Algorithms : Midterm Assignment

응용통계학과 20202850 김지민

Six algorithms :

bubble sort, insertion sort, merge sort, quicksort, radix sort, bucket sort

Input data :

elements of input are assigned in decreasing order.

Output data :

If n is 1000, the first, 250th, 500th, 750th, and 1000th elements were printed.

If n is 5000, the first, 1250th, 2500th, 3750th, and 5000th elements were printed.

If n is 10000, the first, 2500th, 5000th, 7500th, and 10000th elements were printed.

In addition, a function that returns 'T' if the output value is accurate(output is in increasing order) and 'F' if not accurate was created and output together at the end.

```
char output_correct(int list[], int n) {
    for (int i = 0; i < n; i++) {
        if (list[i] != i + 1) {
            printf("%d %d", list[i], i + 1);
            return 'F';
        }
    }
    return 'T';
}
```

1. Bubble sort

- **Algorithm code**

```
void bubble_sort(int list[], int n) {
    int i, j, temp;
    for (i = n - 1; i > 0; i--) {
        for (j = 0; j < i; j++) {
            if (list[j] > list[j + 1]) {
                temp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = temp;
            }
        }
    }
}
```

- **Inputs and Outputs**

```
int main() {

    int num, i;
    int a[1000];
    num = 1000;
    for (i = 0; i < 1000; i++)
        a[i] = num--;
    printf("n = 1000 ... \n");
    printf("input(bubble sort)  : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    bubble_sort(a, 1000);
    printf("output(bubble sort) : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    printf("output is correct?  : %c\n\n", output_correct(a, 1000));
    int b[5000];
    num = 5000;
    for (i = 0; i < 5000; i++)
        b[i] = num--;
    printf("n = 5000 ... \n");
    printf("input(bubble sort)  : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    bubble_sort(b, 5000);
    printf("output(bubble sort) : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    printf("output is correct?  : %c\n\n", output_correct(b, 5000));
    int c[10000];
    num = 10000;
    for (i = 0; i < 10000; i++)
        c[i] = num--;
    printf("n = 10000 ... \n");
    printf("input(bubble sort)  : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    bubble_sort(c, 10000);
    printf("output(bubble sort) : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    printf("output is correct?  : %c\n", output_correct(c, 10000));

    return 0;
}
```


Microsoft Visual Studio 디버그 콘솔
n = 1000 ...
input(bubble sort) : [1] 1000 [250] 751 [500] 501 [750] 251 [1000] 1
output(bubble sort) : [1] 1 [250] 250 [500] 500 [750] 750 [1000] 1000
output is correct? : T
n = 5000 ...
input(bubble sort) : [1] 5000 [1250] 3751 [2500] 2501 [3750] 1251 [5000] 1
output(bubble sort) : [1] 1 [1250] 1250 [2500] 2500 [3750] 3750 [5000] 5000
output is correct? : T
n = 10000 ...
input(bubble sort) : [1] 10000 [2500] 7501 [5000] 5001 [7500] 2501 [10000] 1
output(bubble sort) : [1] 1 [2500] 2500 [5000] 5000 [7500] 7500 [10000] 10000
output is correct? : T
C:₩Users₩jimin.DESKTOP-8V2OQSQ₩source₩repos₩Project1₩Debug₩Project1.exe(프로세스 2041

2. Insertion sort

- **Algorithm code**

```
void insertion_sort(int list[], int n) {
    int i, j, key;
    for (i = 1; i < n; i++) {
        key = list[i]; // unsorted data
        for (j = i - 1; j >= 0 && list[j] > key; j--)
            list[j + 1] = list[j];
        list[j + 1] = key; // place in the position of the sorted interval
    }
}
```

