# Algorithms Midterm Assignment



알고리즘 03분반 소프트웨어학부

20202475 이동훈

## Contents

- 1. Program Codes
- 2. Implementation
  - 1) Decreasing Array
  - 2) Random Array
- 3. Result Table

#### 1. Program Codes

```
setting
                                                       setting_random
                                                       /oid setting_random(int arr[], int n)
void setting(int arr[], int n)
                                                            srand(time(NULL));
      for(int i = 0; i < n; i++)
    arr[i] = n - i;</pre>
                                                            for(int i = 0; i < n; i++)
    arr[i] = rand() % n + 1;</pre>
bubble sort
                                                       insertion sort
                                                        void insertionSort(int arr[], int n)
void bubbleSort(int arr[], int n) {
                                                             int i, j;
int tmp;
     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];
      }
}
    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;
}
                                                                  arr[j + 1] = tmp;
combine – merge sort function
void combine(int arr[], int I, int mid, int r) {
      int i1 = 1, 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 <= |1; i1++)
             tmp[cnt++] = arr[i1];
      for(; i2 <= 12; i2++)
            tmp[cnt++] = arr[i2];
      cnt = 0;
            arr[i] = tmp[cnt++];
```

```
merge sort

void mergeSort(int arr[], int I, int r) {
   int mid;

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

      mergeSort(arr, |, mid);
      mergeSort(arr, mid + 1, r);
      combine(arr, |, mid, r);
   }
}</pre>
```

partition – quick sort function

```
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[pivot] = tmp;
}
return high;
```

```
quick sort

void quickSort(int arr[], int I, int r) {
    if(| < r) {
        int p = partition(arr, I, r);
        if(p - 1 > I) quickSort(arr, I, p - 1);
        if(p + 1 < r) quickSort(arr, p + 1, r);
    }
}</pre>
```

radix sort

```
bucket sort
```

```
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++) {
        if(bucketArr[i][j] != 0) {
            arr[p++] = bucketArr[i][j];
        }
    }
}</pre>
```

#### print function

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

```
execute function
```

```
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;
      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");
```

# 2. Implementation

# 1) Decreasing Array

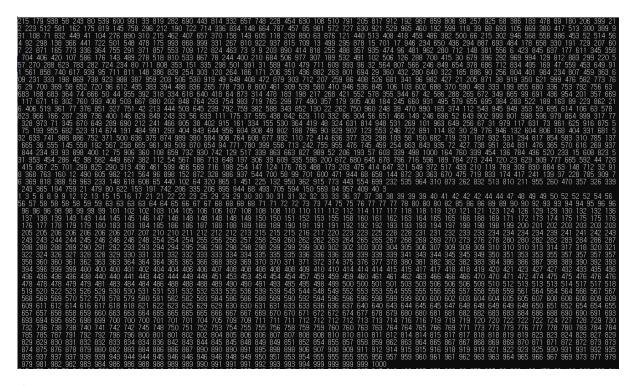
5 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	38 37 36 35 34 33 32 31 30 29 28 27 26 2
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 6  1 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 1  115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 1  144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 1  173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 1  202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 2  231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 2  260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 2  289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 3  318 319 320 321 322 323 324 325 353 354 355 366 357 358 359 360 361 362 363 364 365 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 340 441 442 443 444 445 446 447 448 449 450 421 422 423 443 445 436 437 438 439 350 351 352 353 354 435 356 357 358 359 360 361 362 363 364 365 363 364 365 363 364 365 365 365 365 365 365 365 365 365 365	9 30 31 32 33 34 35 36 37 38 39 40 41 42 36 36 36 69 70 71 72 73 74 75 76 77 78 79 80 8 105 105 106 107 108 109 110 111 112 113 114 134 135 136 137 138 139 140 141 142 143 163 164 165 166 167 168 169 170 171 172 1932 193 194 195 196 197 198 199 200 201 221 222 223 224 225 226 227 228 229 230 250 251 252 253 254 255 256 257 258 259 279 280 281 282 283 284 285 286 287 288 308 309 310 311 312 313 314 315 316 317 337 338 339 340 341 342 343 344 345 346 367 368 369 370 371 372 373 374 375 375 375 375 375 375 375 375 375 375

