알고리즘 실습(2주차)

201000287 일어일문학과 유다훈

1. 과제 설명 및 해결 방법

1. Merge sort

재귀식이 아닌 반복문을 이용하여 합병 정렬 구현하기

C++의 clock()함수를 이용하여 시간측정 및 그래프 확인

2. Bubble sort

버블정렬 구현하기

C++의 clock()함수를 이용하여 시간측정 및 그래프 확인

3. 이론 과제

손으로 작성.

교과서 참고

2. 주요 부분 코드 설명(알고리즘 부분)

1. Merge sort

```
    //정렬할때 이용할 임시배열.
//    int tempArray[10];
//    int tempArray[100];
//    int tempArray[1000];
    int tempArray[10000];
```

- 합병 정렬은 원소가 한 개가 될 때까지 쪼갠 후 정렬해가며 다시 합치는 정렬방법이다.
- 정렬해가는 과정에서 이용하기 위해 정렬할 때 사용할 데이터 크기 만큼의 임시 배열을 선언한다.

```
//사용할 변수들 선언
int right, rightEnd;
int i,j,m;

start = clock();
//반복하며 정렬 시작

for (int k=1; k < arrayLength; k = k*2 ){

    for (int left = 0; left+k < arrayLength; left = left + (k*2)){ //비교대상 왼쪽값 지정
        right = left + k; //비교대상의 오른쪽값 지정
        rightEnd = right + k; //비교대상 오른쪽값의 끝값.
        if (rightEnd > arrayLength)
            rightEnd = arrayLength;
        m = left; //왼쪽값을 m
        i = left; //실제 배열 정렬 시에 쓰이는 왼쪽값.
        j = right; //오른쪽값.

        while (i < right && j < rightEnd) { //i가 오른쪽보다 작고, j가 오른쪽 끝값보다 작을때. 비교대상범위가 인덱스 밖을 벗어나지 않을 때.
            if (array[i] <= array[j]) { //i가 j보다 작거나 같다면?
                tempArray[m] = array[i]; //임시배열에 i를 넣어줌. 오름차순 정렬이기때문에 작은값이 먼저 들어가야한다.
                i++; //왼쪽배열 인덱스값 증가.
            } else {
                tempArray[m] = array[j]; //그렇지 않다면, 오른쪽배열값을 넣어줌.
                j++; //오른쪽배열 인덱스값 증가.
            }
            m++; //임시배열 인덱스 증가.
        }
        while (i < right) { //첫번째 while문을 끝내고, 왼쪽배열에 데이터가 아직 남아있는 경우.
            tempArray[m] = array[i]; //왼쪽 배열의 데이터를 그냥 차례로 임시배열에 넣어줌.
            i++;
            m++;
        }
        while (j < rightEnd) { //첫번째 while문을 끝내고, 오른쪽 배열에 데이터가 아직 남아있는 경우.
            tempArray[m] = array[j]; //오른쪽 배열의 데이터를 그냥 차례로 임시배열에 넣어줌.
            j++;
            m++;
        }
        for (m=left; m < rightEnd; m++) { //임시배열에는 이번 정렬에서 넣은 오름차순의 데이터값이 있음.
                                          //이것을 원래 배열로 순서대로 넣어줌.
            array[m] = tempArray[m];
        }

    }
}

end = clock();
sortingTime = (double)(end - start) / CLOCKS_PER_SEC; //시간측정
```

- 외부 for문은 변수 k의 값에 *2씩 증가한다.
- 내부 for문은 left값에 k*2를 더한 값이 증가한다.
  - 내부 for문에서 쓰이는 right값과 rend값은 각각 오른쪽 값과 오른쪽 끝 값이다.
  - 내부 for문에서 2개의 비교대상데그룹이 정렬을 한다. 정렬 범위는 left에서 right-1값과 right에서 rend-1값으로 두 개의 범위이다.

