로고, 등록 상표, 폰트, 상징이(가) 표시된 사진

자동 생성된 설명

**데이터구조2**

**합병\_퀵\_기수\_sort**

|  |  |
| --- | --- |
| 과목명 | 데이터구조2 |
| 담당교수 | 남윤영 |
| 학과 | 컴퓨터공학과 |
| 학년 | 2 |
| 학번 | 20204101 |
| 이름 | 신동욱 |
| 제출일 | 11.27 |

1. **코드 분석**
2. **실행결과**
3. **전체 소스 코드**
4. **분석**

**문제 1. 시간 측정**

합병정렬의 시간복잡도: O(n log n)

퀵정렬의 시간복잡도: O(n log n)

기수정렬의 시간복잡도: O(dn)

일반적으로 퀵정렬의 속도가 제일 빠르다(Locality 관점) 이후 합병 정렬, 기수 정렬 순서대로 속도가 빠른 것을 알 수 있다

**문제 2. 정렬 과정 출력**

1. 합병정렬에 있어서, merge 함수가 호출될 때 마다 list의 값을 출력한다.

2. 퀵정렬에 있어서, partition 함수 내 SWAP 함수가 호출되기 전에 list의 값을 출력한다. 이 때 list의 값을 출력하는 함수의 인수로서, low와 high을 전달하여 (list[i] == list[low]), (list[i] == list[high]) 인 부분을 표시한다

3. 기수 정렬에 있어서, 각 요소들이 큐에 들어갔다가 나온 뒤의 list의 값을 출력한다

**문제 3. 기수정렬의 각 버킷 출력**

기수정렬의 radix\_sort 함수 내의 반복문에서 각 요소들이 enqueue 한 이후에 모든 큐를 출력한다

이하는 print\_bucket 함수의 일부분

for (int j = queues[i].front + 1; j <= queues[i].rear; j++) {

printf("%d ", queues[i].data[j]);

}

1. **실행 결과**

**과제1**

**텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명**

MAX\_SIZE = 20000

**퀵 > 합병 > 기수** 순서대로 속도가 빠른 것을 볼 수 있다

특이사항: 난수의 자리 수는 8자리이지만, DIGIST를 8로 하기에는 예외가 발생, 4로 지정함에도 유의미한 정보를 도출할 수 있다

**과제2**

**텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

자동 생성된 설명**

위에서부터 순서대로 합병, 퀵, 기수

**과제3**

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

기수 정렬에서의 각 큐의 상황

1. **전체 소스 코드**

**과제 1.**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) )

#define MAX\_SIZE 20000

int sorted[MAX\_SIZE];

//===================합병======================

void merge(int list[], int left, int mid, int right)

{

int i, j, k, l;

i = left; j = mid + 1; k = left;

while (i <= mid && j <= right) {

if (list[i] <= list[j]) sorted[k++] = list[i++];

else sorted[k++] = list[j++];

}

if (i > mid)

for (l = j; l <= right; l++)

sorted[k++] = list[l];

else

for (l = i; l <= mid; l++)

sorted[k++] = list[l];

for (l = left; l <= right; l++)

list[l] = sorted[l];

}

void merge\_sort(int list[], int left, int right)

{

int mid;

if (left < right)

{

mid = (left + right) / 2; // 리스트의 균등분할

merge\_sort(list, left, mid); // 부분리스트 정렬

merge\_sort(list, mid + 1, right);//부분리스트 정렬

merge(list, left, mid, right); // 합병

}

}

//===================퀵======================

int partition(int list[], int left, int right)

{

int pivot, temp;

int low, high;

low = left;

high = right + 1;

pivot = list[left];

do {

do

low++;

while (low <= right && list[low] < pivot);

do

high--;

while (high >= left && list[high] > pivot);

if (low < high) {

SWAP(list[low], list[high], temp);

}

} while (low < high);

SWAP(list[left], list[high], temp);

return high;

}

void quick\_sort(int list[], int left, int right)

{

if (left < right) {

int q = partition(list, left, right);

quick\_sort(list, left, q - 1);

quick\_sort(list, q + 1, right);

}

}

//===================기수======================

#define MAX\_QUEUE\_SIZE 15000

typedef int element;

typedef struct {

int front;

int rear;

element data[MAX\_QUEUE\_SIZE];

}QueueType;

void init\_queue(QueueType\* q) {

q->rear = -1;

q->front = -1;

}

int is\_full(QueueType q) {

return (q.rear == MAX\_QUEUE\_SIZE - 1);

}

int is\_empty(QueueType q) {

return (q.rear == q.front);

}

void enqueue(QueueType\* q, element item) {

if (is\_full(\*q)) {

printf("Queue is full.\n");

return;

}

q->data[++(q->rear)] = item;

}

element dequeue(QueueType\* q) {

if (is\_empty(\*q)) {

printf("Queue is empty.\n");

return -1;

}

return q->data[++(q->front)];

