**프로그래밍 실습 #4**

2023년 10월 1주차

□ 다음 정렬 알고리즘을 ADL로 작성하고 파이썬으로 구현해 보라.

(1) 칵테일 쉐이커 정렬 알고리즘

컴퓨터과학자 도널드 누스(Donald Knuth)가 제안한 **칵테일 쉐이커 정렬(cocktail shaker sort)**은 버블 정렬을 수정하여 매번 반복할 때마다 방향을 바꿔가며 원소의 위치를 결정하는 기법이다. 루프를 반복할 때마다 한 번은 가장 큰 원소를 가장 오른쪽 위치로 보내고 그 다음에는 가장 작은 원소를 가장 왼쪽 위치로 보내고, 계속해서 이런 방법으로 수행된다. 이 알고리즘이 수행되는 예는 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | [1] | [2] | [3] | [4] | [5] | [6] |
| 최초 배열 | 6 | 5 | 4 | 3 | 2 | 1 |
| 첫 번째 패스 종료 후 | 5 | 4 | 3 | 2 | 1 | 6 |
| 두 번째 패스 종료 후 | 1 | 5 | 4 | 3 | 2 | 6 |
| 세 번째 패스 종료 후 | 1 | 4 | 3 | 2 | 5 | 6 |
| 네 번째 패스 종료 후 | 1 | 2 | 4 | 3 | 5 | 6 |
| 다섯 번째 패스 종료 후(완료) | 1 | 2 | 3 | 4 | 5 | 6 |

칵테일 쉐이커 정렬 알고리즘을 ADL로 작성한 다음 파이썬으로 구현하여 버블 정렬과 성능을 비교해 보라.

--------------------------------------------------------------------

cocktailShakerSort(a[], n)

d ← true; i ← 1; k ← n;

|  |
| --- |
|  |

}

end cocktailShakerSort()

--------------------------------------------------------------------

(2) 교환 정렬

**교환 정렬(exchange sort)**은 버블 정렬과 선택 정렬과 비슷하게 메인 루프의 각 반복에서 가장 큰 원소를 가장 왼쪽으로 보내어 역순으로 정렬한다. 이것은 정렬되지 않은 원소 각각을 si와 비교하여, 이 원소가 si보다 크면 si와 교환하는 방식으로 수행된다. 이 알고리즘이 수행되는 예는 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | [1] | [2] | [3] | [4] | [5] | [6] |
| 최초 배열 | 3 | 1 | 2 | 4 | 6 | 5 |
| 첫 번째 패스 종료 후 | 6 | 1 | 2 | 3 | 4 | 5 |
| 두 번째 패스 종료 후 | 6 | 5 | 1 | 2 | 3 | 4 |
| 세 번째 패스 종료 후 | 6 | 5 | 4 | 1 | 2 | 3 |
| 네 번째 패스 종료 후 | 6 | 5 | 4 | 3 | 1 | 2 |
| 다섯 번째 패스 종료 후(완료) | 6 | 5 | 4 | 3 | 2 | 1 |

교환 정렬 알고리즘을 ADL로 작성한 다음 파이썬으로 구현하여 버블 정렬 및 선택 정렬과 성능을 비교해 보라.

**교환 정렬은 한 원소를 잡고 “전체” 와 비교하여 조건이 성립하는 경우 교환 실행.**

--------------------------------------------------------------------

exchangeSort(a[], n)

|  |
| --- |
|  |

end exchangeSort()

--------------------------------------------------------------------

(3) 자연 합병 정렬 알고리즘

합병 정렬 알고리즘 mergeSort()에서 합병 단계는 이미 정렬이 된 부분 배열들을 반복해서 합병한다. 합병 정렬의 또 다른 하향식(top-down) 버전은 주어진 배열 속에 이미 정렬되어 있는 부분 배열, 즉 런(run)을 합병함으로써 이 문제를 해결한다. 배열에서 런이 결정되면, 차례대로 두 개의 런에 대한 합병이 수행된다. 예를 들어 배열 [6, 7, 8, 3, 4, 1, 5, 9, 10, 2]에서 런을 구하면 다음과 같다.

[6, 7, 8], [3, 4], [1, 5, 9, 10], [2]

먼저 런 [6, 7, 8]과 런 [3, 4]를 합병하면 [3, 4, 6, 7, 8]이 되고, 다음으로 런 [1, 5, 9, 10]과 런 [2]을 합병하면 [1, 2, 5, 9, 10]이 된다. 마지막으로 런 [3, 4, 6, 7, 8]과 [1, 2, 5, 9, 10]을 합병하면 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]이 된다. 이 알고리즘을 파이썬으로 구현하고 합병 정렬과 성능을 비교해 보라. 주어진 배열의 부분 정렬(즉 런)을 이용하는 합병 정렬을 **자연 합병 정렬(natural merge sort)**이라 한다.

**RUN 존재.**

**Run: 배열의 이미 정렬이 되어있는 부분**

**장점: 이미 정렬이 되어있는 경우, Run 생성 후 종료. (매우 빠름)**

**단점: 역순으로 정렬되어있는 경우 run이 원소의 개수 만큼 생성됨. (느림)**

**각 Run 을 합병해나감.**

먼저 입력 배열에서 런을 구하는 makeRun() 함수에 대한 ADL을 작성하라.

--------------------------------------------------------------------

makeRun(a[], n)

i ← 1;

r ← []; // 공백 리스트

|  |
| --- |
|  |

return r;

end makeRun()

--------------------------------------------------------------------

makeRun() 알고리즘을 파이썬으로 구현한 다음 실행한 결과는 다음과 같다.

--------------------------------------------------------------------

[0, 6, 7, 8, 3, 4, 1, 5, 9, 10, 2]

[[6, 7, 8], [3, 4], [1, 5, 9, 10], [2]]

--------------------------------------------------------------------

makeRun() 함수를 사용하여 자연 합병 정렬을 수행하는 파이썬 프로그램을 작성한 다음, 합병 정렬과 성능을 비교해 보라.

**<코딩 테스트 연습 #2>**

**□ 나머지 합**

[문제]

수 N개 A1, A2, ..., AN이 주어진다. 이때, 연속된 부분 구간의 합이 M으로 나누어 떨어지는 구간의 개수를 구하는 파이썬 프로그램을 작성하라.

즉, Ai + ... + Aj (i ≤ j) 의 합이 M으로 나누어 떨어지는 (i, j) 쌍의 개수를 구해야 한다.

[입력]

- 첫째 줄에 N과 M이 주어진다. (1 ≤ N ≤ 106, 2 ≤ M ≤ 103)

- 둘째 줄에 N개의 수 A1, A2, ..., AN이 주어진다. (0 ≤ Ai ≤ 109)

[출력]

첫째 줄에 연속된 부분 구간의 합이 M으로 나누어 떨어지는 구간의 개수를 출력한다.

[실행 예]

--------------------------------------------------------------------

4 2

1 2 3 1

4

**구간 합 = 1367**

**2로 나누었을 때 나머지 = 1101**

**1의 개수 (3C2) + 0의 개수 (1) = 4**

--------------------------------------------------------------------

--------------------------------------------------------------------

5 3

1 2 3 1 2

7

--------------------------------------------------------------------