- **Inputs and Outputs**

```
int main() {

    int num, i;
    int a[1000];
    num = 1000;
    for (i = 0; i < 1000; i++)
        a[i] = num--;
    printf("n = 1000 ... \n");
    printf("input(insertion sort)  : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    insertion_sort(a, 1000);
    printf("output(insertion sort) : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    printf("output is correct?  : %c\n\n", output_correct(a, 1000));
    int b[5000];
    num = 5000;
    for (i = 0; i < 5000; i++)
        b[i] = num--;
    printf("n = 5000 ... \n");
    printf("input(insertion sort)  : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    insertion_sort(b, 5000);
    printf("output(insertion sort) : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    printf("output is correct?  : %c\n\n", output_correct(b, 5000));
    int c[10000];
    num = 10000;
    for (i = 0; i < 10000; i++)
        c[i] = num--;
    printf("n = 10000 ... \n");
    printf("input(insertion sort)  : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    insertion_sort(c, 10000);
    printf("output(insertion sort) : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    printf("output is correct?  : %c\n", output_correct(c, 10000));

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
n = 1000 ...
input(insertion sort)  : [1] 1000 [250]  751 [500]  501 [750]  251 [1000]    1
output(insertion sort) : [1]    1 [250]  250 [500]  500 [750]  750 [1000] 1000
output is correct?  : T

n = 5000 ...
input(insertion sort)  : [1] 5000 [1250] 3751 [2500] 2501 [3750] 1251 [5000]    1
output(insertion sort) : [1]    1 [1250] 1250 [2500] 2500 [3750] 3750 [5000] 5000
output is correct?  : T

n = 10000 ...
input(insertion sort)  : [1] 10000 [2500]  7501 [5000]  5001 [7500]  2501 [10000]     1
output(insertion sort) : [1]     1 [2500]  2500 [5000]  5000 [7500]  7500 [10000] 10000
output is correct?  : T

C:\Users\jimin.DESKTOP-8V2OQSQ\source\repos\Project1\Debug\Project1.exe(프로세스 10992개)
```

3. Merge sort

- **Algorithm code**

```
//combine divided arrays
merge(int list[], int left, int mid, int right, int n) {
    int i, j, k;
    int* tmp = malloc(sizeof(int) * n);
    i = left; j = mid + 1; k = left;
    while (i <= mid && j <= right) {
        if (list[i] < list[j])
            tmp[k++] = list[i++];
        else
            tmp[k++] = list[j++];
    }
    while (i <= mid)
        tmp[k++] = list[i++];
    while (j <= right)
        tmp[k++] = list[j++];
    for (int a = left; a <= right; a++)
        list[a] = tmp[a];
    free(tmp);
}

void merge_sort(int list[], int left, int right, int n) {
    int mid;
    if (left < right) {
        mid = (left + right) / 2;
        merge_sort(list, left, mid, n);
        merge_sort(list, mid + 1, right, n);
        merge(list, left, mid, right, n);
    }
}
```

- **Inputs and Outputs**

```
int main() {

    int num, i;
    int a[1000];
    num = 1000;
    for (i = 0; i < 1000; i++)
        a[i] = num--;
    printf("n = 1000 ... \n");
    printf("input(merge sort)  : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    merge_sort(a, 0, 999, 1000);
    printf("output(merge sort) : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    printf("output is correct?  : %c\n\n", output_correct(a, 1000));
    int b[5000];
    num = 5000;
    for (i = 0; i < 5000; i++)
        b[i] = num--;
    printf("n = 5000 ... \n");
    printf("input(merge sort)  : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    merge_sort(b, 0, 4999, 5000);
    printf("output(merge sort) : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    printf("output is correct?  : %c\n\n", output_correct(b, 5000));
    int c[10000];
    num = 10000;
    for (i = 0; i < 10000; i++)
        c[i] = num--;
    printf("n = 10000 ... \n");
    printf("input(merge sort)  : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    merge_sort(c, 0, 9999, 10000);
    printf("output(merge sort) : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    printf("output is correct?  : %c\n", output_correct(c, 10000));

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
n = 1000 ...
input(merge sort)  : [1] 1000 [250]  751 [500]  501 [750]  251 [1000]    1
output(merge sort) : [1]    1 [250]  250 [500]  500 [750]  750 [1000] 1000
output is correct?  : T

n = 5000 ...
input(merge sort)  : [1] 5000 [1250] 3751 [2500] 2501 [3750] 1251 [5000]    1
output(merge sort) : [1]    1 [1250] 1250 [2500] 2500 [3750] 3750 [5000] 5000
output is correct?  : T

n = 10000 ...
input(merge sort)  : [1] 10000 [2500]  7501 [5000]  5001 [7500]  2501 [10000]     1
output(merge sort) : [1]     1 [2500]  2500 [5000]  5000 [7500]  7500 [10000] 10000
output is correct?  : T

C:\Users\jimin.DESKTOP-8V20QSQ\source\repos\Project1\Debug\Project1.exe(프로세스 24068
```

