Journal of The Korean Association
of Information Education Vol. 22, No. 6,
December 2018, pp. 671-679

http://dx.doi.org/10.14352/jkaie.2018.22.6.671

컴퓨터과학 교육용 정렬 놀이를 위한 실험적 분석

박영기

춘천교육대학교 컴퓨터교육과

요 약

CSUnplugged에 나타난 교육용 정렬 놀이는 만 8세 이상이면 할 수 있지만, 학생들을 지도하기에 쉬운 활동은 아니다. 왜냐하면 (1) 좋은 정렬 방법을 찾는 것은 컴퓨팅 사고력이 뛰어난 전공자라 하더라도 어려울 수 있고, (2) 정렬 알고리즘의 수가 많아 모든 내용을 파악하기가 어렵기 때문이다. 또, (3) 우수한 성능을 나타낸다고 알려져 있는 정렬 알고리즘들이 교육용 정렬 놀이에서는 반드시 좋은 결과를 만들어 내지도 않는다. 본 논문에서는 정렬 놀이를 할 때 어떤 알고리즘이 더 효과적인지 분석하고, 교수자가 알아야 하는 내용이 무엇인지에 대해 논의한다.

키워드 : 정렬 알고리즘, 컴퓨팅 사고, 놀이 활동, 삽입 정렬, 빠른 정렬

An Experimental Analysis on the Unplugged Sorting Activity for Computer Science Education

Youngki Park

Department of Computer Education, Chuncheon National University of Education

ABSTRACT

Sorting algorithms are the basic building blocks that computer science students need to learn. In recent years, sorting algorithms also have begun to be taught in K-12 classrooms using “the educational sorting game” described in CSUnplugged. However, although the educational sorting game was developed for students aged 8 and up, it is hard for K-12 teachers to play with their students because it is difficult for teachers to understand all of the algorithms and some popular algorithms do not work well in the educational sorting game. In this paper, we discuss what teachers should know, and experimentally analyze the performance of the existing algorithms when applied to the educational sorting game.

Keywords : Sorting algorithm, CSUnplugged, Sorting activity, Quicksort, Insertion sort

논문투고 : 2018-12-18
논문심사 : 2018-12-24
심사완료 : 2018-12-26

1. 서론

정렬 알고리즘은 컴퓨터과학을 전공하는 학생이라면 누구나 배우게 되는 기본적인 컴퓨터과학 지식이다. 대학교 교육과정에서는 보통 정렬 알고리즘을 배울 때, 이 알고리즘이 어떻게 동작하는지를 배움과 동시에 알고리즘을 분석하는 방법(시간 및 공간 복잡도 등)과 프로그래밍 언어를 이용하여 구현하는 실습을 병행하게 된다.

최근에는 초중등 교육에서도 정렬 알고리즘을 가르치는 사례가 늘고 있는데, 이 경우에는 정렬 알고리즘을 구현하기보다 그 과정을 이해하는 데에 초점이 맞춰져 있다. 즉, 프로그램 코드를 제시하기보다는 CSUnplugged와 같은 사이트에 나타난 놀이를 통해 교육을 하는 방식이다. 이 활동은 초등학교 교육과정에서 제시하는 '순차, 선택, 반복' 알고리즘을 가르치는 데에도 효과적일 수 있는데, 대부분의 정렬 알고리즘들은 순차, 선택, 반복 구조를 활용하기 때문이다.

CSUnplugged에 나타난 정렬 놀이는 만 8세 이상이 할 수 있다고 명시되어 있고, 국내 초등학교 저학년 학생들을 대상으로도 수행할 수 있다는 연구 결과가 제시되어 있다[11]. 그러나 학생들과 놀이를 할 때 교사는 몇 가지 어려움을 겪을 수 있다. 그 이유는 (1) 첫째, 정렬 알고리즘의 종류가 많기 때문에 교사가 학생들을 지도하기 전에 학습 부담을 가지게 된다. (2) 또, 학생들이 기존에 알려져 있지 않은 창의적인 아이디어로 새로운 알고리즘을 제안할 경우, 그 알고리즘의 성능이 어느 정도인지 가늠하기 쉽지 않다. (3) 마지막으로, 컴퓨터과학 전공자가 정렬 알고리즘을 교사에게 자세히 설명해 준다고 하더라도, 프로그램으로 구현했을 때 우수한 성능을 내는 알고리즘이 정렬 놀이에서 반드시 뛰어난 성능을 보장하지는 않는다.

본 논문에서는 CSUnplugged에 나타난 정렬 놀이를 실험적으로 분석한 결과를 제시하였다. 이 결과를 숙지함으로써 교수자는 학생들과 정렬 놀이를 할 때 더 효과적으로 지도하고 안내할 수 있을 것이라 기대된다. 구체적으로, 본 논문이 공헌한 바는 다음과 같다.

인기도를 기반으로 하여 교수자가 알아두어야 할 알고리즘이 어떤 것인지 정리하였다(3.1절).

컴퓨터과학 분야에서 우수한 정렬 알고리즘과 정렬 놀이에서 우수한 결과를 내는 알고리즘이 어떻게 다르고 같은지 비교분석하였다(3.2절, 3.3절).

