**시스템 프로그래밍 개인과제**

201621025

신주연

**목차**

1. 알고리즘에 대한 개요
2. 실행 결과 스크린 샷 및 결과에 대한 설명
3. 구현사항/ 미 구현사항/ 버그/ 개선점 및 이유
4. 고찰
5. 알고리즘에 대한 개요
6. Bubble sort

버블 정렬은 서로 인접한 두 원소를 검사하여 정렬하는 알고리즘으로, 인접한 2개의 원소를 비교하여 크기가 순서대로 되어 있지 않으면 서로 교환한다. 1회 수행하고 나면 가장 큰 값을 가진 원소가 맨 뒤로 이동하게 되고 2회에서는 맨 끝에 있는 자료는 정렬에서 제외되고, 2회를 수행하고 나면 끝에서 두 번째 자료까지는 정렬에서 제외된다. 이렇게 정렬을 1회전 수행할 때마다 정렬에서 제외되는 데이터가 하나씩 늘어난다. 구현이 간단하나 시간 복잡도 측면에서 n^2으로 비효율적이다.

1. Insertion sort

삽입 정렬은 앞의 원소보단 크고, 뒤에 원소보단 작은 위치에 삽입하는 방법이다. 즉, 앞쪽에 있는 원소들은 이미 정렬이 됐다고 가정한다. 두 번째 원소부터 시작하여 그 앞에 있는 원소들을 정렬하는 방식이다. 시간 복잡도는 안정한 정렬이라는 특징이 있고 버블 정렬과 마찬가지로 n^2이지만 정렬이 되어있는 경우 이동이 없으므로 특정 경우 버블 정렬보다 빠르다는 장점이 있다. 그러나 원소의 수가 많고 크기가 클 경우에는 적합하지 않다.

1. Selection sort

선택 정렬에 대하여 고찰해 본 결과 두 가지의 방식이 있는 것으로 보인다. 먼저 첫째는 첫 번째 자료를 두 번째 자료부터 마지막 자료까지 차례대로 비교하여 가장 작은 값을 찾아 첫 번째에 놓고, 두 번째 자료를 세 번째 자료부터 마지막 자료까지 차례대로 비교하여 그 중 가장 작은 값을 찾아 두 번째 위치에 놓는 과정을 반복하며 정렬을 수행하는 방법이다. 이어서 둘째는 최소가 되는 원소를 찾은 후 이를 정렬된 부분에 추가해 나가는 방식이다. 본 과제에서 예시 코드는 두 번째 경우에 해당한다.

1. Heap sort

힙은 완전 이진 트리의 일종으로 우선순위 큐를 위하여 만들어진 자료구조이다. 힙 정렬은 최대 힙 트리나 최소 힙 트리를 구성해 정렬을 하는 방법이다. 내림차순 정렬을 위해서는 최대 힙을 구성하고 오름차순 정렬을 위해서는 최소 힙을 구성하면 된다. 힙을 만든 후 배열을 순차적으로 저장하고 삭제되는 요소들은 값의 순서에 따라 정렬되는 방식이다. 시간 복잡도가 nlogn으로 굉장히 좋은 편이고 필요한 가장 큰 값만 정렬하는 경우 사용하기 좋다.

1. Merge sort

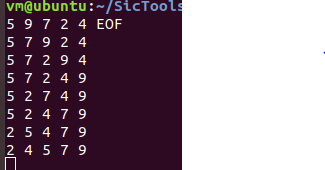
합병 정렬은 분할 정복 방법 중 하나이며 리스트를 절반으로 잘라 나눠가며 재귀적으로 합병 정렬을 이용하여 정렬한다. 정렬되지 않은 리스트를 절반으로 잘라 비슷한 크기의 두 부분 리스트로 나눈 후 각 부분 리스트를 재귀적으로 합병 정렬을 이용하여 정렬한다. 이후 다시 합병하는 방식을 이용한다. 시간 복잡도 측면에서 힙 정렬과 동일한 효율성을 보이지만, 원소들의 크기가 큰 경우 이동 횟수가 많아지게 된다.