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

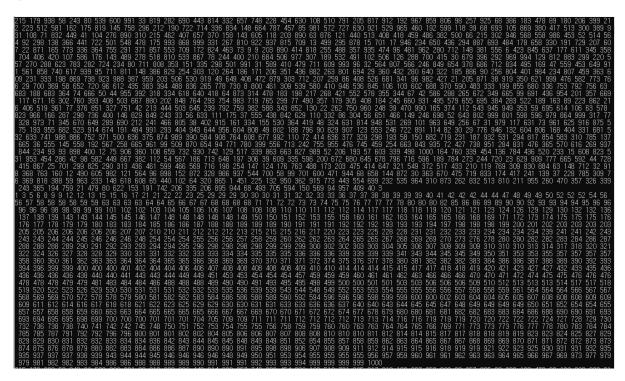
## 2) Random Array

(1) N = 1000

#### ① Bubble Sort



#### ② Insertion Sort



#### 3 Merge Sort

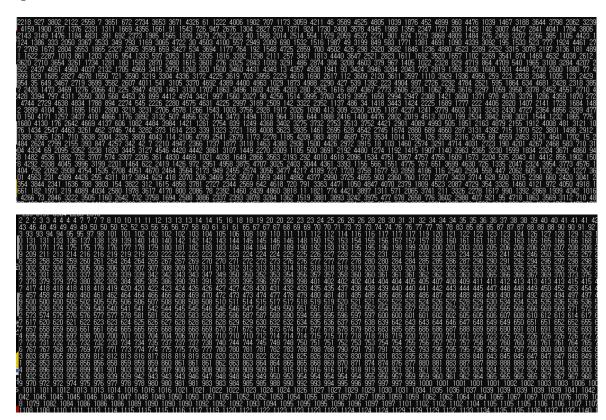
#### 4 Quick Sort

#### ⑤ Radix Sort

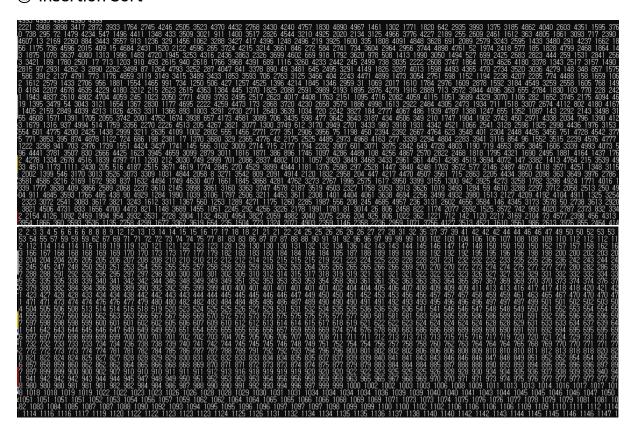
#### 6 Bucket Sort

## (2) N = 5000

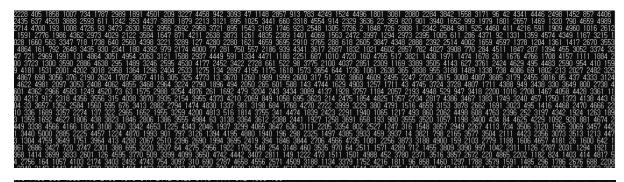
## ① Bubble Sort

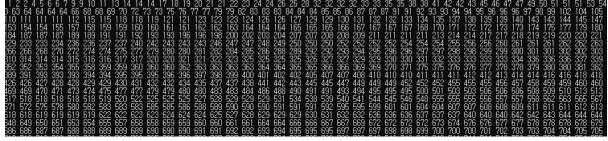