정렬 놀이의 알고리즘들을 성능에 따라 그룹화하고, 각 그룹의 알고리즘이 같은 그룹의 다른 알고리즘과 비교했을 때 성능 차이가 어느 정도인지 분석하였다(3.4절).

각 알고리즘의 프로그램 코드를 통해 학생들에게 가르쳐주기 쉬운 알고리즘이 어떤 것인지 분석하였다(3.5절).

2. 관련 연구

2.1 정렬 놀이 활동

CSUnplugged에 나타난 정렬 놀이 활동의 규칙은 다음과 같다[1]. 먼저 8개의 물체를 준비한다. 이 물체들은 겉으로 보기에는 무게를 가늠할 수 없지만, 실제 무게는 모두 다르다. 이 8개의 물체를 양팔저울을 이용하여 무게 순으로 정렬해야 한다. 양팔저울에는 각각 하나의 물체만 올려둘 수 있다. 최소의 횟수로 비교하면서 물체들을 정렬하는 방법을 찾는 것이 이 놀이의 목표다.

이 정렬 놀이는 2개씩 비교하며 정렬하는 활동으로, 이와 관련된 정렬 알고리즘들은 병합 정렬(Merge sort), 빠른 정렬(Quicksort) 등이 있다. 우리가 코딩을 할 때 일반적으로 사용하는 정렬 방식이다. 그러나 2개씩 비교하며 정렬하지 않는, 다른 방식의 컴퓨터과학 정렬 놀이들도 있을 수 있다. 예를 들어 물체의 상대적인 무게만을 아는 것이 아니라, 모든 물체의 무게를 알고 있는 상태에서 정렬하는 놀이가 있을 수 있고, 이때에는 기수 정렬(Radix sort) 등을 사용하면 빠르게 정렬할 수 있다.

본 논문에서는 2개씩 비교하며 정렬하는, CSUnplugged 방식의 정렬 활동에 대해 한정지어 논하기로 한다.

2.2 정렬 문헌 연구

놀이를 통해 컴퓨터과학을 배우는 다양한 활동들이 있다[2, 3, 4, 5, 6, 7, 10, 13, 14, 16]. 정렬 놀이와 관련된 연구도 수행되고 있는데, 정보교육학회논문지를 중심으로 하여 이와 관련된 문헌들을 아래에 정리하였다.

임민영 외 2인의 논문[12]에서는 초등학생들을 대상

<Table 1> The sorting algorithms ranked by their "popularity", which is calculated as a number of documents retrieved by their search keywords

#Doc.	Algorithm	#Doc.	Algorithm	#Doc.	Algorithm
1,080,000	**Quicksort(C)**	103,000	Block sort (C)	19,700	Patience sorting (C)
873,000	**Bubblesort(C)**	59,000	Timsort (C)	18,300	Pigeonhole sort (N)
786,000	**Mergesort(C)**	58,500	Library sort (C)	17,600	**BinaryInsertionsort(C)**
680,000	**Insertionsort(C)**	51,300	**Odd-evensort(C)**	13,600	Gnome sort (C)
454,000	**Selectionsort(C)**	38,600	Bogosort (O)	12,500	Binary tree sort (C)
307,000	**Heapsort(C)**	31,200	Strand sort (C)	10,500	Tournament sort (C)
215,000	Radix sort (N)	24,600	Cycle sort (C)	10,100	Bitonic sorter (O)
151,000	**Shellsort(C)**	23,300	Introsort (C)	9,990	Stooge sort (O)
133,000	Bucket sort (N)	23,200	**Combsort(C)**	9,600	Bead sort (O)
130,000	Counting sort (N)	20,400	Cocktail sort (C)	8,960	Flashsort (N)

으로 지도하여야 할 검색과 정렬 알고리즘이 무엇인지 정리하고, 이를 가르치는 것이 효과적인지를 분석하였다. 교육 내용으로 삽입정렬(Insertion sort)과 거품정렬(Bubble sort)이 선정되었다. 주로 숫자카드를 사용하여 놀이를 진행하였다.

정인기의 논문[9]에서는 코딩을 통해 정렬 알고리즘을 배울 수 있도록 수업 내용을 설계하였다. CSUnplugged와 같이 놀이를 통해 정렬을 배우기보다는, '코딩 놀이'를 통해 학생들이 직접 정렬 방법들을 고안해 내도록 유도했다.

이용배, 이영미의 논문[11]에서는 놀이 활동을 중심으로 하여 정렬 알고리즘을 가르치는 방식과, 정렬하는 과정을 애니메이션으로 보여줌으로써 가르치는 방식 중 어떤 방법이 교육적으로 더 우수한지에 대해 연구하였다. 놀이 활동은 우유팩과 양팔저울을 사용하는 방식처럼 CSUnplugged의 내용을 변형한 경우도 있었지만, 캐릭터 카드를 이름순으로 정렬하는 등 별도로 제안한 놀이 활동도 있었다.