1. Quick sort

할 정복 알고리즘의 하나로, 평균적으로 매우 빠른 수행 속도를 자랑하는 정렬 방법이다. 리스트 안에 있는 한 요소를 선택한다. 이렇게 고른 원소를 피벗(pivot) 이라고 한다. 피벗을 기준으로 피벗보다 작은 요소들은 모두 피벗의 왼쪽으로 옮겨지고 피벗보다 큰 요소들은 모두 피벗의 오른쪽으로 옮겨진다. 피벗을 제외한 왼쪽 리스트와 오른쪽 리스트를 다시 정렬한다. 이후 부분 리스트들이 더 이상 분할이 불가능할 때까지 반복한다. nlogn 알고리즘 중에서도 속도가 가장 빠르며 추가 메모리 공간을 필요로 하지 않는다는 장점이 있다.

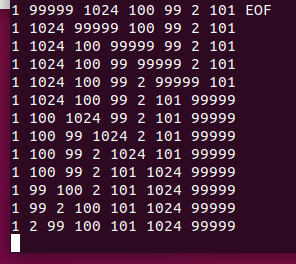
1. 실행 결과 스크린 샷 및 결과에 대한 설명
2. Bubble sort

과제 예시



버블 정렬의 경우 과제 예시에선 swap이 일어날 경우에만 출력이 되도록 하였다. 과제 예시 코드와 일치한다.

특이한 조건



입력 조건이 정해져 있었기에 입력 조건에 해당하는 부분 중, 자리 수를 한 자리 씩 출력하면서 이루어지는 여러 10으로의 나누기에서 취약한 숫자들을 넣어 보았다. 또한 1024와 같은 0이 아닌 숫자 사이에 0이 들어있는 경우 자리 수 분할을 위한 또한 10으로의 나누기에 취약하므로 넣어보았다. 버블 정렬이 완전히 이루어졌다.

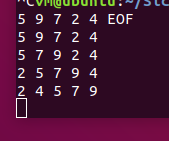
이미 정렬된 형태



이미 정렬된 형태의 경우 버블 정렬에서 swap이 일어나는 경우에만 출력이 이루어 지므로 원래의 알고리즘으로 구현 시 아무 것도 출력되지 않았다. 그러나 예외적으로 처리하여 stop당시 print를 한 번도 실행하지 않았다면 print로 pc를 보내어 출력하고 다른 경우들의 프린트 완료 후 jump되는 pc와는 달리 stop에서 온 경우는 프로그램이 종료되도록 구현하였다.

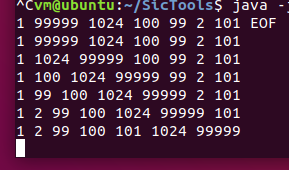
1. Insertion sort

과제 예시



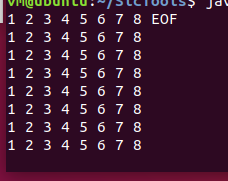
삽입 정렬의 경우 첫 번째 59724가 출력됨을 보아 두 번째 인덱스부터 순차적으로 보면서 버블 정렬과는 다르게 이동이 일어나지 않아도 출력됨을 알 수 있다. 과제 예시 코드와 일치한다.

특이한 조건



마찬가지로 취약한 숫자들의 경우에도 삽입 정렬이 잘 이루어졌다.

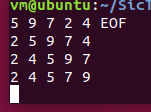
이미 정렬된 형태



이미 정렬된 경우 한 번만 출력하는 것으로 예외 처리를 진행할 것인지에 대해 고민을 많이 하였지만, 과제 예시에서 59724의 경우 변함이 없는데 출력되었음을 파악하여 모든 경우를 출력하였다.

