



알고리즘 레포트#2

과 목 명	알고리즘
교 수	우 진 운
학 번	32163006
이 름	이 건 욱

1-1. MergeSort Code

```
1 package report;
2
3 public class MergeSortClass {
4     private int a[], b[];
5     private int aSize;
6
7     public MergeSortClass(int arr[], int n) {
8         a = arr;
9         aSize = n;
10        b = new int[aSize + 1];
11    }
12
13    public int[] MergeSortCall() {
14        MergeSort(1, aSize);
15        return a;
16    }
17
18    public void MergeSort(int low, int high) {
19        if (low < high) {
20            int mid = (low + high) / 2;
21            MergeSort(low, mid);
22            MergeSort(mid + 1, high);
23            Merge(low, mid, high);
24        }
25    }
26
27    public void Merge(int low, int mid, int high) {
28        int h = low, i = low, j = mid + 1, k;
29        while ((h <= mid) && (j <= high)) {
30            if (a[h] <= a[j]) {
31                b[i] = a[h];
32                h++;
33            } else {
34                b[i] = a[j];
35                j++;
36            }
37            i++;
38        }
39        if (h > mid)
40            for (k = j; k <= high; k++) {
41                b[i] = a[k];
42                i++;
43            }
44        else
45            for (k = h; k <= mid; k++) {
46                b[i] = a[k];
47                i++;
48            }
49        for (k = low; k <= high; k++)
50            a[k] = b[k];
51    }
52
53    public static void main(String args[]) {
54        int size[] = { 1000, 5000, 10000, 20000, 50000, 100000 };
55        int store[][] = new int[6][]; // 정렬 후 저장할 배열
56
57        for (int i = 0; i < 6; i++)
58            store[i] = new int[size[i] + 10]; // 배열에 대한 사이즈
59
60        System.out.println("MergeSort");
61        System.out.println("[1000] [5000] [10000] [20000] [50000] [100000]");
62        for (int i = 0; i < 10; i++) { // 데이터 10개 테스트
63            for (int j = 0; j < 6; j++) // 각각의 크기만큼 6번 수행
```

```

64         for (int k = 0; k < size[j]; k++) // 할당받은 크기만큼 반복
65             store[j][k] = (int) (java.lang.Math.random() * size[j]);
66         // 난수 입력
67
68         for (int j = 0; j < 6; j++) {
69             MergeSortClass merge = new MergeSortClass(store[j], size[j]);
70             long before = System.nanoTime(); // 합병 정렬 시작 시간
71             store[j] = merge.MergeSortCall(); // 합병 정렬
72             long after = System.nanoTime(); // 합병 정렬 끝난 시간
73             long execute = after - before; // 합병 정렬 걸린 시간 구하기
74             System.out.print(execute + " ");
75         }
76         System.out.println("");
77     }
78     System.out.println();
79 }
80 }

```

1-2. MergeSort 실행화면

Console

<terminated> MergeSortClass [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\java.exe

MergeSort					
[1000]	[5000]	[10000]	[20000]	[50000]	[100000]
455200	719800	1993900	2961200	7438799	11226301
86499	392399	839700	1778200	4793400	10076100
94300	394599	841300	1760799	4757500	10171700
73200	399099	837400	1785800	4812501	10473900
72301	392699	838500	1822701	4929099	10161500
109500	465000	909900	1752100	4747000	10312000
67800	382200	854500	1814600	4908300	10688300
68099	381300	850100	1878501	5047900	10290600
69799	378600	848000	2007500	4936701	10450600
72000	403900	843000	1789500	5077800	10524300

2-1. QuickSort Code

```
1 package report;
2
3 public class QuickSortClass {
4     private int a[];
5     private int aSize;
6
7     private QuickSortClass(int arr[], int n) {
8         a = arr;
9         aSize = n;
10        a[n + 1] = Integer.MAX_VALUE;
11    }
12
13    public int[] QuickSortCall() {
14        QuickSort(1, aSize);
15        return a;
16    }
17
18    void QuickSort(int p, int q) {
19        if (p < q) {
20            int j = Partition(a, p, q + 1);
21            QuickSort(p, j - 1);
22            QuickSort(j + 1, q);
23        }
24    }
25
26    int Partition(int a[], int m, int p) {
27        int v = a[m];
28        int i = m, j = p;
29
30        do {
31            do
32                i++;
33            while (a[i] < v);
34
35            do
36                j--;
37            while (a[j] > v);
38
39            if (i < j)
40                Interchange(a, i, j);
41        } while (i < j);
42
43        a[m] = a[j];
44        a[j] = v;
45        return (j);
46    }
47
48    void Interchange(int a[], int i, int j) {
49        int temp = a[i];
50        a[i] = a[j];
51        a[j] = temp;
52    }
53
54    public static void main(String[] args) {
55        int size[] = { 1000, 5000, 10000, 20000, 50000, 100000 };
56        int store[][] = new int[6][]; // 정렬 후 저장할 배열
57
58        for (int i = 0; i < 6; i++)
59            store[i] = new int[size[i] + 10]; // 배열에 대한 사이즈
```