#### ② Insertion Sort

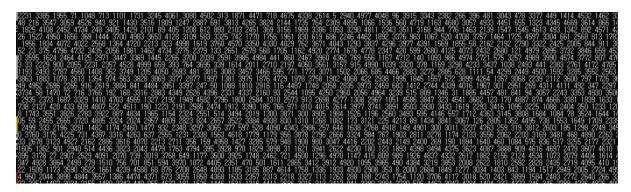


#### 3 Merge Sort



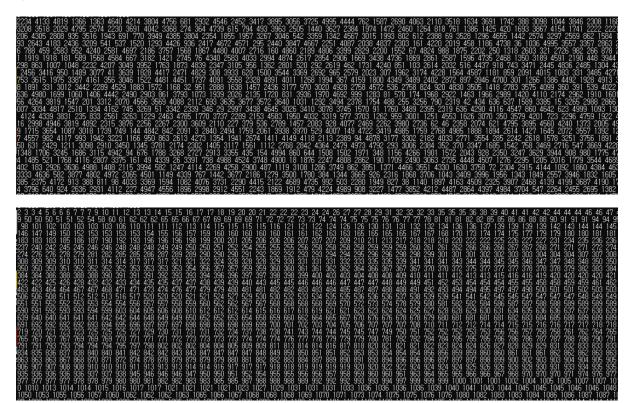


#### 4 Quick Sort

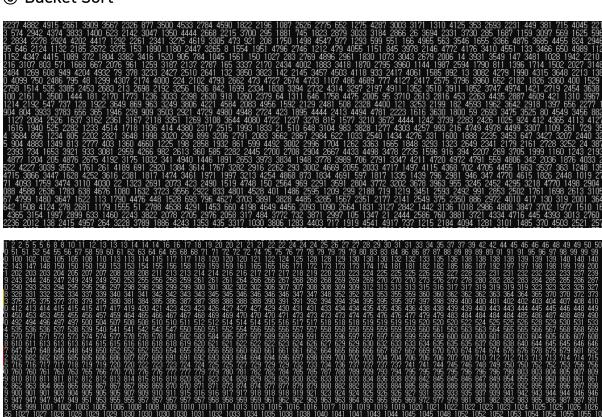




#### ⑤ Radix Sort

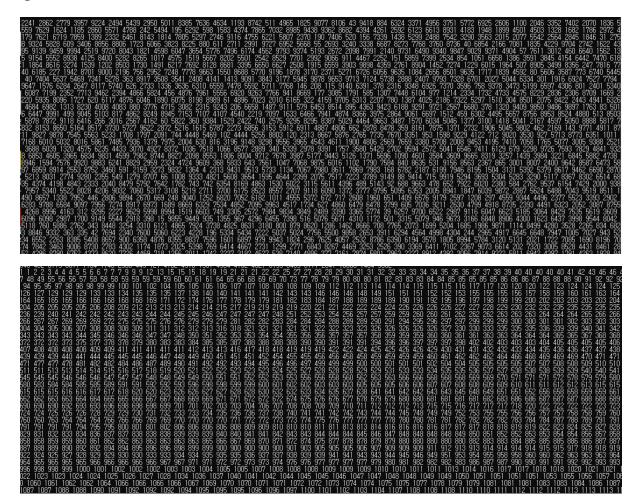


#### 6 Bucket Sort

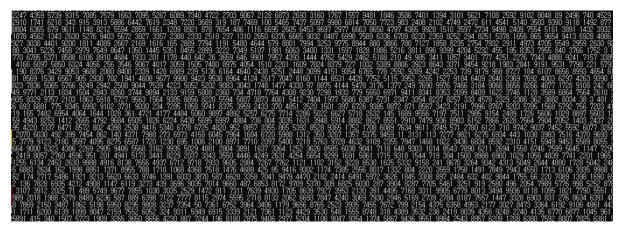


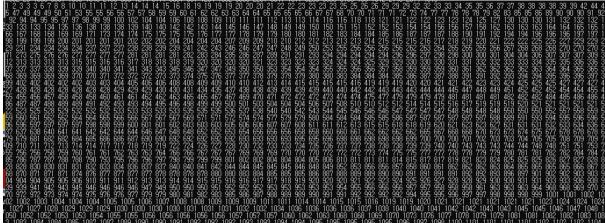
## (3) N = 10000

## ① Bubble Sort

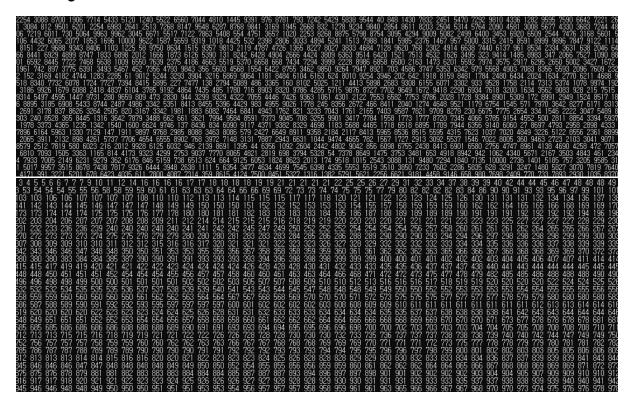