4. Quicksort

- Algorithm code

Stack code to be used for quicksort

```
typedef struct {
    int data[10000];
    int top;
} Stack;

void init_stack(Stack* s) {
    s->top = -1;
}

int isempty(Stack* s) {
    if (s->top == -1) return 1;
    else return 0;
}

int isfull(Stack* s) {
    if (s->top == 10000) return 1;
    else return 0;
}

void push(Stack* s, int element) {
    if (isfull(s)) {
        fprintf(stderr, "Stack is full.₩n");
        exit(1);
    }
    s->data[++s->top] = element;
}

int pop(Stack* s) {
    if (isempty(s)) {
        fprintf(stderr, "Stack is empty.₩n");
        exit(1);
    }
    return s->data[s->top--];
}
```

```c
void quick_sort(int list[], int left, int right) {
    Stack s;
    init_stack(&s);
    int pivot, pivot_idx, temp;
    int low, high, start, end;
    push(&s, right);
    push(&s, left);

    while (!isempty(&s)) {
        start = pop(&s);
        end = pop(&s);
        low = start;
        high = end + 1;
        pivot_idx = (start + end) / 2; // select a value for the middle index as pivot

        temp = list[pivot_idx]; // swap a pivot for leftmost value of array
        list[pivot_idx] = list[start];
        list[start] = temp;
        pivot = list[start];

        do {
            do // compare from the left
                low++;
            while (low <= end && list[low] < pivot);
            do // compare from the right
                high--;
            while (high >= start && list[high] > pivot);


            if (low < high) {
                temp = list[low];
                list[low] = list[high];
                list[high] = temp;
            }
        } while (low < high);

        //swap a pivot for the largest of the smaller values than the pivot
        temp = list[high];
        list[high] = list[start];
        list[start] = temp;

        pivot_idx = high;

        if (pivot_idx - 1 > start) {
            push(&s, pivot_idx - 1);
            push(&s, start);
        }
        if (pivot_idx + 1 < end) {
            push(&s, end);
            push(&s, pivot_idx + 1);
        }
    }
}
```

- **Inputs and Outputs**

```
int main() {

    int num, i;
    int a[1000];
    num = 1000;
    for (i = 0; i < 1000; i++)
        a[i] = num--;
    printf("n = 1000 ... \n");
    printf("input(quick sort)  : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    quick_sort(a, 0, 999);
    printf("output(quick sort) : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    printf("output is correct?  : %c\n\n", output_correct(a, 1000));
    int b[5000];
    num = 5000;
    for (i = 0; i < 5000; i++)
        b[i] = num--;
    printf("n = 5000 ... \n");
    printf("input(quick sort)  : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    quick_sort(b, 0, 4999);
    printf("output(quick sort) : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    printf("output is correct?  : %c\n\n", output_correct(b, 5000));
    int c[10000];
    num = 10000;
    for (i = 0; i < 10000; i++)
        c[i] = num--;
    printf("n = 10000 ... \n");
    printf("input(quick sort)  : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    quick_sort(c, 0, 9999);
    printf("output(quick sort) : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    printf("output is correct?  : %c\n", output_correct(c, 10000));

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
n = 1000 ...
input(quick sort)  : [1] 1000 [250]  751 [500]  501 [750]  251 [1000]    1
output(quick sort) : [1]    1 [250]  250 [500]  500 [750]  750 [1000] 1000
output is correct?  : T

n = 5000 ...
input(quick sort)  : [1] 5000 [1250] 3751 [2500] 2501 [3750] 1251 [5000]    1
output(quick sort) : [1]    1 [1250] 1250 [2500] 2500 [3750] 3750 [5000] 5000
output is correct?  : T

n = 10000 ...
input(quick sort)  : [1] 10000 [2500]  7501 [5000]  5001 [7500]  2501 [10000]     1
output(quick sort) : [1]     1 [2500]  2500 [5000]  5000 [7500]  7500 [10000] 10000
output is correct?  : T

C:\Users\jimin.DESKTOP-8V20QSQ\source\repos\Project1\Debug\Project1.exe(프로세스 19492
```