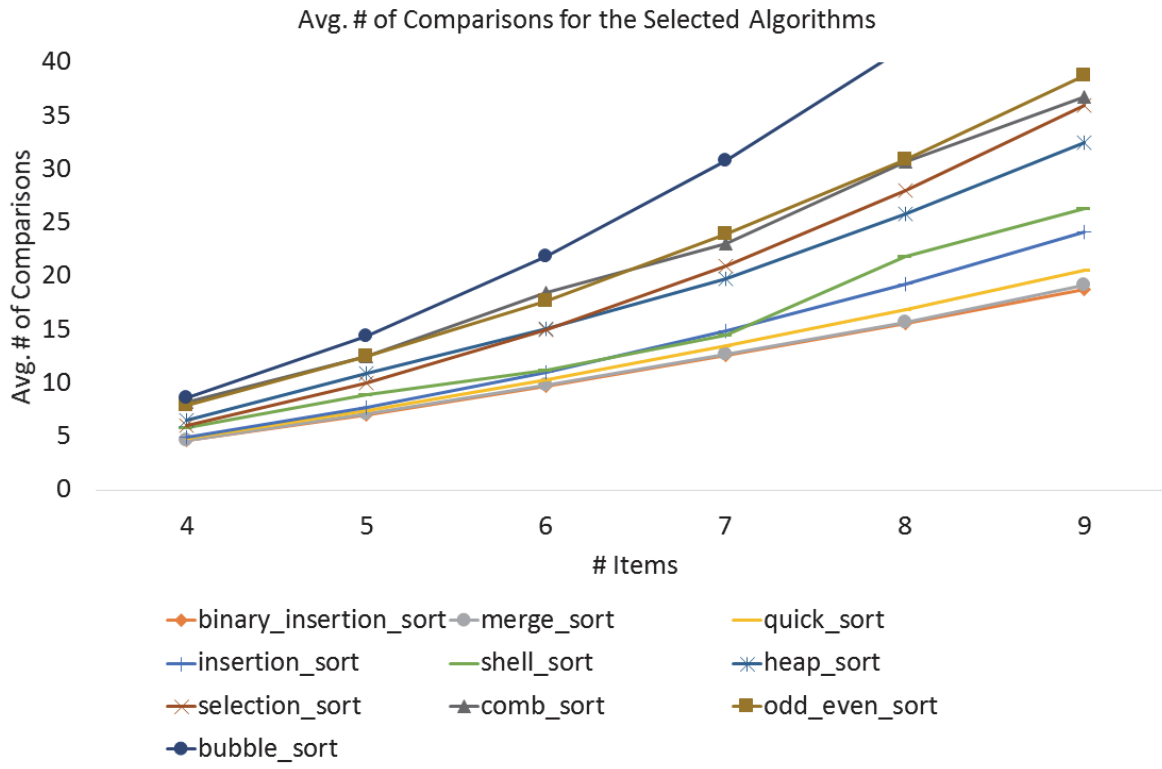
장정훈, 김종우의 논문[8]에서는 다양한 종류의 정렬 알고리즘을 여러 정렬 놀이를 통해 학생들에게 설명하였다. 학생들에게 알려준 정렬 알고리즘은 선택정렬(Selection sort), 거품정렬, 삽입정렬, 병합정렬, 빠른정렬 등이 있다.

3. 정렬 알고리즘 분석

본 절에서는 CSUnplugged에 나타난 놀이를 사용한다. 다만 설명의 편의상 무게를 재는 물체 대신 카드를 사용하여 놀이하는 것으로 가정한다. 예를 들어 6개의 카드를 사용할 경우, 1, 2, 3, 4, 5, 6을 사용하고, 8개의 카드를 사용할 경우, 1, 2, 3, 4, 5, 6, 7, 8이라 적힌 카드를 오름차순으로 정렬하는 것이 목적이다. 단, CSUnplugged의 규칙과 동일하게 2개씩만 비교할 수 있고, 이때 상대적인 크고 작음을 확인할 수는 있지만 카드의 내용은 확인할 수 없다고 가정한다.

3.1 교육에 적합한 알고리즘의 선정

학생들에게 어떤 알고리즘을 가르쳐야 하느냐에 대해서는 정답이 없다. 그렇지만 어떤 알고리즘이 더 유명하고, 많이 사용되는지를 교수자가 미리 알고 있다면, 놀이 활동을 할 때 도움이 될 것이다. Table 1은 위키피디아 페이지에 나타난 정렬 알고리즘들을 구글에 검색했을 때, 검색된 문서의 수가 많은 순서로 각 알고리즘들을 나열한 것이다. 괄호 안에 C라고 표기된 것은 2개씩 비교하며 정렬하는 방식을 뜻하며, CSUnplugged에 나타난 정렬 놀이에 적합한 알고리즘이다. 괄호 안에 N이라고 표기된 것은 반드시 2개씩 비교하며 정렬할 필요가 없는 방식을 뜻하며, CSUnplugged의 정렬 놀이 활동과는 어울리지 않는 알고리즘이다. 괄호 안에 O라고 표기된 것은 너무 느리거나 실용적이지는 않지만 교육적으로 활용될 가치는 있는 알고리즘을 뜻한다. C, N, O로 나타낸 기준은 위키피디아 페이지의 내용을 따랐다[15].