- 두 개의 정렬 대상 데이터그룹이지만 각각의 그룹의 범위는 넓어진다.
  - 처음에는 0과 1과 같은 한 개씩이지만 외부 for문이 한 바퀴 돌고나면 0~1, 2~3 과 같은 식으로 증가한다.
- 첫 번째 while문은 비교를 하여 작은 값을 먼저 임시 배열에 삽입한다.
  - 비교 데이터 범위가 오른쪽 그룹이나 왼쪽 그룹의 범위를 넘어가게 된다면, while문을 빠져나온다.
- 두 번째 while문은 왼쪽 비교 그룹의 데이터가 전부 임시배열에 들어가고 남은 오른쪽 비교그룹의 데이터를 임시배열에 넣는 것이다.
- 세 번째 while문은 오른쪽 데이터그룹의 데이터가 전부 임시 배열에 들어가고 남은 왼쪽 비교대상그룹의 데이터를 임시 배열에 넣는다.
- 마지막 for문은 왼쪽 비교대상 그룹 첫번째 데이터부터 오른쪽 비교대상그룹 마지막 데이터까지의 임시 배열을 원래 데이터 배열에 넣는다.

- 위의 작업을 k*2의 증가값으로 입력 받은 데이터 범위 길이까지 증가시켜준다. 즉, 첫 회전에서는 원래 배열에서 1개 1개씩을 골라 왼쪽, 오른쪽으로 지정하여 임시 배열을 이용하여 정렬 후 원래 배열에 다시 넣어준다. 내부 for문이 다 끝나면, 그 다음 회전부터는 왼쪽과 오른쪽의 범위를 2의 배수만큼 늘려 나가며 정렬한다.

2. Bubble sort

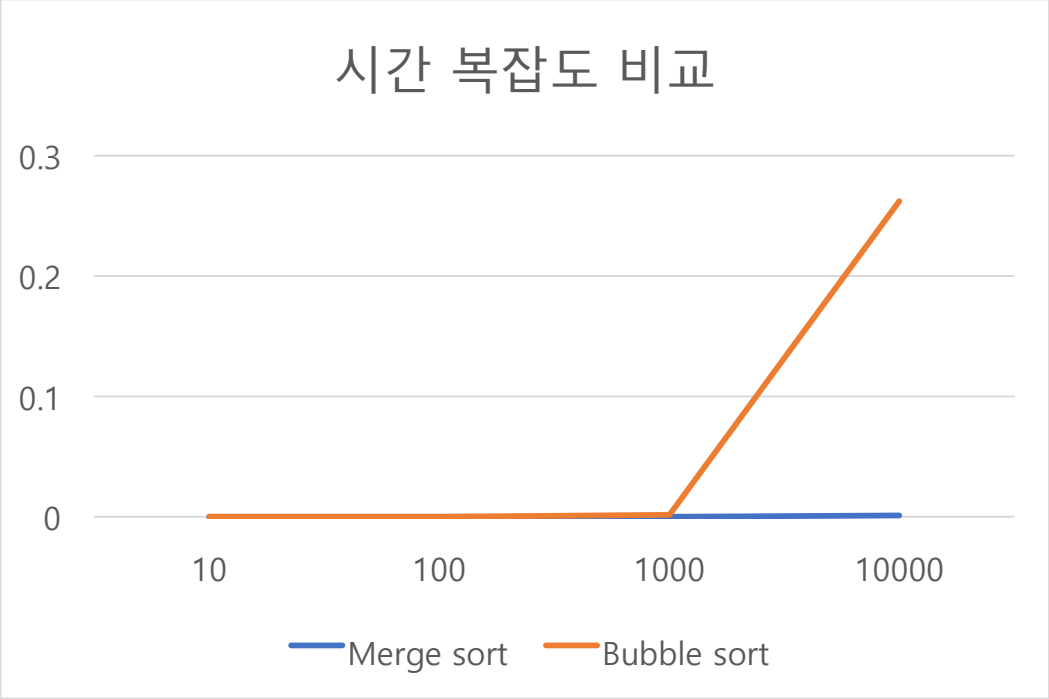
```
/* Bubble sort */

start = clock();

for(int i = 0; i < arrayLength-1; i++ ) { //정렬대상 마지막 원소의 직전 데이터까지.

    for(int z = 0; z <= (arrayLength-i); z++) { //전체 길이에서 i를 뺀 값.
        if(array[z] > array[z+1]) { //앞값이 뒷값보다 크다면
            int temp = array[z]; //임시 변수를 이용하여 자리를 스왑해줌.
            array[z] = array[z+1];
            array[z+1] = temp;
        }

    }
}

end = clock();
sortingTime = (double)(end - start) / CLOCKS_PER_SEC; //시간측정

/* Bubble sort end */
```

- 버블 정렬은 n번째 원소와 n+1번째 원소를 비교하여 정렬하는 과정을 반복하는 정렬이다.
- 외부 반복문은 전체배열 길이의 -2값까지만 실행된다. n번째 원소와 n+1원소를 비교하는 것이므로, 실제로는 마지막 원소의 직전 원소개

수까지만 반복하면 된다.