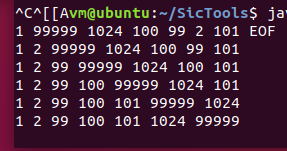
1. Selection sort

과제 예시



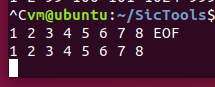
기존에 잘 알려진 선택 정렬 방식이 아닌 최소값을 찾아 순서대로 앞에 정렬해 나가는 선택 정렬이다. 과제 예시 코드와 일치한다.

특이한 조건



마찬가지로 취약한 숫자들의 경우에도 선택 정렬이 잘 이루어졌다.

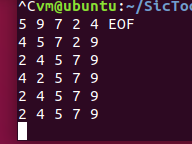
이미 정렬된 형태



버블 정렬과 같이 이동이 이루어진 경우에만 출력되는 방식이므로 이동이 없는 경우버블 정렬과같은 방식으로 따로 예외적으로 처리하였다.

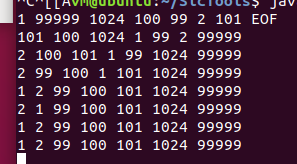
1. Heap sort

과제 예시



과제 예시에 대한 출력을 진행해 보았다. 정렬에 성공하였다.

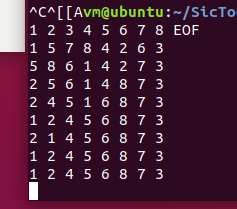
특이한 조건



이미 정렬된 형태

마찬가지로 취약한 숫자들의 경우에도 힙 정렬이 잘 이루어졌다.

이미 정렬된 형태



버그가 생겨 처리하지 못했다. 뒤에서 다루도록 하겠다.

1. 구현 사항/미 구현사항/버그/개선점 및 이유

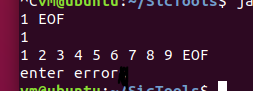
개인과제로 주어진 구현은 삽입 정렬이었으나, 구현 완료 후 버블 정렬 및 선택 정렬을 구현하였고, 이어서 힙 정렬도 구현해 보았다. 모든 알고리즘을 구현할 때 경우 익숙하지 않은 어셈블리어로 바로 구현을 시도하지 않고 C언어로 알고리즘을 구현해 본 후 이를 어셈블리어로 바꾸는 방식으로 진행하였다.

코드 구현은 모든 정렬에서 네 부분으로 나누어져 있다. 첫째, 배열에 입력 받은 숫자들을 넣는 부분. 둘째, 정렬이 이루어지는 부분. 셋째, 출력이 이루어지는 부분. 넷째, 예외 처리 및 종료 부분. 다섯째, 변수 선언 부분. 이 있다. 공통된 부분을 먼저 언급한 후 정렬 별로 살펴보도록 하겠다.

배열에 입력 받은 숫자들을 넣는 부분은 모든 정렬에서 공통적으로 작용하므로 동일한 코드를 복사하여 사용하였다. 스페이스바, 숫자, EOF마다 라벨을 만들어 처리하였다. 숫자의 경우 한 자리 숫자가 아닌 경우 한 자리씩 저장한 후 이미 저장된 값이 있다면 10을 곱하여 더한 후 다시 저장하는 방법을 사용하였다. for문과 if문과 같은 pc의 이동이 주가 되는 코드들은 모두 1-pass에서 읽히는 라벨로 처리하였다. for문의 첫 번째 순환의 경우 인덱스 i의 초기화가 이루어지므로 FORIS(for문의 인덱스는 i인 S start)앞에 초기화 코드를 작성하고 FORIS를 시작하였다. 두 번째 순환부터는 S(start)가 아닌 FORIM(middle)을 사용하여 i++을 구현하고 ++이후에도 for문 순환 조건에 어긋나지 않는지 검사하여 만족한다면 다시 FORIS로 pc를 이동시키는 방법을 사용하여 모든 반복문들을 구현하였다.

