Algorithms

Midterm Assignment

Report

알고리즘 03분반

소프트웨어학부

20202475 이동훈

Contents

1. Program Codes

setting	setting_random

``c
void setting(int arr[], int n) {
    for(int i = 0; i < n; i++)
        arr[i] = n - i;
}
``
	``c
	void setting_random(int arr[], int n)
	    srand(time(NULL));
	
	    for(int i = 0; i < n; i++)
	        arr[i] = rand() % n + 1;
	``

bubble sort	insertion sort

``c
void bubbleSort(int arr[], int n) {
    int i, j;
    int tmp;
 	
    for(i = n - 1; i > 0; i--) {
        for(j = 0; j < i; j++) {
            if(arr[j] > arr[j + 1]) {
                tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
        }
    }
}
``
	``c
	void insertionSort(int arr[], int n)
	    int i, j;
	    int tmp;
	
	    for(i = 1; i < n; i++) {
	        tmp = arr[i];
	        for(j = i - 1; j >= 0; j--) {
	            if(arr[j] > tmp) {
	                arr[j + 1] = arr[j];
	            }
	            else break;
	        }
	        arr[j + 1] = tmp;
	    }
	}
	``

combine – merge sort function	

``c
void combine(int arr[], int l, int mid, int r) {
    int i1 = l, i2 = mid + 1;
    int l1 = mid, l2 = r;
    int tmp[10000], cnt = 0;
 	
    while(i1 <= mid && i2 <= r) {
        if(arr[i1] <= arr[i2])
            tmp[cnt++] = arr[i1++];
        else
            tmp[cnt++] = arr[i2++];
    }
    for(; i1 <= l1; i1++)
        tmp[cnt++] = arr[i1];
 	
    for(; i2 <= l2; i2++)
        tmp[cnt++] = arr[i2];
    cnt = 0;
    for(int i = l; i <= r; i++)
        arr[i] = tmp[cnt++];
}
``	

merge sort

```c
void mergeSort(int arr[], int l, int r) {
    int mid;

    if(l < r) {
        mid = (l + r) / 2;

        mergeSort(arr, l, mid);
        mergeSort(arr, mid + 1, r);
        combine(arr, l, mid, r);
    }
}
```

partition – quick sort function

```c
int partition(int arr[], int l, int r) {
    int pivot = l;
    int low, high, tmp;
    l++;
    low = l; high = r;

    while(low < high) {
        for(;arr[low] <= arr[pivot] && low <= r; low++);
        for(;arr[high] > arr[pivot] && high >= l; high--);

        if(low < high) {
            tmp = arr[low];
            arr[low] = arr[high];
            arr[high] = tmp;
        }
    }

    if(arr[pivot] > arr[high]) {
        tmp = arr[pivot];
        arr[pivot] = arr[high];
        arr[high] = tmp;
    }
    return high;
}
```

quick sort

```
void quickSort(int arr[], int l, int r) {
    if(l < r) {
        int p = partition(arr, l, r);

        if(p - 1 > l) quickSort(arr, l, p - 1);
        if(p + 1 < r) quickSort(arr, p + 1, r);
    }
}
```

radix sort

```
void radixSort(int arr[], int n) {
    int bucketArr[10][n];
    int bucketNum, p = 0;
    int len = 0, tmp;

    tmp = n;
    while(tmp > 0) {
        tmp /= 10;
        len++;
    }

    for(int k = 0; k < len; k++) {
        p = 0;
        int cnt[10] = {0, };
        for(int i = 0; i < n; i++) {
            bucketNum = arr[i] / (int)(pow(10, k)) % 10;
            bucketArr[bucketNum][cnt[bucketNum]++] = arr[i];
        }

        for(int i = 0; i < 10; i++) {
            for(int j = 0; j < cnt[i]; j++) {
                if(bucketArr[i][j] != 0) {
                    arr[p++] = bucketArr[i][j];
                }
            }
        }
    }
}
```

bucket sort

```c
int bucketArr[501][10001];

void bucketSort(int arr[], int n, int bn) {;
    int cnt[bn + 1];
    int bucketNum, p = 0;

    for(int i = 0; i <= bn; i++)
        cnt[i] = 0;

    for(int i = 0; i < n; i++) {
        bucketNum = arr[i] / (n / bn);
        bucketArr[bucketNum][cnt[bucketNum]++] = arr[i];
    }

    for(int i = 0; i <= bn; i++) {
        if(cnt[i] != 0) {
            quickSort(bucketArr[i], 0, cnt[i] - 1);
        }
    }

    for(int i = 0; i <= bn; i++) {
        for(int j = 0; j < cnt[i]; j++) {
            if(bucketArr[i][j] != 0) {
                arr[p++] = bucketArr[i][j];
            }
        }
    }
}
```