(Fig. 1) The average number of comparisons for the selected ten algorithms

이 표에서 가장 순위가 높은 10개의 알고리즘 중 우리가 사용할 정렬 놀이에 적합한(C로 분류된) 알고리즘들을 1차적으로 선정했다. 이 알고리즘들은 총 7가지로, 각각 빠른 정렬, 거품 정렬, 병합 정렬, 삽입 정렬, 선택 정렬, 힙 정렬, 쉘 정렬이다.

그리고 상위 10개의 알고리즘에 포함되지는 못했지만, 학생들이 스스로 생각해내기 쉬운 3개의 알고리즘을 추가로 선정하였다. 홀짝 정렬(Odd-even sort), 빗질 정렬(Comb sort), 이진 삽입 정렬(Binary Insertion sort)이 그것이다. 홀짝 정렬은 정렬 놀이를 하기 전에 정렬 네트워크 활동을 한 경우, 학생들이 쉽게 생각해낼 수 있는 방식이다. 정렬 네트워크로도 활동할 수 있기 때문에 학생들에게 교육하기에도 적절하다. 빗질 정렬은 거품 정렬을 먼저 교육한 경우, 창의적인 학생들이 생각해내기 쉽다. 이진 삽입 정렬의 경우에도 삽입 정렬과 이진 검색을 배운 학생들이 생각해 내는 경우가 많아 선정하였다.

선정된 10개의 알고리즘들은 Table 1에서 굵은 색으로 표기하고 다른 실험을 통해 분석을 진행하였다. 각 알고리즘들은 구현 방식에 따라 조금씩 성능이 다를 수 있는데, 예를 들어 거품 정렬의 경우 다양한 최적화 방식이 존재하며 어떤 최적화 방법을 사용하느냐에 따라 성능이 크게 달라진다. 따라서 가급적 가장 널리 알려져 있는 프로그램 코드[1]를 사용하여 실험하였다.

1) 이진 삽입 정렬, https://www.geeksforgeeks.org/binary-insertion-sort/
병합 정렬, https://www.geeksforgeeks.org/merge-sort/
빠른 정렬, https://www.geeksforgeeks.org/python-program-for-quicksort/
삽입 정렬, https://www.geeksforgeeks.org/insertion-sort/
쉘 정렬, https://www.geeksforgeeks.org/shellsort/
힙 정렬, https://www.sanfoundry.com/python-program-implement-heapsort/
선택 정렬, https://www.geeksforgeeks.org/python-program-for-selection-sort/
빗질 정렬, https://www.geeksforgeeks.org/comb-sort/

3.2 카드 수에 따른 비교 횟수

Fig. 1은 카드 수에 따라 평균적으로 얼마나 비교를 하게 되는지를 나타낸 것이다. 즉, 나타날 수 있는 모든 순열마다 비교 횟수를 계산하고, 이를 평균한 것으로, 균등 분포를 가정하였다고 볼 수 있다. 보통 6장이나 8장의 카드를 사용하여 정렬 놀이를 하게 되는데, 이때 각 알고리즘의 평균 비교 횟수을 알고 있다면 학생들을 지도할 때 도움이 될 것이다.

선정된 10개의 알고리즘들은 카드 개수에 따라 비교 횟수가 다르게 나타난다. 그렇지만 Fig. 1에 따르면 전반적으로 비교 횟수가 상대적으로 적은 알고리즘은 (1) 이진 삽입 정렬(축약하여 BI라 표기), (2) 병합 정렬(ME), (3) 빠른 정렬(QU), (4) 삽입 정렬(IN), (5) 쉘 정렬(SH), (6) 힙 정렬(HE), (7) 선택 정렬(SE), (8) 빗질 정렬(CO), (9) 홀짝 정렬(OD), (10) 거품 정렬(BU)의 순서로 나타났다. 즉, 이진 삽입 정렬이 가장 비교 횟수가 적은 것으로 나타났고, 거품 정렬이 가장 비교 횟수가 많은 것으로 나타났다. 카드 수가 적을 때와 많을 때 어떤 차이가 있는지를 교수자가 아는 것도 중요하다. 보통 컴퓨터과학 분야에서는 시간 복잡도(Time Complexity)와 공간 복잡도(Space Complexity) 용어를 이용하여 이 개념을 설명하는데, 공간 복잡도는 코딩에 익숙한 학생이 아니라면 이해하기가 쉽지 않고, 시간 복잡도의 경우에는 별도로 용어를 설명하기보다 개념만 알려줄 수 있으면 좋다.

Fig. 1이 나타내는 결과는 컴퓨터과학에서 보편적으로 이야기하는 알고리즘의 성능과 약간의 차이가 있다. 보통 빠른 정렬이 실제적으로 가장 빠르다고 말하지만, 이 그림에서 빠른 정렬은 3번째로 우수한 알고리즘이다. 그리고 병합 정렬이 이론적으로 가장 우수한 알고리즘 중 하나라고 알려져 있지만 이진 삽입 정렬이 더 비교 횟수가 적다. 실제로 이진 삽입 정렬은 데이터를 삽입하는 데에 걸리는 오버헤드 때문에 병합 정렬이나 빠른 정렬보다 느린 알고리즘이고, 시간 복잡도도 더 좋지 않다. 그렇지만 이 놀이 활동에서는 단순히 비교 횟수만을 고려하기 때문에 더 우수한 결과를 나타낸다.

