[R E P O R T]

정보통신공학전공

200301582

김성태

과제 : mergeSort 알고리즘 구현.

1. 목적

① 의사코드로 나와있는 mergeSort 알고리즘을 실제 프로그래밍 언어(C)를 이용하여 구현한다.

② mergeSort 알고리즘을 개량한 mergeSort_2 알고리즘을 실제 프로그래밍 언어(C)를 이용하여 구현한다.

③ 각각의 알고리즘에 대한 시간복잡도, 공간복잡도를 분석한다.

2. 필요환경

① GCC.

② Linux

3. 과제 수행 내역.

① Foundation of Algorithms p.53 ~ 55 예제 알고리즘 구현

② Foundation of Algorithms p.58 ~ 59 예제 알고리즘 구현

4. 과제 수행

① mergeSort_1 구현

소스

```
#include <stdio.h>

void merge(int h, int m, int *U, int *V, int *S);
void mergesort(int n, int *S);

int main()
{
        int i;
        int S[8] = {27, 10, 12, 20, 25, 13, 15, 22};

        mergesort(8, S);
```

```
	for(i = 0; i < 8; i++) {
		printf("%d, ", S[i]);
	}
	return 0;
}

/*************************************
* mergesort                          *
* n : 사이즈                         *
* S : 배열의 포인터                  *
* 크기 n의 배열 S를 합병정렬을 이용하여 정렬.  *
*  merge 함수와 함께 쓰여진다.       *
*************************************/
void mergesort(int n, int *S)
{
	int i;

	if(n > 1) {
		int h = n / 2;
		int m = n - h;

		int U[h], V[m];

		for(i = 0; i < h; i++)  {
			U[i] = S[i];
			// printf("U[%d] : %d ", i, U[i]);
		}
		// printf("\n");
		for(i = 0; i < m; i++) {
			V[i] = S[i + h];
			// printf("V[%d] : %d ", i +h, V[i]);
		}
		// printf("\n");

		mergesort(h, U);
		mergesort(m, V);

		merge(h, m, U, V, S);
	}
}

/*************************************
* merge                              *
* h : low value                      *
* m : high value                     *
```

```
* U : 병합을 하기원하는 배열.                    *
* V : 병합을 하기원하는 배열.                    *
* S : 병합이 되어 리턴되는 배열.                 *
* 합병 정렬시 실제로 합병이 이루어지는 부분      *
*****************************************/
void merge(int h, int m, int *U, int *V, int *S)
{
        int i, j, k;

        i = j = k = 0;

        while(i < h && j < m) {
                if(U[i] < V[j]) {
                        S[k] = U[i];
                        i++;
                }
                else {
                        S[k] = V[j];
                        j++;
                }
                k++;
        }
        if(i >= h) {
                for(; k < h + m; k++) {
                        S[k] = V[j];
                        j++;
                }
        }
        else {
                for(; k < h + m; k++) {
                        S[k] = U[i];
                        i++;
                }
        }
}
```