5. Radix sort

- **Algorithm code**

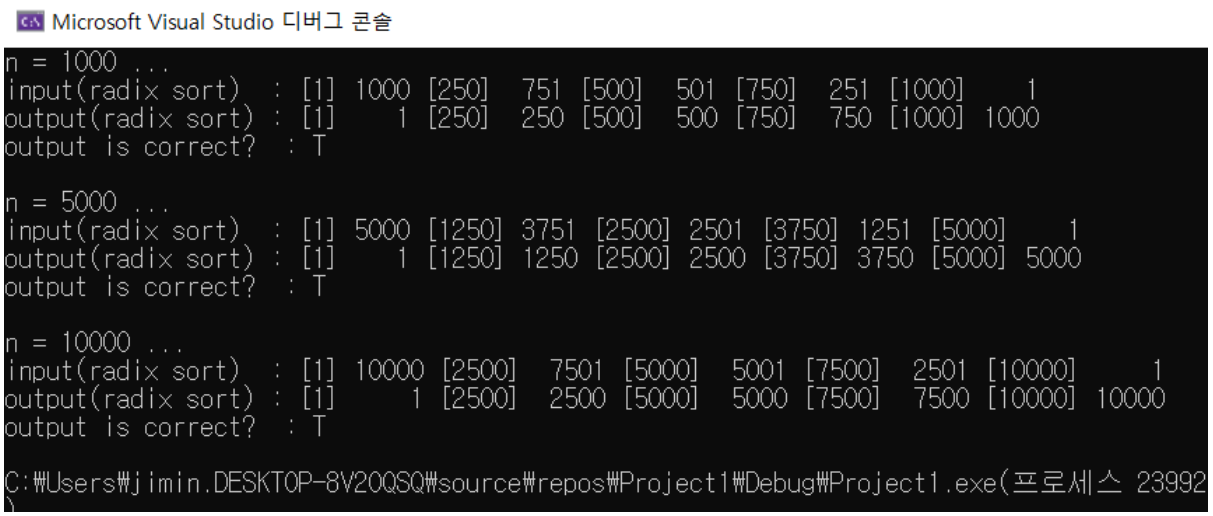
Queue code to be used radix sort

```
typedef struct {
    int queue[10001];
    int front;
    int rear;
} Queue;

void init(Queue* q) {
    q->front = q->rear = 0;
}

int is_empty(Queue* q) {
    return (q->front == q->rear);
}

int is_full(Queue* q) {
    return ((q->rear + 1) % 10001 == q->front);
}

void enqueue(Queue* q, int num) {
    if (is_full(q)) {
        fprintf(stderr, "Queue is full.₩n");
        exit(1);
    }
    q->rear = (q->rear + 1) % 10001;
    q->queue[q->rear] = num;
}

int dequeue(Queue* q) {
    if (is_empty(q)) {
        fprintf(stderr, "Queue is empty.₩n");
        exit(1);
    }
    q->front = (q->front + 1) % 10001;
    return q->queue[q->front];
}
```

```
void radix_sort(int list[], int n, int maxdigits) {
    int i, j, k;
    int pos = 1;
    Queue queues[10];

    for (i = 0; i < 10; i++) {
        init(&queues[i]);
    }

    for (j = 0; j < maxdigits; j++) {
        for (k = 0; k < n; k++) {
            // zero padding
            int num = 0;

            if (list[k] >= pos) {
                num = (list[k] / pos) % 10;
            }
            enqueue(&queues[num], list[k]);
        }
        k = 0;
        // move sorted data
        for (i = 0; i < 10; i++) {
            while (!is_empty(&queues[i])) {
                list[k++] = dequeue(&queues[i]);
            }
        }
        pos *= 10;
    }
}
```