}

#define BUCKETS 10

#define DIGITS 4

void radix\_sort(int list[], int n)

{

int i, b, d, factor = 1;

QueueType queues[BUCKETS];

for (b = 0; b < BUCKETS; b++) init\_queue(&queues[b]); // 큐들의 초기화

for (d = 0; d < DIGITS; d++) {

for (i = 0; i < n; i++) // 데이터들을 자리수에 따라 큐에 입력

enqueue(&queues[(list[i] / factor) % 10], list[i]);

for (b = i = 0; b < BUCKETS; b++) // 버켓에서 꺼내어 list로 합친다.

while (!is\_empty(queues[b]))

list[i++] = dequeue(&queues[b]);

factor \*= 10; // 그 다음 자리수로 간다.

}

}

int main()

{

int merge\_list[MAX\_SIZE];

int quick\_list[MAX\_SIZE];

int radix\_list[MAX\_SIZE];

int n = MAX\_SIZE;

int randomitem;

srand(time(NULL));

for (int i = 0; i < n; i++) { // 난수 생성

randomitem = rand() % 10000000 + 1;

merge\_list[i] = randomitem;

quick\_list[i] = randomitem;

radix\_list[i] = randomitem;

}

clock\_t start1 = clock(); double result1;

merge\_sort(merge\_list, 0, n - 1); //합병정렬

clock\_t end1 = clock(); result1 = (double)(end1 - start1) / CLOCKS\_PER\_SEC;

printf("\n합병 정렬 시간 %lf \n", result1);

clock\_t start2 = clock(); double result2;

quick\_sort(quick\_list, 0, n - 1); // 퀵정렬

clock\_t end2 = clock(); result2 = (double)(end2 - start2) / CLOCKS\_PER\_SEC;

printf("\n퀵 정렬 시간 %lf \n", result2);

clock\_t start3 = clock(); double result3;

radix\_sort(radix\_list, MAX\_SIZE); // 기수정렬

clock\_t end3 = clock(); result3 = (double)(end3 - start3) / CLOCKS\_PER\_SEC;

printf("\n기수 정렬 시간 %lf \n", result3);

}

**과제 2.**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) )

#define MAX\_SIZE 10

int sorted[MAX\_SIZE];

void merge(int list[], int left, int mid, int right)

{

int i, j, k, l;

i = left; j = mid + 1; k = left;

// 분할 정렬된 list의 합병

while (i <= mid && j <= right) {

if (list[i] <= list[j]) sorted[k++] = list[i++];

else sorted[k++] = list[j++];

}

if (i > mid) // 남아 있는 레코드의 일괄 복사

for (l = j; l <= right; l++)

sorted[k++] = list[l];

else // 남아 있는 레코드의 일괄 복사

for (l = i; l <= mid; l++)

sorted[k++] = list[l];

// 배열 sorted[]의 리스트를 배열 list[]로 복사

for (l = left; l <= right; l++)

list[l] = sorted[l];

}

void print\_list\_merge(int list[]) {

for (int i = 0; i < MAX\_SIZE; i++)

printf("%d ", list[i]);

printf("\n");

}

void merge\_sort(int list[], int left, int right)

{

int mid;

if (left < right)

{

mid = (left + right) / 2; // 리스트의 균등분할

merge\_sort(list, left, mid); // 부분리스트 정렬

merge\_sort(list, mid + 1, right);//부분리스트 정렬

merge(list, left, mid, right); // 합병

print\_list\_merge(list);

}

}

void print\_list\_quick(int list[], int low, int high) {

for (int i = 0; i < MAX\_SIZE; i++)

printf(" %d ", list[i]);

printf("\n");

for (int i = 0; i < MAX\_SIZE; i++) {

if (list[i] == list[low])

printf("low ");

else if (list[i] == list[high])

printf("high");

else

printf(" ");

}

printf("\n");

}

int partition(int list[], int left, int right)

{

int pivot, temp;

int low, high;

low = left;

high = right + 1;

pivot = list[left];

do {

do

low++;

while (low <= right && list[low] < pivot);

do

high--;

while (high >= left && list[high] > pivot);

if (low < high) {

print\_list\_quick(list, low, high);

SWAP(list[low], list[high], temp);

}

} while (low < high);

print\_list\_quick(list, low, high);

SWAP(list[left], list[high], temp);

return high;

}

void quick\_sort(int list[], int left, int right)

{

if (left < right) {

int q = partition(list, left, right);

quick\_sort(list, left, q - 1);

quick\_sort(list, q + 1, right);

}

}

#define MAX\_QUEUE\_SIZE 100

typedef int element;

typedef struct {

int front;

int rear;

element data[MAX\_QUEUE\_SIZE];

}QueueType;

void init\_queue(QueueType\* q) {

q->rear = -1;

q->front = -1;