힙 정렬도 매우 빠른 알고리즘이라 알려져 있지만, 이 놀이 활동에서는 순위가 중위권이다. 알고리즘 분야에서 이론적으로 좋은 성능을 내는 알고리즘들이 본 놀이에서는 상대적으로 성능이 좋지 않을 수 있다는 것을 보여준다.

이 결과에 따르면 본 놀이를 할 때 각 알고리즘들을 아래와 같이 5가지 그룹으로 분류할 수 있다.최상위 그룹에는 이진 삽입 정렬, 병합 정렬, 빠른 정렬이 포함되고, 그 다음으로 삽입 정렬과 쉘 정렬이 우수하다. 힙 정렬은 이 놀이에서 그 다음 순위에 위치한다. 선택 정렬, 빗질 정렬, 홀짝 정렬은 하위 그룹에 속하고, 거품 정렬은 가장 성능이 좋지 않다.

3.3 비교 횟수와 시간 복잡도의 상관관계

보통 시간 복잡도가 좋은 알고리즘들은 본 정렬 놀이에서도 비교 횟수가 적다. 그러나 보통 놀이를 할 때의 카드 개수가 많지 않고(10장 이하), 놀이를 할 때는 비교 횟수만을 고려하기 때문에, 시간 복잡도가 좋다고 하여 항상 놀이를 할 때도 좋은 방법이라 말할 수는 없다.

카드를 처음에 어떻게 섞느냐에 따라 알고리즘의 성능은 달라진다. <Table 2>는 8장의 카드를 사용했을 때, 최상의 경우, 시간 복잡도(best-case time complexity)가 어떻게 되는지와 비교해야 하는 횟수가 몇 번인지를 나타낸다. 이 표에서는 시간 복잡도가 좋을수록 실제 비교 횟수도 대체적으로 낮다는 것을 확인할 수 있다.

<Table 3>는 마찬가지로 8장의 카드를 사용했을 때, 최악의 경우 시간 복잡도(worst-case time complexity)가 어떻게 되고, 실제 비교해야 하는 횟수가 몇 번인지를 나타낸다. 이진 삽입 정렬은 시간 복잡도가 좋은 편이 아니지만, 최대 비교 횟수는 병합 정렬과 함께 매우 적음을 알 수 있다. 빠른 정렬의 경우 가장 빠르다고 아는 교수자가 많지만, 최악의 경우 시간 복잡도가 좋지 않고, 비교 횟수도 많다. 따라서 코딩을 할 때는 최악의 상황을 피하기 위해 무작위적(randomized) 빠른 정렬을 사용하는 경우도 많다. 힙 정렬의 경우에는 시간 복잡도가 좋은 편이지만 비교 횟수가 상당히 많음을 볼 수 있다. 거품 정렬도 별도로 최적화를 하지 않는다면 동일한 시간 복잡도를 가지는 다른 알고리즘에 비해 약 2배 많은 비교를 해야함을 볼 수 있다.

홀짝 정렬, https://www.geeksforgeeks.org/odd-even-sort-brick-sort/

거품 정렬(최적화되지 않은 버전), https://en.wikipedia.org/wiki/Bubble_sort

<Table 2> The best-case analysis of the unplugged sorting activity

Alg.	Best-caseTC	Min.#Comparisons(Sequence)
BI	$O(nlogn)$	**13** (3, 6, 2, 7, 4, 5, 1, 8)
ME	$O(nlogn)$	**12** (1, 2, 3, 4, 5, 6, 7, 8)
QU	$O(nlogn)$	**13** (1, 2, 4, 3, 7, 6, 8, 5)
IN	$O(n)$	**7** (1, 2, 3, 4, 5, 6, 7, 8)
SH	$O(n^{7/6})$	**17** (1, 2, 3, 4, 5, 6, 7, 8)
HE	$O(nlogn)$	**21** (8, 1, 4, 2, 7, 5, 6, 3)
SE	$O(n^2)$	**28** (1, 2, 3, 4, 5, 6, 7, 8)
CO	$O(nlogn)$	**24** (1, 2, 3, 4, 5, 6, 7, 8)
OD	$O(n)$	**7** (1, 2, 3, 4, 5, 6, 7, 8)
BU	$O(n)$	**7** (1, 2, 3, 4, 5, 6, 7, 8)

<Table 3> The worst-case analysis of the unplugged sorting activity

Alg.	Worst-caseTC	Max.#Comparisons (Sequence)
BI	$O(n^2)$	**17** (1, 2, 3, 4, 5, 6, 7, 8)
ME	$O(nlogn)$	**17** (1, 3, 2, 7, 4, 6, 5, 8)
QU	$O(n^2)$	**28** (1, 2, 3, 4, 5, 6, 7, 8)
IN	$O(n^2)$	**28** (1, 8, 7, 6, 5, 4, 3, 2)
SH	$O(n^2)$	**30** (7, 3, 5, 1, 8, 4, 6, 2)
HE	$O(nlogn)$	**29** (1, 4, 2, 6, 5, 3, 7, 8)
SE	$O(n^2)$	**28** (1, 2, 3, 4, 5, 6, 7, 8)
CO	$O(n^2)$	**31** (1, 2, 3, 4, 5, 6, 8, 7)
OD	$O(n^2)$	**35** (1, 3, 4, 5, 6, 7, 8, 2)
BU	$O(n^2)$	**56** (2, 3, 4, 5, 6, 7, 8, 1)