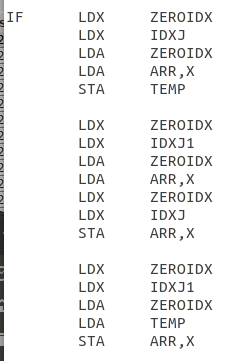
출력이 이루어지는 부분 또한 모든 정렬에서 공통적으로 작용하므로 동일한 코드를 복사하여 사용하였다. 자리 수 마다 라벨을 만들어야 함을 깨달은 후 순차적으로 9999, 999, 99, 9와 크기를 비교하여 자리 수를 알아내고 한 자리씩 밑의 라벨로 내려가며 모든 자리를 확인하였다. 10진수로 저장된 수를 0을 더한 후 출력하고 다시 0을 빼 주어 저장하였으며, 각각의 자리수를 따로 처리하여 출력하였다. 예로 388의 경우 99보다 크므로 3자리 수 라벨에 들어가게 되고, 이후 n자리수의 가장 첫 글자를 알아내기 위한 n자리수 10의 배수인 100으로 나눠주어 3을 얻은 후 388-300을 해준 값을 한 자리수 적은 라벨 즉 두 자리 수 라벨로 보낸다. 이후 두 자리 수 라벨에서도 이를 반복하여 처리하였다. 마지막 한 자리 수의 경우의 마침으로는 출력 이후 시점으로 pc를 돌리는 j code를 작성하였다.

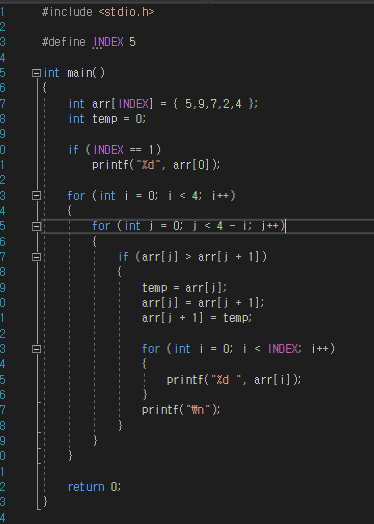
예외 처리 및 종료 부분 또한 모든 정렬에 대하여 과제에서 주어진 대로 입력에 대한 적절한 예외처리를 동일하게 진행하였다. (1개 및 9개의 입력, EOF가 아닌 종료문구, 99999이상의 값) 예외적인 입력이 발생한 경우 enter error라는 문구를 출력하고 stop시켜 종료시켰다.





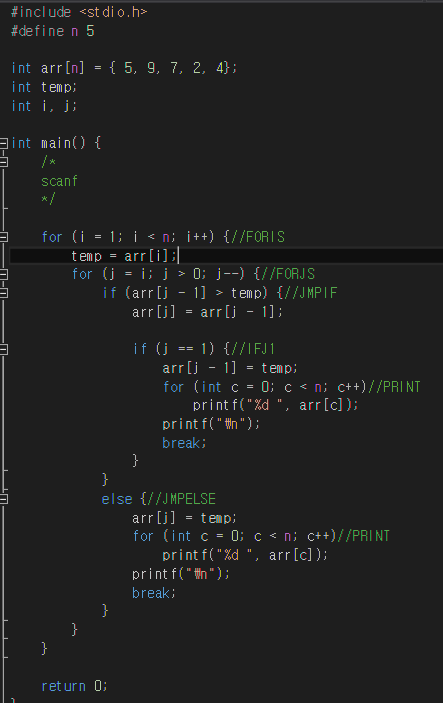
1. 버블 정렬

가독성을 높이기 위해 과제에 사용한 c코드를 같이 첨부하였다. 아래와 같은 c코드로 작성 후 구현을 시작하였다. 입력 조건에 대한 여러가지 경우의 수를 테스트해 보던 중 1개의 입력만 있을 경우 알고리즘이 제대로 동작하지 않음을 알게 되었고, 예외적으로 index가 1인 경우를 처리하였다. 또한 정렬이 이미 완성되어 있는 경우 역시 예외적으로 처리하였다. 상당히 많은 부분에서 swap이 이루어 졌으며 for문의 조건이 상당히 까다로웠다. 그러나 이것 외에는 어렵거나 복잡한 부분이 없이 직관적이어서 가장 짧은 시간안에 구현을 완료할 수 있었다.