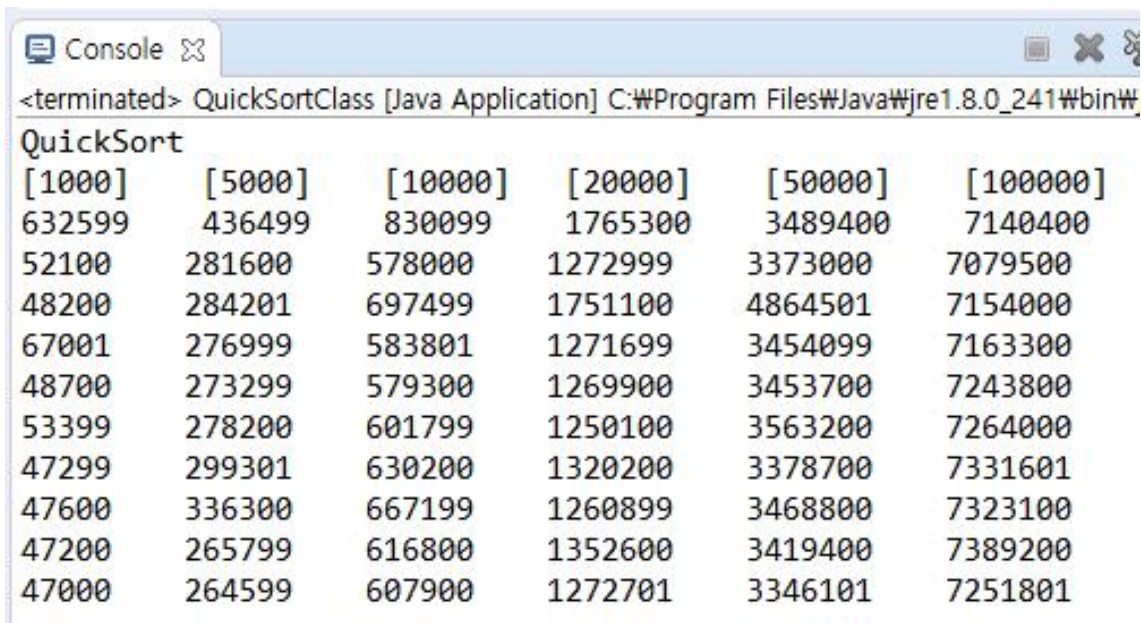


```

60
61 System.out.println("QuickSort");
62 System.out.println("[1000]    [5000]    [10000]    [20000]    [50000]    [100000]");
63 for (int i = 0; i < 10; i++) { // 데이터 10개 테스트
64     for (int j = 0; j < 6; j++) // 각각의 크기만큼 6번 수행
65         for (int k = 0; k < size[j]; k++) // 할당받은 크기만큼 반복
66             store[j][k] = (int) (java.lang.Math.random() * size[j]);
67     // 난수 입력
68
69     for (int l = 0; l < 6; l++) {
70         QuickSortClass quick = new QuickSortClass(store[l], size[l]);
71         long before = System.nanoTime(); // 퀵 정렬 시작 시간
72         store[l] = quick.QuickSortCall(); // 퀵 정렬
73         long after = System.nanoTime(); // 퀵 정렬 끝난 시간
74         long execute = after - before; // 퀵 정렬 걸린 시간 구하기
75         System.out.print(execute + "    ");
76     }
77     System.out.println("");
78 }
79 System.out.println();
80 }
81 }

```

2-2. QuickSort 실행화면



```

<terminated> QuickSortClass [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\
QuickSort
[1000]    [5000]    [10000]    [20000]    [50000]    [100000]
632599    436499    830099    1765300    3489400    7140400
52100     281600    578000    1272999    3373000    7079500
48200     284201    697499    1751100    4864501    7154000
67001     276999    583801    1271699    3454099    7163300
48700     273299    579300    1269900    3453700    7243800
53399     278200    601799    1250100    3563200    7264000
47299     299301    630200    1320200    3378700    7331601
47600     336300    667199    1260899    3468800    7323100
47200     265799    616800    1352600    3419400    7389200
47000     264599    607900    1272701    3346101    7251801

```

MergeSort										
	1	2	3	4	5	6	7	8	9	10
1000	455200	86499	94300	73200	72301	109500	67800	68099	69799	72000
5000	719800	392399	394599	399099	392699	465000	382200	381300	378600	403900
10000	1993900	839700	841300	837400	838500	909900	854500	850100	848000	843000
20000	2961200	1778200	1760799	1785800	1822701	1752100	1814600	1878501	2007500	1789500
50000	7438799	4793400	4757500	4812501	4929099	4747000	4908300	5047900	4936701	5077800
100000	11226301	10076100	10171700	10473900	10161500	10312000	10688300	10290600	10450600	10524300

QuickSort										
	1	2	3	4	5	6	7	8	9	10
1000	632599	52100	48200	67001	48700	53399	47299	47600	47200	47000
5000	436499	281600	284201	276999	273299	278200	299301	336300	265799	264599
10000	830099	578000	697499	583801	579300	601799	630200	667199	616800	607900
20000	1765300	1272999	1751100	1271699	1269900	1250100	1320200	1260899	1352600	1272701
50000	3489400	3373000	4864501	3454099	3453700	3563200	3378700	3468800	3419400	3346101
100000	7140400	7079500	7154000	7163300	7243800	7264000	7331601	7323100	7389200	7251801

Average						
	1000	5000	10000	20000	50000	100000
Quick	116869.8	430959.6	965630	1935090.1	5144900	10437530.1
Merge	109109.8	299679.7	639259.7	1378749.8	3581090.1	7234070.2