- **Inputs and Outputs**

```
int main() {

    int num, i;
    int a[1000];
    num = 1000;
    for (i = 0; i < 1000; i++)
        a[i] = num--;
    printf("n = 1000 ... \n");
    printf("input(radix sort)  : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    radix_sort(a, 1000, 4);
    printf("output(radix sort) : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    printf("output is correct?  : %c\n\n", output_correct(a, 1000));
    int b[5000];
    num = 5000;
    for (i = 0; i < 5000; i++)
        b[i] = num--;
    printf("n = 5000 ... \n");
    printf("input(radix sort)  : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    radix_sort(b, 5000, 4);
    printf("output(radix sort) : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    printf("output is correct?  : %c\n\n", output_correct(b, 5000));
    int c[10000];
    num = 10000;
    for (i = 0; i < 10000; i++)
        c[i] = num--;
    printf("n = 10000 ... \n");
    printf("input(radix sort)  : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    radix_sort(c, 10000, 5);
    printf("output(radix sort) : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    printf("output is correct?  : %c\n", output_correct(c, 10000));

    return 0;
}
```


Microsoft Visual Studio 디버그 콘솔
n = 1000 ...
input(radix sort) : [1] 1000 [250] 751 [500] 501 [750] 251 [1000]   1
output(radix sort) : [1]   1 [250] 250 [500] 500 [750] 750 [1000] 1000
output is correct? : T
n = 5000 ...
input(radix sort) : [1] 5000 [1250] 3751 [2500] 2501 [3750] 1251 [5000]   1
output(radix sort) : [1]   1 [1250] 1250 [2500] 2500 [3750] 3750 [5000] 5000
output is correct? : T
n = 10000 ...
input(radix sort) : [1] 10000 [2500] 7501 [5000] 5001 [7500] 2501 [10000]   1
output(radix sort) : [1]   1 [2500] 2500 [5000] 5000 [7500] 7500 [10000] 10000
output is correct? : T
C:₩Users₩jimin.DESKTOP-8V20QSQ₩source₩repos₩Project1₩Debug₩Project1.exe(프로세스 23992

6. Bucket sort

- **Algorithm code**

```
void bucket_sort(int list[], int size, int bucket_size, int max) {

    max /= bucket_size;
    int i, j, k;
    int** arr = calloc(bucket_size, sizeof(int*));
    for (i = 0; i < bucket_size; i++) {
        // assume that list is almost evenly distributed and optionally allocate max*2
        arr[i] = calloc(max*2, sizeof(int));
    }

    int* buckets = calloc(bucket_size, sizeof(int));

    for (i = 0; i < size; i++) {
        // decide which bucket to put the data in
        k = (list[i] - 1) / max;
        arr[k][buckets[k]++] = list[i];
    }
    for (i = 0; i < bucket_size; i++) {
        insertion_sort(arr[i], buckets[i]);
    }
    int idx = 0;
    // move sorted data
    for (i = 0; i < bucket_size; i++) {
        for (j = 0; j < buckets[i]; j++) {
            list[idx++] = arr[i][j];
        }
    }

    free(buckets);
    for (i = 0; i < bucket_size; i++) {
        free(arr[i]);
    }
    free(arr);
}
```