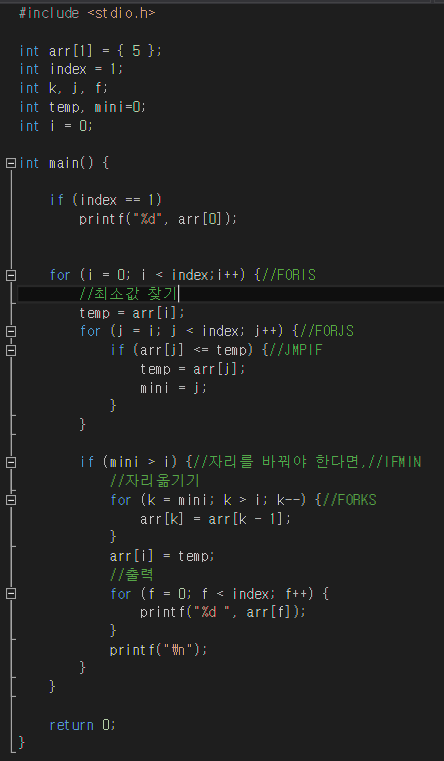
위의 오른 쪽에 첨부한 코드와 같은 swap이 상당히 많이 이루어 진 것이 특징이다.

1. 삽입 정렬



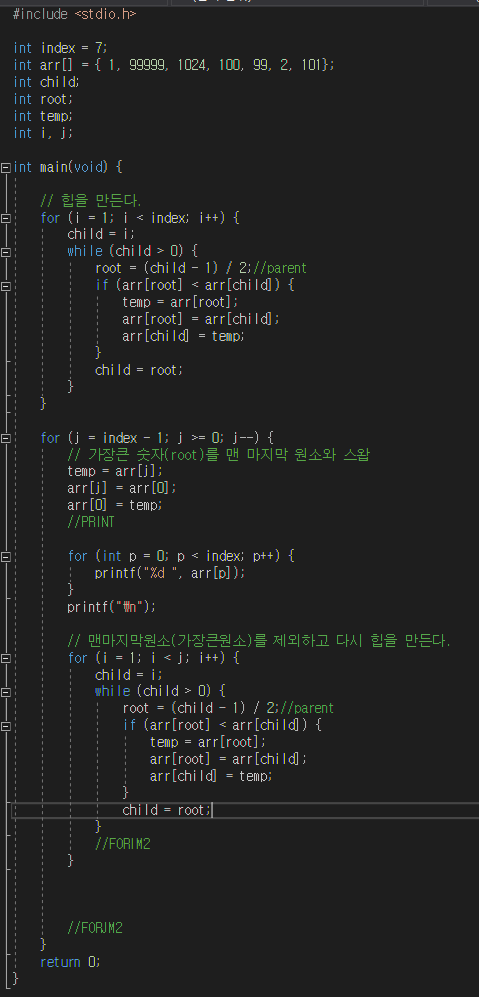
위와 같은 알고리즘으로 진행하였다. 각각의 라벨은 위의 주석과 같이 작성하였으며 반복문과 조건문이 상당히 얽혀 있어서 pc의 이동 및 라벨의 사용이 굉장히 많았다. 반복문의 중간에 조건문이 들어가기도 하고 if 와 else가 temp의 크기에 따라 이루어 지기도 해서 for문의 중간 단계와 시작 단계 및 초기화 단계를 나누어서 라벨을 지정하여 사용하였다. 그런 만큼 깊게 이해한 후 실행하지 않으면 발생하는 오류를 잡는 것이 어려웠기에 충분한 이해를 토대로 코드를 구현 및 성공하였다.