print function

```c
void print(int arr[], int n) {
    for(int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

execute function	
```c void execute(int n) { int arr[n]; int idx, bn; clock_t start, end; if (n == 1000) {idx = 0; bn = 50;} else if (n == 5000) {idx = 1; bn = 250;} else if (n == 10000) {idx = 2; bn = 500;} setting_random(arr, n); print(arr, n); start = clock(); bubbleSort(arr, n); end = clock(); print(arr, n); resultSet[idx][0] = end - start; setting_random(arr, n); print(arr, n); start = clock(); insertionSort(arr, n); end = clock(); print(arr, n); resultSet[idx][1] = end - start; setting_random(arr, n); print(arr, n); start = clock(); mergeSort(arr, 1, n); end = clock(); print(arr, n); resultSet[idx][2] = end - start; ```	
```c setting_random(arr, n); print(arr, n); start = clock(); quickSort(arr, 1, n); end = clock(); print(arr, n); resultSet[idx][3] = end - start; setting_random(arr, n); print(arr, n); start = clock(); radixSort(arr, n); end = clock(); print(arr, n); resultSet[idx][4] = end - start; setting_random(arr, n); print(arr, n); printf("\n\n\n"); start = clock(); bucketSort(arr, n, bn); end = clock(); print(arr, n); resultSet[idx][5] = end - start; printf("\n"); } ```	

main function

```c
int main() {
    int n;
    char name[6][20] = {"Bubble Sort", "Insertion Sort", "Merge Sort",
                        "Quick Sort", "Radix Sort", "Bucket Sort"};

    n = 1000;
    execute(n);

    n = 5000;
//    execute(n);

    n = 10000;
//    execute(n);

    printf("        N        |    1000    5000    10000\n");
    printf("-----------------|------------------------------\n");
    for(int i = 0; i < 6; i++) {
        printf("  %14s |   %3ld     %3ld      %3ld\n", name[i],
                resultSet[0][i], resultSet[1][i], resultSet[2][i]);
    }