<Table 4>는 평균적인 시간 복잡도(average-case time complexity)와 평균적으로 비교해야 하는 횟수를 나타낸다. 시간 복잡도가 우수한 병합 정렬과 빠른 정렬은 예상대로 비교 횟수가 굉장히 적다. 그러나 이진 삽입 정렬은 시간 복잡도가 좋지 않지만 오히려 병합 정렬과 빠른 정렬보다 더 비교 횟수가 적은 특징을 보인다. 쉘 정렬도 삽입 정렬에 비해 시간 복잡도가 우수하지만, 8장의 카드를 사용한 경우 상대적으로 더 많은 비교 횟수가 필요하다. 힙 정렬은 앞에서 보았던 실험 결과와 마찬가지로 이론적인 성능에 비해 비교 횟수가 많은 편이다.

<Table 4> The average-case analysis of the unplugged sorting activity

Alg.	Average-caseTC	Average#Comparisons
BI	$O(n^2)$	15.59
ME	$O(nlogn)$	15.73
QU	$O(nlogn)$	16.92
IN	$O(n^2)$	19.28
SH	$O(n^{5/4})$	21.82
HE	$O(nlogn)$	25.81
SE	$O(n^2)$	28
CO	$O(n^2)$	30.73
OD	$O(n^2)$	30.87
BU	$O(n^2)$	41.43

3.4 알고리즘 간 성능 비교

3.3절에서 살펴본 것처럼, 최상의 경우, 최악의 경우, 평균적인 경우에 비교 횟수가 크게 달라졌다. 따라서 본 놀이를 할 때 어떤 특정 알고리즘이 다른 알고리즘보다 무조건 더 좋다고 말하기가 쉽지 않다.

그러나 본 놀이를 할 때 더 이길 확률이 높은 알고리즘은 분명 존재한다. 어떤 학생 A와 B가 서로 경쟁하여 각각 정렬 알고리즘을 고안했다고 하자. 선생님이 카드를 섞은 후, 이 상태에서 A가 만든 알고리즘을 실행했을 때와 B가 만든 알고리즘을 실행해 보았을 때 비교 횟수가 적은 학생이 이긴다고 가정해 보자. 만약 A가 만든 알고리즘이 이진 삽입 정렬이고, B가 만든 알고리즘이 삽입 정렬이라면, A가 이길 확률은 질 확률보다 더 높을 것이라 예상할 수 있다.

각각의 알고리즘이 다른 알고리즘을 이길 확률이 어떻게 되는지를 Table 5, 6, 7에서 나타내었다. Table 5는 6장의 카드를 사용한 경우, Table 6는 8장의 카드를 사용한 경우, Table 7은 10장의 카드를 사용한 경우를 나타낸다. 각 표에서 (i번째 열, j번째 행)의 값은 i번째 알고리즘이 j번째 알고리즘을 이길 확률을 나타낸다. 예를 들어 위 사례에서 A 학생이 B 학생을 이길 확률은 8장의 카드를 사용했을 때 84.15%가 된다.

이 3개의 표에서 주목할 부분을 굵은색 테두리로 표기하였다. 먼저 이진 삽입 정렬, 병합 정렬, 빠른 정렬의

<Table 5> The relative performance of the selected algorithms when using six cards

Alg.	BI	ME	QU	IN	SH	HE	SE	CO	OD	BU
BI	-	55.3	54.4	71.66	80.91	100	100	100	97.46	95.56
ME	44.7	-	53.59	72.23	88.51	100	100	100	99.86	99.86
QU	45.6	46.41	-	66.46	67.89	98.43	100	98.89	96.9	96.64
IN	28.34	27.77	33.54	-	55.37	96.5	100	99.44	100	100
SH	19.09	11.49	32.11	44.63	-	100	100	99.86	98.73	99.15
HE	0	0	1.57	3.5	0	-	42.86	92.31	78.21	88.71
SE	0	0	0	0	0	57.14	-	90	92.98	94.58
CO	0	0	1.11	0.56	0.14	7.69	10	-	62.78	79.31
OD	2.54	0.14	3.1	0	1.27	21.79	7.02	37.22	-	91.15
BU	4.44	0.14	3.36	0	0.85	11.29	5.42	20.69	8.85	-

<Table 6> The relative performance of the selected algorithms when using eight cards