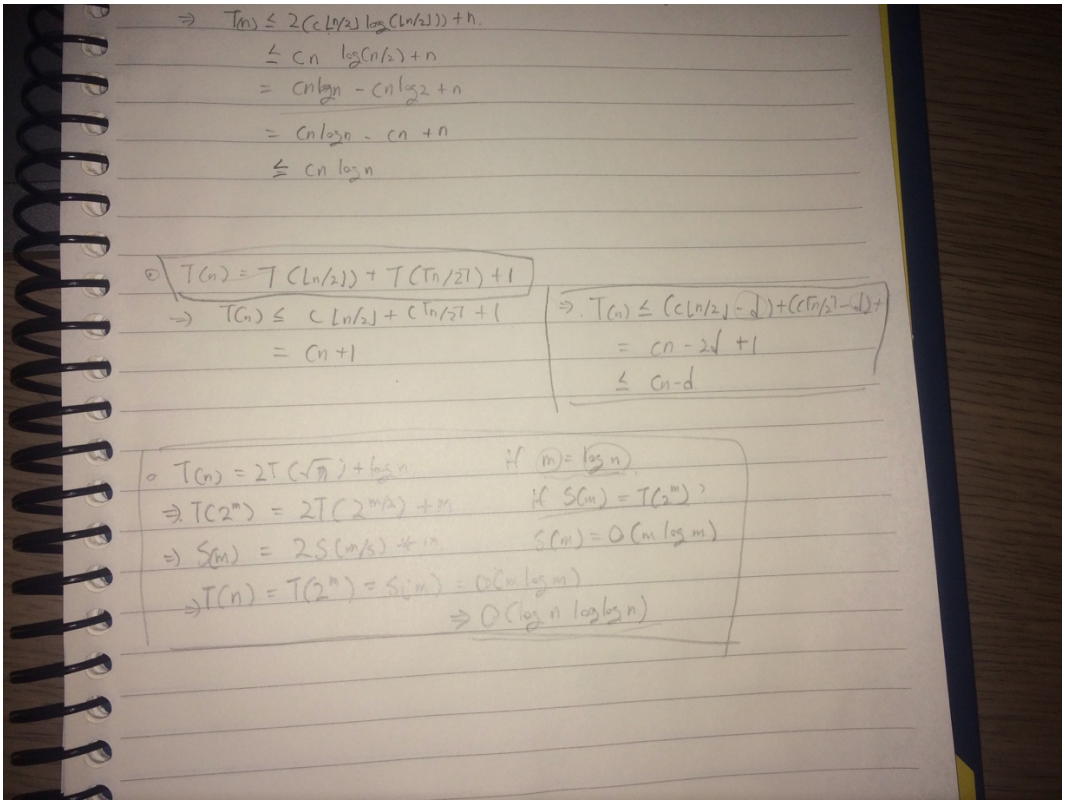
- 내부 반복문은 원소를 비교하여 정렬하는 반복문이다. n번째 원소값이 크면 n+1번째 값과 자리를 교환해준다.
  - 이렇게 비교할 경우 제일 큰 값이 제일 뒤로 가게 된다. 이 다음 정렬부터는 맨 뒷자리를 제외한 나머지값들을 정렬시켜야 하므로 배열의 길이에서 외부 반복 변수 i만큼을 뺀 나머지 값까지만 반복한다.

3. 결과(시간 복잡도 포함)



- Merge sort : O(n logn)
- Bubbble sort : O(n^2)
- 데이터 크기가 작을 때는 두 정렬 모두 큰 차이가 없으나, 데이터가 늘어남에 따라 버블 정렬은 n에 제곱에 비례한 시간이 걸리고, 합병 정렬은 n log n만큼의 시간이 걸렸다.
- 합병 정렬이 매우 우수하고 빠른 정렬방법이다.
- 이 그래프는 이론 시간에 확인한 수업자료에서의 그래프의 모양과도 일치한다.

4. 이론 과제



위의 네모 칸 두 개가 첫 번째 문제.

아래의 네모 칸이 두 번째 문제입니다.