    return 0;
}
```

2. Implementation

1) Decreasing Array

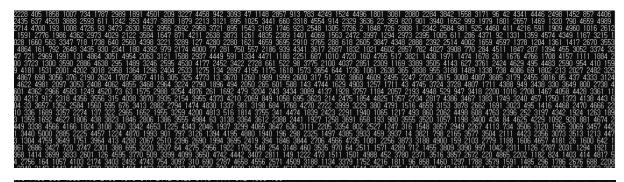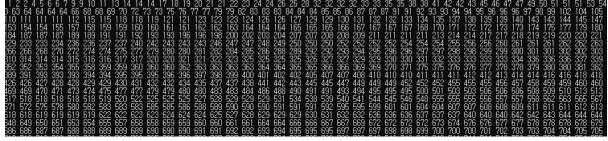| N		1000	5000	10000
Bubble Sort		2	44	186
Insertion Sort		1	35	143
Merge Sort		0	0	1
Quick Sort		1	34	136
Radix Sort		0	1	0
Bucket Sort		0	0	0

2) Random Array
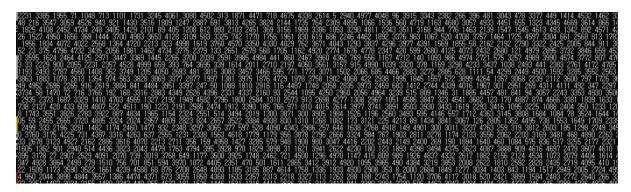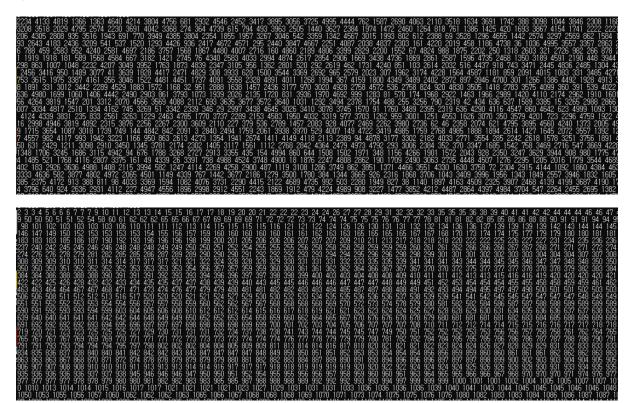
(1) N = 1000

① Bubble Sort

② Insertion Sort

③ Merge Sort

④ Quick Sort
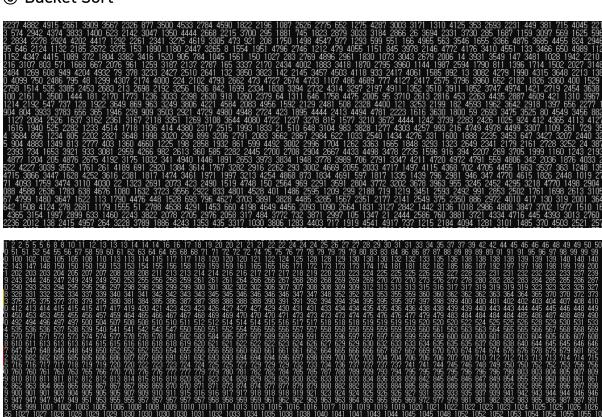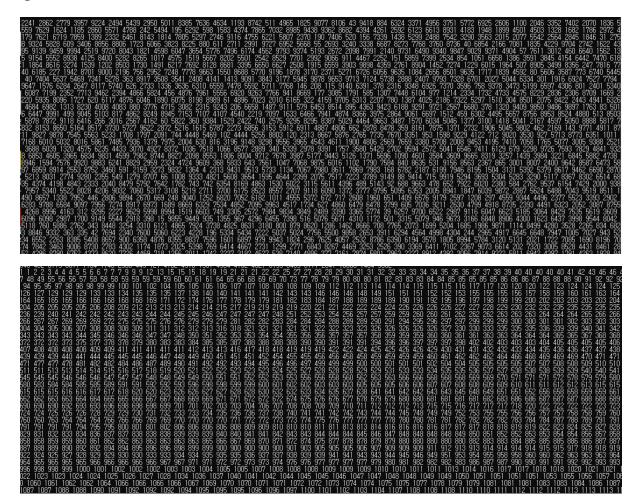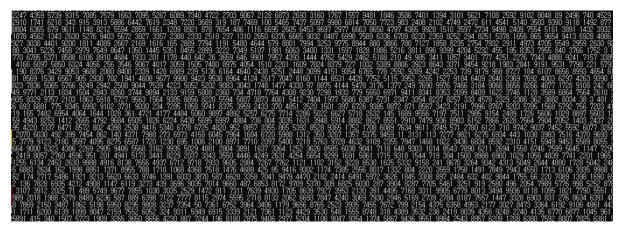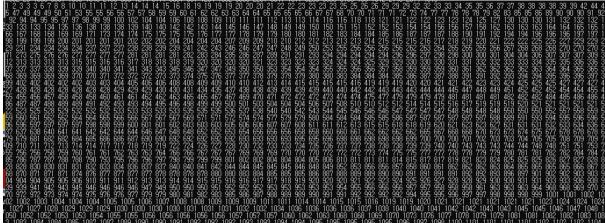
⑤ Radix Sort

⑥ Bucket Sort

(2) N = 5000

① Bubble Sort
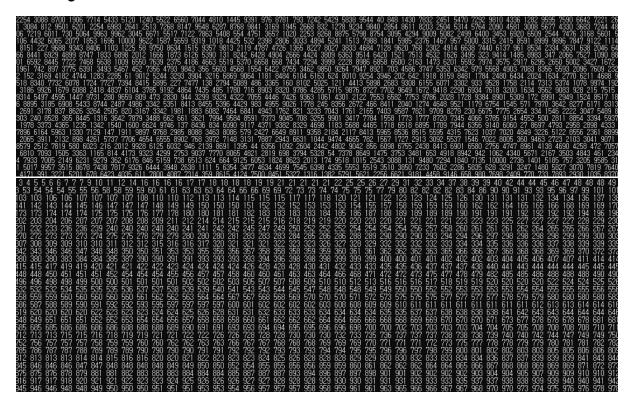

② Insertion Sort

③ Merge Sort

④ Quick Sort

⑤ Radix Sort

⑥ Bucket Sort

(3) N = 10000

① Bubble Sort

② Insertion Sort

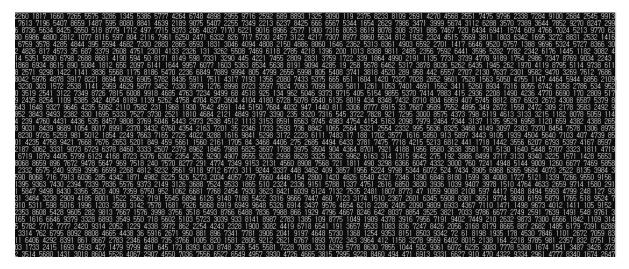

③ Merge Sort

④ Quick Sort

⑤ Radix Sort

⑥ Bucket Sort

3. Result Table

1) Decreasing Array

```
        N         |    1000     5000    10000
------------------|-------------------------------
    Bubble Sort   |     2        44      186
 Insertion Sort   |     1        35      143
     Merge Sort   |     0         0        1
     Quick Sort   |     1        34      136
     Radix Sort   |     0         1        0
    Bucket Sort   |     0         0        0
```

2) Random Array

```
        N         |    1000     5000    10000
------------------|-------------------------------
    Bubble Sort   |     1        46      199
 Insertion Sort   |     0        17       69
     Merge Sort   |     0         1        2
     Quick Sort   |     0         1        1
     Radix Sort   |     0         0        1
    Bucket Sort   |     0         1        1
```