}

void queue\_print(QueueType\* q) {

for (int i = 0; i < MAX\_QUEUE\_SIZE; i++) {

if (i <= q->front || i > q->rear)

printf("#|");

else

printf("%d|", q->data[i]);

}

printf("\n");

}

int is\_full(QueueType q) {

return (q.rear == MAX\_QUEUE\_SIZE - 1);

}

int is\_empty(QueueType q) {

return (q.rear == q.front);

}

void enqueue(QueueType\* q, element item) {

if (is\_full(\*q)) {

printf("Queue is full.\n");

return;

}

q->data[++(q->rear)] = item;

}

element dequeue(QueueType\* q) {

if (is\_empty(\*q)) {

printf("Queue is empty.\n");

return -1;

}

return q->data[++(q->front)];

}

#define BUCKETS 10

#define DIGITS 3

void print\_radix\_list(int list[], int n) {

for (int i = 0; i < n; i++)

printf("%d ", list[i]);

printf("\n");

}

void radix\_sort(int list[], int n)

{

printf("초기값:\n");

int i, b, d, factor = 1;

QueueType queues[BUCKETS];

for (b = 0; b < BUCKETS; b++)

init\_queue(&queues[b]); // 큐들의 초기화

for (d = 0; d < DIGITS; d++) {

for (i = 0; i < n; i++) // 데이터들을 자리수에 따라 큐에 입력

enqueue(&queues[(list[i] / factor) % 10], list[i]);

print\_radix\_list(list, n);

for (b = i = 0; b < BUCKETS; b++) // 버켓에서 꺼내어 list로 합친다.

while (!is\_empty(queues[b]))

list[i++] = dequeue(&queues[b]);

factor \*= 10; // 그 다음 자리수로 간다.

}

}

int main(void)

{

int merge\_list[MAX\_SIZE];

int quick\_list[MAX\_SIZE];

int radix\_list[MAX\_SIZE];

int n = MAX\_SIZE;

int randomitem;

srand(time(NULL));

for (int i = 0; i < MAX\_SIZE; i++) { // 난수 생성

randomitem = rand() % 100 + 1;

merge\_list[i] = randomitem;

quick\_list[i] = randomitem;

radix\_list[i] = randomitem;

}

printf("초기값: ");

for (int i = 0; i < MAX\_SIZE; i++) { // 난수 생성

printf("%d ", merge\_list[i]);

}

printf("\n");

merge\_sort(merge\_list, 0, n - 1);

printf("==========================================\n");

printf("초기값: \n");

quick\_sort(quick\_list, 0, n - 1); // 퀵정렬 호출

printf("==========================================\n");

radix\_sort(radix\_list, MAX\_SIZE);

return 0;

}

**과제 3.**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX\_SIZE 10

#define MAX\_QUEUE\_SIZE 10

typedef int element;

typedef struct {

int front;

int rear;

element data[MAX\_QUEUE\_SIZE];

}QueueType;

void init\_queue(QueueType\* q) {

q->rear = -1;

q->front = -1;

}

int is\_full(QueueType q) {

return (q.rear == MAX\_QUEUE\_SIZE - 1);

}

int is\_empty(QueueType q) {

return (q.rear == q.front);

}

void enqueue(QueueType\* q, element item) {

if (is\_full(\*q)) {

printf("Queue is full.\n");

return;

}

q->data[++(q->rear)] = item;

}

element dequeue(QueueType\* q) {

if (is\_empty(\*q)) {

printf("Queue is empty.\n");

return -1;

}

element item = q->data[++(q->front)];

return item;

}

#define BUCKETS 10

#define DIGITS 3

void print\_bucket(QueueType\* queues) {

for (int i = 0; i < BUCKETS; i++) { //모든 큐에 대해 출력

printf("[%d]-> \t", i);

for (int j = queues[i].front + 1; j <= queues[i].rear; j++) {

printf("%d ", queues[i].data[j]);

}

printf("\n");

}

printf("\n");

}

void radix\_sort(int list[], int n)

{

int i, b, d, factor = 1;

QueueType queues[BUCKETS];

for (b = 0; b < BUCKETS; b++)

init\_queue(&queues[b]); // 큐들의 초기화

for (d = 0; d < DIGITS; d++) {

for (i = 0; i < n; i++) // 데이터들을 자리수에 따라 큐에 입력

enqueue(&queues[(list[i] / factor) % 10], list[i]);

print\_bucket(queues);

for (b = i = 0; b < BUCKETS; b++) // 버켓에서 꺼내어 list로 합친다.

while (!is\_empty(queues[b]))

list[i++] = dequeue(&queues[b]);

factor \*= 10; // 그 다음 자리수로 간다.

}

}

int main(void)

{

int radix\_list[MAX\_SIZE];

srand(time(NULL));

for (int i = 0; i < MAX\_SIZE; i++) { // 난수 생성

radix\_list[i] = rand() % 500 + 1;

}

radix\_sort(radix\_list, MAX\_SIZE);

return 0;

}