#### 2 Insertion Sort

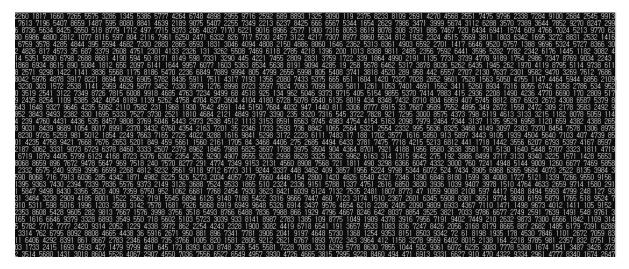




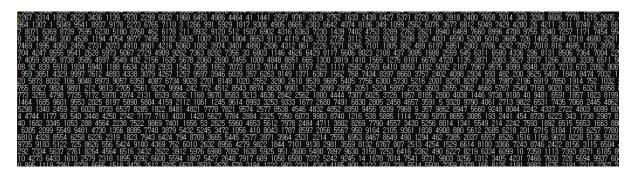
#### ③ Merge Sort

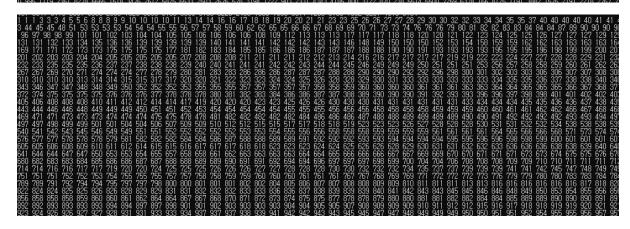


#### 4 Quick Sort

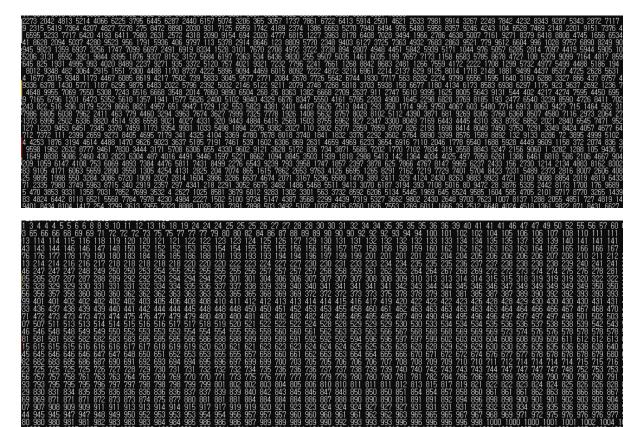


#### ⑤ Radix Sort





#### 6 Bucket Sort



## 3. Result Table

## 1) Decreasing Array

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

# 2) Random Array

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