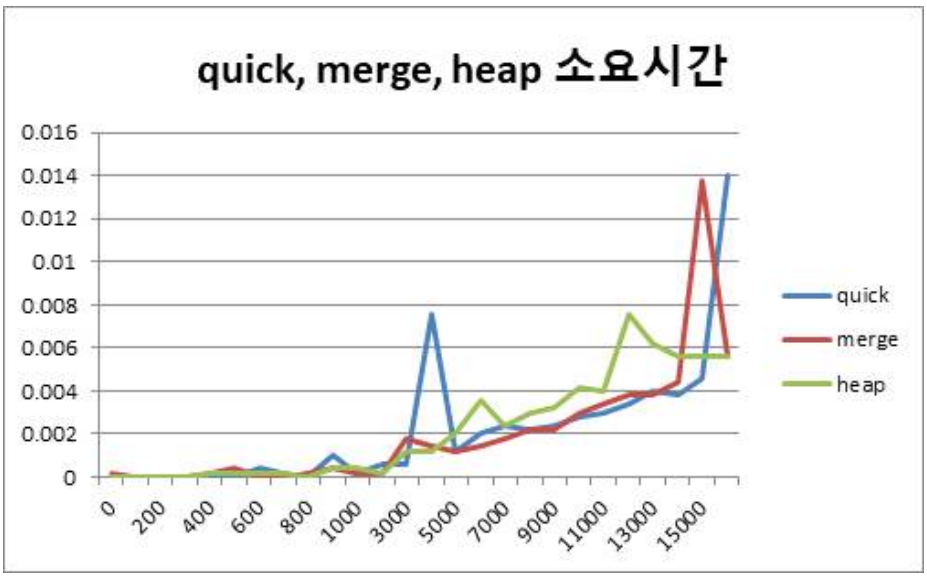

그림 14 랜덤 데이터 quick, merge, heap 정렬 평균 소요시간

- **선택 정렬** : 선택정렬의 랜덤 데이터에 대한 시간복잡도는 타 알고리즘에 비해 데이터 수가 늘어날수록 상향되는 경향을 크게 보인다.
- **삽입 정렬** : 선택정렬 보다는 작은 기울기로 데이터 수 증가에 따라 정렬 소요시간이 늘어남을 알 수 있다.
- **퀵 정렬, 합병 정렬, 힙 정렬** : 랜덤데이터에 대한 정렬 소요시간이 선택정렬, 삽입정렬에 비하면 많이 작은 수준으나 퀵, 합병, 힙 정렬의 소요시간 평균은 크게 차이가 없음을 알 수 있다.
- **평균 복잡도 : 선택 >> 삽입 >> 퀵 = 합병 = 힙**

9) 소요시간 표준편차에 대한 정리

n	selection	insertion	quick	merge	heap
0	0	0	0	0.000447	0
100	0	0.000447	0	0	0
200	0.000447	0	0	0	0
300	0	0	0	0	0
400	0.000548	0.000447	0	0.000447	0.000447
500	0.000447	0.000447	0	0.000548	0.000447
600	0.000447	0.000548	0.000548	0	0.000447
700	0.000447	0.002168	0.000447	0	0.000447
800	0.000548	0	0	0.000447	0
900	0.000447	0.000447	0.001732	0.000548	0.000548
1000	0.005983	0.000447	0.000447	0.000447	0.000548
2000	0.006364	0.000894	0.000548	0.000447	0.000447
3000	0.027105	0.007842	0.000548	0.001789	0.000447
4000	0.03528	0.025094	0.014206	0.000548	0.000447
5000	0.074909	0.003271	0.000447	0.000447	0
6000	0.090894	0.028604	0.000707	0.000548	0.004336
7000	0.088934	0.028934	0.002074	0.000837	0.000548
8000	0.136617	0.057686	0.000837	0.000447	0.000707
9000	0.126934	0.108062	0.000894	0.000447	0.000447
10000	0.130858	0.081555	0.000837	0.000707	0.001789
11000	0.332027	0.104189	0.000707	0.000894	0.001225
12000	0.131597	0.147135	0.000894	0.001789	0.006986
13000	0.189257	0.123242	0.000707	0.001304	0.00295
14000	0.259508	0.119391	0.001304	0.000894	0.002074
15000	0.226799	0.097392	0.00114	0.020813	0.001342
16000	0.228707	0.116731	0.015199	0.00251	0.001342