Alg.	BI	ME	QU	IN	SH	HE	SE	CO	OD	BU
BI	-	60.23	62.23	84.15	100	100	100	100	99.68	99.68
ME	39.77	-	59.29	86.2	100	100	100	100	99.92	99.92
QU	37.77	40.71	-	73.88	90.98	99.04	100	99.91	98.94	99.34
IN	15.85	13.8	26.12	-	76.67	95.49	100	99.76	100	100
SH	0	0	9.02	23.33	-	92.08	99.68	99.67	97.11	98.96
HE	0	0	0.96	4.51	7.92	-	98.83	96.29	90.76	96.29
SE	0	0	0	0	0.32	1.17	-	96.11	85.93	95.91
CO	0	0	0.09	0.24	0.33	3.71	3.89	-	51.14	85.13
OD	0.32	0.08	1.06	0	2.89	9.24	14.07	48.86	-	95.64
BU	0.32	0.08	0.66	0	1.04	3.71	4.09	14.87	4.36	-

<Table 7> The relative performance of the selected algorithms when using ten cards

Alg.	BI	ME	QU	IN	SH	HE	SE	CO	OD	BU
BI	-	64.49	68.31	91.15	99.92	100	100	100	99.99	99.99
ME	35.51	-	61.81	94.51	99.97	100	100	100	100	100
QU	31.69	38.19	-	80.93	87.07	99.58	100	99.88	99.6	99.77
IN	8.85	5.49	19.07	-	49.71	94.24	100	99.21	100	100
SH	0.08	0.03	12.93	50.29	-	99.93	100	99.93	99.38	99.86
HE	0	0	0.42	5.76	0.07	-	100	93.05	87.57	97.41
SE	0	0	0	0	0	0	-	13.61	76.49	97.07
CO	0	0	0.12	0.79	0.07	6.95	86.39	-	80.96	97.35
OD	0.01	0	0.4	0	0.62	12.43	23.51	19.04	-	97.56
BU	0.01	0	0.23	0	0.14	2.59	2.93	2.65	2.44	-

상대적인 승률을 보면, 카드의 수가 많아짐에 따라 상대 승률의 격차가 더 커지는 것을 확인할 수 있다. 보통 병합 정렬이나 빠른 정렬이 가장 뛰어난 알고리즘이라고 말하는 것과는 크게 다른 결과라 볼 수 있다.

두 번째로, 삽입 정렬과 쉘 정렬을 비교해 보면 특이한 결과가 나타남을 볼 수 있다. 삽입 정렬은 6장의 카드를 사용했을 때는 두 알고리즘의 승률이 크게 차이가 나지 않는데(삽입 정렬의 승률 55.37%), 8장의 카드를 사용하면 삽입 정렬의 승률이 76.67%로 올라간다. 그리고 10장의 카드를 사용하면 반대로 승률이 49.71%로 떨어지는 것을 확인할 수 있다.

세 번째로, 선택 정렬, 빗질 정렬, 홀짝 정렬을 비교한 결과도 특이하다. 선택 정렬의 경우 카드의 수가 6장일 때보다 8장일 때 더 승률이 높아지지만, 10장이 되면 승

률이 크게 떨어진다. 반대로 빗질 정렬은 카드 수가 8장일 때 선택 정렬과 대결 시 승률이 3.89% 밖에 되지 않지만, 카드 수가 10장이 되면 승률은 86.39%로 올라간다.

3.5 알고리즘의 난이도 분석

학생들에게 본 논문에 나타난 10가지 알고리즘을 모두 알려줄 수 있으면 좋지만, 컴퓨터과학에 대한 흥미를 오히려 떨어뜨릴 수 있으므로 주의할 필요가 있다. 10가지 알고리즘 중 몇 가지만 가르쳐야 한다면 어떤 것을 가르쳐야 하는지를 알기 위해, 각 알고리즘을 프로그램으로 구현했을 때 얼마나 복잡한 프로그램이 되는지를 분석해 보았다. Table 8은 각 알고리즘을 Python으로 구현했을 때의 프로그램 라인 수(Lines), 순환 복잡도(cyclomatic complexity), 토큰 수(Tokens), 재귀호출 사용 여부(Recur.)를 나타낸 것이다.

Table 8에 따르면 프로그램의 복잡도를 나타낸다고 볼 수 있는 순환 복잡도가 5 이하인 알고리즘은 빠른 정렬(4), 삽입 정렬(4), 쉘 정렬(5), 선택 정렬(4), 거품 정렬(5)이다. 여기에서 쉘 정렬은 순환 복잡도가 낮은 편이지만, 예외 조건이 있어 규칙을 완전히 이해하기에는 쉽지 않다. 반면 거품 정렬은 순환 복잡도가 5이지만, 구현하는 방식에 따라 순환 복잡도가 4로 내려갈 수 있고, 단순 작업을 반복하여 수행하면 되기 때문에 가르치기에 매우 용이하다. 빠른 정렬의 경우 보통 재귀 호출을 이용하여 코딩하는 경우가 많기 때문에 프로그램 코드를 작성할 때는 매우 난이도가 높지만, 단순히 놀이를 할 때에는 그 규칙을 이해하기 쉬운 편이다. 실제로 똑똑한 학생들이 이 방법을 종종 생각해 내곤 한다.