- **Inputs and Outputs**

```c
int main() {

    int num, i;
    int a[1000];
    num = 1000;
    for (i = 0; i < 1000; i++)
        a[i] = num--;
    printf("n = 1000 ... \n");
    printf("input(bucket sort)  : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    bucket_sort(a, 1000, 50, 1000);
    printf("output(bucket sort) : [1] %4d [250] %4d [500] %4d [750] %4d [1000] %4d\n", a[0], a[249], a[499], a[749], a[999]);
    printf("output is correct?  : %c\n\n", output_correct(a, 1000));
    int b[5000];
    num = 5000;
    for (i = 0; i < 5000; i++)
        b[i] = num--;
    printf("n = 5000 ... \n");
    printf("input(bucket sort)  : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    bucket_sort(b, 5000, 250, 5000);
    printf("output(bucket sort) : [1] %4d [1250] %4d [2500] %4d [3750] %4d [5000] %4d\n", b[0], b[1249], b[2499], b[3749], b[4999]);
    printf("output is correct?  : %c\n\n", output_correct(b, 5000));
    int c[10000];
    num = 10000;
    for (i = 0; i < 10000; i++)
        c[i] = num--;
    printf("n = 10000 ... \n");
    printf("input(bucket sort)  : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    bucket_sort(c, 10000, 500, 10000);
    printf("output(bucket sort) : [1] %5d [2500] %5d [5000] %5d [7500] %5d [10000] %5d\n", c[0], c[2499], c[4999], c[7499], c[9999]);
    printf("output is correct?  : %c\n", output_correct(c, 10000));

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
n = 1000 ...
input(bucket sort)  : [1] 1000 [250]  751 [500]  501 [750]  251 [1000]    1
output(bucket sort) : [1]    1 [250]  250 [500]  500 [750]  750 [1000] 1000
output is correct?  : T

n = 5000 ...
input(bucket sort)  : [1] 5000 [1250] 3751 [2500] 2501 [3750] 1251 [5000]    1
output(bucket sort) : [1]    1 [1250] 1250 [2500] 2500 [3750] 3750 [5000] 5000
output is correct?  : T

n = 10000 ...
input(bucket sort)  : [1] 10000 [2500]  7501 [5000]  5001 [7500]  2501 [10000]     1
output(bucket sort) : [1]     1 [2500]  2500 [5000]  5000 [7500]  7500 [10000] 10000
output is correct?  : T

C:₩Users₩jimin.DESKTOP-8V20QSQ₩source₩repos₩Project1₩Debug₩Project1.exe(프로세스 20288
```

A result table

- **Main code**

Check the execution time of each sort algorithm when n is 1000.

```
int main() {

    clock_t start, finish;

    printf("=============================================================================================\n");
    printf("|input size |                                Algorithms                                     |\n");
    printf("=============================================================================================\n");
    printf("|     n     |   bubble   | insertion  |   merge    |   quick    |   radix    |   bucket   |\n");

    int a[1000];
    int num = 1000;
    for (int i = 0; i < 1000; i++)
        a[i] = num--;

    start = clock();
    bubble_sort(a, 1000);
    finish = clock();
    double time1 = (double)(finish - start);

    num = 1000;
    for (int i = 0; i < 1000; i++)
        a[i] = num--;
    start = clock();
    insertion_sort(a, 1000);
    finish = clock();
    double time2 = (double)(finish - start);

    num = 1000;
    for (int i = 0; i < 1000; i++)
        a[i] = num--;
    start = clock();
    merge_sort(a, 0, 999, 1000);
    finish = clock();
    double time3 = (double)(finish - start);

    num = 1000;
    for (int i = 0; i < 1000; i++)
        a[i] = num--;
    start = clock();
    quick_sort(a, 0, 999);
    finish = clock();
    double time4 = (double)(finish - start);

    num = 1000;
    for (int i = 0; i < 1000; i++)
        a[i] = num--;
    start = clock();
    radix_sort(a, 1000, 4);
    finish = clock();
    double time5 = (double)(finish - start);

    num = 1000;
    for (int i = 0; i < 1000; i++)
        a[i] = num--;
    start = clock();
    bucket_sort(a, 1000, 50, 1000);
    finish = clock();
    double time6 = (double)(finish - start);
    printf("=============================================================================================\n");
    printf("| n = 1000  | %10f | %10f | %10f | %10f | %10f | %10f |\n",
        time1, time2, time3, time4, time5, time6);
```