표 10 랜덤 데이터 정렬 소요시간 편차 표

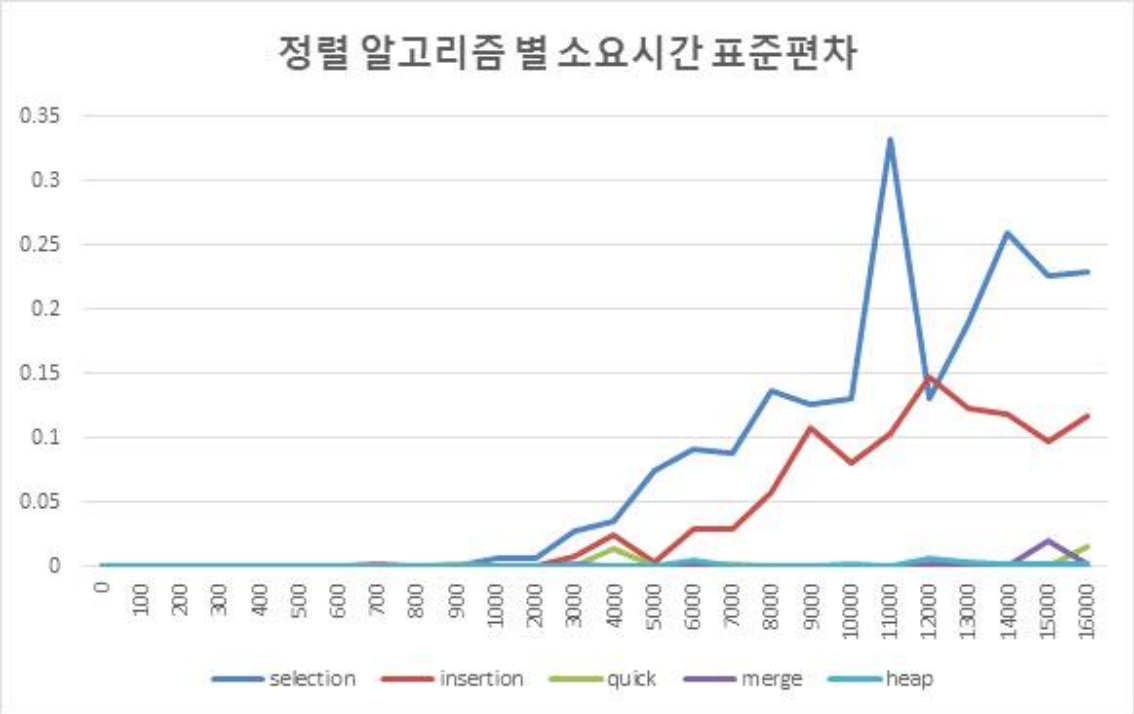

그림 15 랜덤데이터 정렬 소요시간 편차 그래프

- 표준편차의 방면으로 보았을 때, 데이터 수가 증가 할수록 선택정렬의 표준편차가 가장 크게 늘어나는 것을 볼 수 있다.
- 또한 삽입정렬의 표준편차는 선택정렬의 표준편차 증가량 보다 작지만 데이터 수의 증가에 따라 표준편차도 증가하는 경향을 보인다.
- 나머지 퀵, 합병, 힙 정렬의 표준편차는 0에 가까워 편차가 존재하지 않는다고 해석할 수 있다고 생각한다.

3. 결론

가. 이론적 복잡도와 실험결과의 복잡도 비교

1) 이론적 복잡도

알고리즘	최선	평균	최악
선택 정렬	n^2	n^2	n^2
삽입 정렬	n	n^2	n^2
퀵 정렬	$n \log n$	$n \log n$	n^2
합병 정렬	$n \log n$	$n \log n$	$n \log n$
힙 정렬	$n \log n$	$n \log n$	$n \log n$

표 11 정렬 알고리즘의 이론적 복잡도

2) 실험결과의 복잡도

알고리즘	최선	평균	최악
선택 정렬	**0.615**	**0.8762**	**0.624**
삽입 정렬	**0**	**0.3922**	**0.618**
퀵 정렬	**0.277**	**0.014**	**0.568**
합병 정렬	**0.002**	**0.0056**	**0.003**
힙 정렬	**0.005**	**0.0056**	**0.004**

표 12 실험 결과에 따른 정렬 알고리즘의 성능

위 표는 n = 16000에서 정렬 소요시간을 나타낸 것입니다.

★ **실험결과에 의한 복잡도 비교**

- 최선 복잡도 : 선택 > 퀵 >>> 삽입 = 합병 = 힙
- 평균 복잡도 : 선택 >> 삽입 >> 퀵 = 합병 = 힙
- 최악 복잡도 : 선택 = 삽입 = 퀵 >>> 합병 = 힙

3) 이론적 복잡도와 실험결과의 복잡도 비교

- 이론적 복잡도를 살펴볼 때, 삽입정렬은 n, 퀵 정렬은 $n \log n$, 합병 정렬은 $n \log n$의 복잡도를 가지는 반면, 실험 결과에서 삽입정렬의 소요시간은 0, 퀵 정렬은 합병정렬과 다르게 0.277의 소요시간을 가지며 이론적 복잡도와 맞지 않는 결과가 나타났습니다.
- 또한 평균 복잡도의 비교시 이론적 복잡도에서 선택정렬과 삽입정렬의 복잡도는 n^2으로 같았으나 실험결과를 보았을 때 선택정렬의 소요시간이 삽입 정렬에 비해 약 2배 이상 높은 모습을 볼 수 있었습니다.

나. 정리

1) 이론적 복잡도와 실제 복잡도의 차이

- 윗 장의 결과를 통해 이론적 복잡도에 비해 실제 실험결과에서의 소요시간에 차이가 있음을 알 수 있었다.
- 특히, 선택정렬 알고리즘의 경우 내림차순 데이터를 오름차순으로 정렬하는 과정과, 랜덤 데이터를 오름차순 정렬 하는 과정에서 랜덤데이터 정렬 소요시간의 평균이 내림차순 데이터를 오름차순으로 정렬하는데 드는 소요시간이 더 길다는 결과를 얻었다.
- 삽입 정렬의 경우 이론적 평균 복잡도가 선택정렬과 같은 반면 실험 결과에서는 선택정렬의 1/2 에 해당하는 소요시간을 나타냄으로써 이론적 복잡도와 차이가 있음을 보였다.

다. 최종결론

- 실험결과로 인한 최선, 평균, 최악 소요시간과 랜덤 데이터 정렬에 대한 표준편차를 비교했을 때, 합병정렬과 힙 정렬이 효율적인 알고리즘으로 보이는데, 합병정렬은 데이터 수가 늘어남에 따라 메모리 사용량도 늘어나는 것을 고려하면 다량의 데이터 정렬 시에는 힙 정렬이 가장 효율적인 알고리즘이라고 할 수 있다.
- 또한, 선택정렬 알고리즘은 데이터 수가 증가할수록 소요시간의 증가폭이 커지며, 편차도 가장 큰 것을 보아 가장 비효율적인 알고리즘이라고 할 수 있다.