따라서 거품 정렬, 선택 정렬, 삽입 정렬, 빠른 정렬을 놀이 시 우선적으로 알려주는 것이 좋고, 코딩 과정과 연계할 시에는 거품 정렬, 선택 정렬, 삽입 정렬을 활용하는 방법이 좋을 것이라 판단된다.

4. 결론

본 논문에서는 언플러그드 활동 중 하나인 정렬 놀이를 분석하였다. 이 놀이는 컴퓨터과학 분야의 정렬 알고리즘들을 활용할 수 있지만, 컴퓨터과학에서 우수한 성능을 나타내는 알고리즘이라 할지라도 본 놀이에서 반드시 좋은 성능을 보장하지는 못한다는 것을 보였다. 또, 본 논문에서는 교수자가 알아야 할 알고리즘이 어떤 것인지, 학생들에게 가르쳐야 할 알고리즘이 어떤 것인지를 판단할 때 도움이 되는 분석 결과들을 제시하였다.

<Table 8> The statistics of each sorting program

Alg.	Lines	CC	Tokens	Recur.
BI	24	8	165	Yes
ME	25	7	150	Yes
QU	**12**	**4**	**117**	**Yes**
IN	**8**	**4**	**63**	**No**
SH	12	5	75	No
HE	19	8	151	Yes
SE	**7**	**4**	**67**	**No**
CO	13	6	96	No
OD	14	6	131	No
BU	**10**	**5**	**72**	**No**

본 논문은 정렬 놀이를 분석했지만, 학생들에게 본 놀이를 어떤 방식으로 가르치는 것이 좋을지에 대해 깊게 탐구하지 않았다. 향후 연구를 통해 정렬 놀이를 효과적으로 가르치는 방법, 그리고 이를 엔트리나 스크래치, 파이썬과 같은 프로그래밍 활동과 연계하는 방안에 대해 고민할 필요가 있다.

참고문헌

[1] CSUnplugged, https://csunplugged.org

[2] Go, H. and Kim, C. (2016). Development of Finite State Automata Learning Materials for Elementary School Students. *Journal of the Korean Association of Information Education, Vol 20*(4), pp. 401-408.

[3] Han, B. (2017). The Elementary Students' Understanding of Computer Science Through the Computer Science Show Program. *Journal of the Korean Association of Information Education, Vol 21*(2), pp. 209-217.

[4] Han, B. (2013). The Research of Unplugged Computing Method for Computational Thinking in Elementary *Informatics Education. Vol 17*(2), pp. 147-156.

[5] Han, H. and Han, S. (2008). A Case Study on Information Education for Pre-Service Teacher using Unplugged Computing. *Journal of the Korean Association of Information Education, Vol 13*(1), pp. 23-30.

[6] Han, B., Gu, J. and Song, T. (2016). An Activity-based Instructional Design for Search Algorithm Expression of Elementary Students. *Journal of the Korean Association of Information Education. Vol 20*(2), pp. 161-170.

[7] Han, S. and Shin, S. (2011). Development of Edutainment Program using Computer Science Unplugged. *Journal of the Korean Association of Information Education. Vol 15*(2). pp. 201-208.

[8] Jang, J. and Kim, C. (2016). Development of Sorting Algorithm Contents for Improving the Problem-solving Ability in Elementary Student. *Journal of the Korean Association of Information Education. Vol 20*(2), pp. 151-160.

[9] Jeong, I. (2018). Software Battle for Algorithm Education – Focused on Sorting Algorithm. *Journal of the Korean Association of Information Education. Vol 22*(2), pp. 223-230.

[10] Kim, J. (2018). A Study on Systematic Review of Unplugged Activity. *Journal of the Korean Association of Information Education. Vol 22*(1), pp. 103-111.

[11] Lee, Y. and Lee, Y. (2009). A Comparison of Teaching and Learning Method of Sorting Algorithm based on the Playing Activity and Animation. *Journal of the Korean Association of Information Education. Vol 13*(2), pp. 225-236.

[12] Lim, M. and Han, B. (2006). A Study on Learnability of Search and Sort Algorithm in Elementary School Computer Education. *Journal of the Korean Association of Information Education. Vol 10*(3), pp. 289-298.

[13] Ma, D. (2016). A Study of Data Representation Education for Elementary Students. *Journal of the Korean Association of Information Education. Vol 20*(1), pp. 13-20.

[14] Park, Y. (2018). An Unplugged Activity to Understand the PageRank Algorithm. *Korean Association of Information Education. Vol 22*(4), pp. 409-417.

[15] Sorting Algorithm, https://en.wikipedia.org/wiki/Sorting_algorithm

[16] Yang, C. (2016). Computer Science Unplugged Activities of Graph Theory for Primary School Students. *Journal of the Korean Association of Information Education, Vol 20*(1), pp. 93-100.

저자소개

박 영 기

2008 KAIST 전자전산학과 전산학전공(공학사)

2010 서울대학교 대학원 컴퓨터공학과(공학석사)

2015 서울대학교 대학원 컴퓨터공학과(공학박사)

2015~2016 삼성전자 종합기술원 전문연구원

2016~현재 춘천교육대학교 컴퓨터교육과 조교수

관심분야 : 초등컴퓨터과학교육

E-Mail : ypark@cnue.ac.kr