Check the execution time of each sort algorithm when n is 5000.

```
num = 5000;
int b[5000];
for (int i = 0; i < 5000; i++) {
    b[i] = num--;
}

start = clock();
bubble_sort(b, 5000);
finish = clock();
time1 = (double)(finish - start);

num = 5000;
for (int i = 0; i < 5000; i++)
    b[i] = num--;
start = clock();
insertion_sort(b, 5000);
finish = clock();
time2 = (double)(finish - start);

num = 5000;
for (int i = 0; i < 5000; i++)
    b[i] = num--;
start = clock();
merge_sort(b, 0, 4999, 5000);
finish = clock();
time3 = (double)(finish - start);

num = 5000;
for (int i = 0; i < 5000; i++)
    b[i] = num--;
start = clock();
quick_sort(b, 0, 4999);
finish = clock();
time4 = (double)(finish - start);

num = 5000;
for (int i = 0; i < 5000; i++)
    b[i] = num--;
start = clock();
radix_sort(b, 5000, 4);
finish = clock();
time5 = (double)(finish - start);

num = 5000;
for (int i = 0; i < 5000; i++)
    b[i] = num--;
start = clock();
bucket_sort(b, 5000, 250, 5000);
finish = clock();
time6 = (double)(finish - start);

printf("=============================================================================================\n");
printf("| n = 5000  | %10f | %10f | %10f | %10f | %10f | %10f |\n",
    time1, time2, time3, time4, time5, time6);
```

Check the execution time of each sort algorithm when n is 10000.

```
    num = 10000;
    int c[10000];
    for (int i = 0; i < 10000; i++)
        c[i] = num--;

    start = clock();
    bubble_sort(c, 10000);
    finish = clock();
    time1 = (double)(finish - start);

    num = 10000;
    for (int i = 0; i < 10000; i++)
        c[i] = num--;
    start = clock();
    insertion_sort(c, 10000);
    finish = clock();
    time2 = (double)(finish - start);

    num = 10000;
    for (int i = 0; i < 10000; i++)
        c[i] = num--;
    start = clock();
    merge_sort(c, 0, 9999, 10000);
    finish = clock();
    time3 = (double)(finish - start);

    num = 10000;
    for (int i = 0; i < 10000; i++)
        c[i] = num--;
    start = clock();
    quick_sort(c, 0, 9999);
    finish = clock();
    time4 = (double)(finish - start);

    num = 10000;
    for (int i = 0; i < 10000; i++)
        c[i] = num--;
    start = clock();
    radix_sort(c, 10000, 5);
    finish = clock();
    time5 = (double)(finish - start);

    num = 10000;
    for (int i = 0; i < 10000; i++)
        c[i] = num--;
    start = clock();
    bucket_sort(c, 10000, 500, 10000);
    finish = clock();
    time6 = (double)(finish - start);
    printf("=============================================================================================₩n");
    printf("| n = 10000 | %10f | %10f | %10f | %10f | %10f | %10f |₩n",
        time1, time2, time3, time4, time5, time6);
    printf("=============================================================================================₩n");

    return 0;
}
```

- **Result**

Microsoft Visual Studio 디버그 콘솔

```
=============================================================================================
|input size |                                Algorithms                                     |
=============================================================================================
|     n     |   bubble   | insertion |   merge    |   quick    |   radix    |   bucket   |
=============================================================================================
| n = 1000  |   5.000000 |  3.000000 |  1.000000 |  0.000000 |  2.000000 |  0.000000 |
=============================================================================================
| n = 5000  |  95.000000 | 74.000000 | 14.000000 |  2.000000 |  4.000000 |  1.000000 |
=============================================================================================
| n = 10000 | 418.000000 | 144.000000 | 23.000000 |  3.000000 | 11.000000 |  2.000000 |
=============================================================================================

C:₩Users₩jimin.DESKTOP-8V20QSQ₩source₩repos₩Project1₩Debug₩Project1.exe(프로세스 5980개)이(가
```