실행 화면

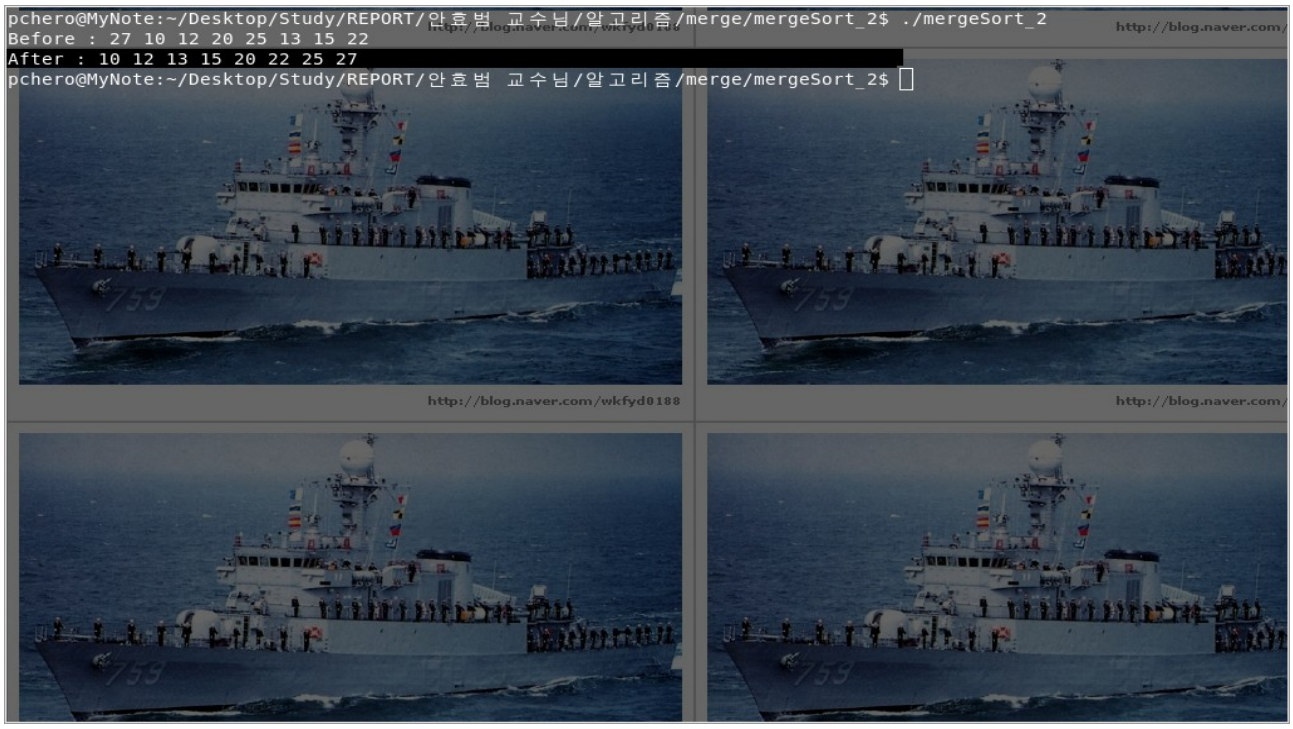

사진 *1: mergeSort_1* 실행화면

② mergeSort_2

소스

```
#include <stdio.h>

void merge2(int *S, int low, int mid, int high);
void mergesort2(int *S, int low, int high);

int count = 0;

int main()
{
	static int S[8] = {3, 4, 2, 5, 6, 8, 1, 7};
	int i;
	printf("before: ");
	for(i = 0; i < 8; i++)
		printf("%d ", S[i]);
	printf("\n");

	mergesort2(S, 0, 7);

	printf("\nafter: ");
```

```
	for(i = 0; i < 8; i++)
		printf("%d ", S[i]);
	printf("\n");
}

/***********************************************
* mergesort2                                   *
* S : 합병을 원하는 배열                       *
* low : 최하단 index값                         *
* high : 최상위 index값                        *
* 배열을 반으로 나누는 부분. merge2와 연동하여 정렬을 *
* 실행한다.                                    *
***********************************************/
void mergesort2(int *S, int low, int high)
{
	int mid;

	if(low < high) {
		mid = (low + high) / 2;
		printf("[%d] low : %d, mid : %d, high %d\n", œunt, low, mid, high);
		count++;
		mergesort2(S, low,mid);
		mergesort2(S, mid + 1, high);
		merge2(S, low, mid, high);
	}
}

/*****************************************
* merge2                                 *
* S : 정렬이 이루어지는 배열.            *
* low : 최하단값                         *
* mid : 중간값                           *
* high : 최상위값                        *
* 합병정렬시 실제 합병이 이루어지는 부분 *
*****************************************/
void merge2(int *S, int low, int mid, int high)
{
	int i, j, k;

	int U[high - low];

	i = low;
	j = mid + 1;
	k = 0;
```

```
	while(i <= mid && j <= high) {
		if(S[i] < S[j]) {
			U[k] = S[i];
			i++;
		}
		else {
			U[k] = S[j];
			j++;
		}
		k++;
	}
	if(i > mid)
		for(; j <= high; j++) {
			U[k] = S[j];
			k++;
		}
	else {
		for(; i <= mid; i++) {
			U[k] = S[i];
			k++;
		}
	}
//	for(i = 0; i < 8; i++)
//		printf("U[k] = %d\n", U[i]);

//	printf("\n");
	k = 0;
	for(i = low; i <= high; i++) {
		S[i] = U[k];
		k++;
	}

}
```

실행 화면

사진 *2: mergeSort_2* 실행화면

③ 시간복잡도 분석.

i = h이고, j = m – 1인 상태로 loop에서 빠져나가는 때가 최악의 경우로서, 이 때 기본동작의 실행 횟수는 h + m – 1이다. 따라서 최악의 경우 합병하는 시간복잡도는 W(h, m) = h + m – 1이다.

최악의 경우 수행시간은 W(h, m) = W(h) + W(m) + h + m – 1이 된다. 여기서 W(h)는 U를 정렬하는데 걸리는 시간, W(m)은 V를 정렬하는데 걸리는 시간, 그리고 h + m – 1은 합병하는데 걸리는 시간이다. 정수 n 을 2^k(k >= 1)이라고 한다면, h = n / 2, m = n / 2이 된다. 따라서 최악의 경우 시간복잡도는

W(n) = 2W(n / 2) + n – 1 n > 1이고, n = 2^k (k> = 1)

W(1) = 0 왜냐하면 합병이 전혀 이루어지지 않으므로.

④ 공간복잡도 분석.

mergeSort_1의 경우 새로운 반복적인 루프를 돌때마다 새로운 배열을 생성한다. 즉 추가적인 저장장소가 필요하다. 수식으로 나타내면...n + (n / 2) + (n / 4) + ... = 2n이다.

하지만 mergeSort_2의 경우 반복적인 루프를 돌아도 새로운 배열을 생성하지 않는다. 즉, 공간복잡도는 n이다.

5. 참조 사이트

① http://kldp.org : 한국 리눅스 문서 프로젝트

② http://phpschool.com : PHP 사용자 포럼