1. 선택 정렬



기존에 알고 있던 swap방식을 이용한 선택 정렬 방법과 다르게 자리를 바꿔야 하는 경우 빈 자리를 한 칸 씩 이동하여 채워주는 선택 정렬 방식이었다. 최소 값을 찾는 for문을 통해 최소 인덱스와 값을 알아낸 후 이를 temp에 저장하여 빈 자리를 한 칸 씩 당기고 적절한 자리에 temp를 이동시키는 방식을 사용하였다. 자리를 바꾼 경우에만 출력이 이루어 지는 것이 아님을 파악하고 단계별로 진행할 수 있었다. 모든 반복문의 조건과 비교하여 만족하지 않는 즉 모든 반복문을 탈출하는 시점을 STOP시점으로 정하였다.

1. 힙 정렬



힙 정렬의 개념을 이해하는 것부터 오랜 시간이 소요되었다. 배열과 트리를 이용하여 정렬 할 때 가장 조심해야 할 개념인 트리의 인덱스를 맞추는 것에도 상당 시간을 소요하였다. 자식 노드로부터 부모의 인덱스를 알아낼 때 parent = (child – 1)/2가 사용되는데, 이를 어셈블리어로 구현할 땐 (child / 3 – 1) / 2 \* 3이 이루어 진다고 생각했다. 그리고 이를 간단히 하기 위한 약분을 진행하였고 (idx – 3) / 2로 부모 노드를 구현 했었 는데 이에 큰 문제가 있음을 디버깅 중 발견하였다. 위의 경우 부모 노드의 값이 4가 발생하게 된다. 이는 0부터 3씩 증가하는 배열의 인덱스에 존재하지 않는 값이었으며 늦게 발견하여 아쉬웠다.

모든 알고리즘에 대하여 기본적인 정렬 및 예외 처리를 모두 진행하였으며, 여러 가지 경우의 수를 생각해 보며 끊임없이 수정하였다. 다만 힙 정렬에서 대부분의 정렬은 성공적이지만 이미 정렬이 완료 되어있는 배열의 경우 n만큼의 시간 복잡도가 발생하는 특징을 살려 구현하지 못한 것이 아쉽다.

1. 고찰

배열과 트리를 연결하는 과정에서 인덱스의 사용이 미숙했던 힙 정렬을 제외한 다른 정렬들의 구현은 상당히 만족스럽다. c코드로 작성 시 굉장히 간결한 정렬들인데 어셈블리어로 작성 해보니 상당히 길어진 것이 인상깊었고, 무엇보다 출력이 이렇게 복잡하게 진행 되는 것을 몰랐던 부분이 가장 인상적이다. SicTool의 디버깅이 순조롭기도 하고 코드 한 줄 한 줄을 풀어서 작성하다 보니 오류를 찾는 데에 소요한 시간은 그리 오래 걸리진 않았던 것 또한 인상적이다.

출력에 대한 부분을 어떻게 구현할 것인가에 대한 고민을 하는 데에 상당히 많은 시간을 투자하였다. 처음엔 1 10 100 1000 10000 으로 경우를 나눴었는데 이 다섯 가지의 숫자들과 배열 안의 수를 비교한 후 출력할 때 100이 출력이 안됨을 보고 충격 받았던 것이 기억에 남는다. 100보다 큰 경우와 그렇지 않은 경우에 대한 조건만 걸어 줬었고 이에 당황했었다. 이후 100으로 진행하면 3자리 수들 중 100이 빠지게 됨을 인지하고 모든 비교 조건을 9 99 999 9999 로 바꾼 것이 가장 기억에 남는 큰 수정이었다.

기본 정렬 세 가지를 구현 할 때 본 과제인 삽입 정렬을 가장 먼저 구현한 후 선택 정렬버블 정렬을 순서대로 구현하였는데, 이전의 두 정렬을 구현하면서 실력이 향상되기도 했겠지만, 확실히 버블 정렬이 간단함을 몸소 느꼈다. 그러나 그런 만큼 출력 결과의 길이가 길고 복잡했던 것 또한 인상적이다. 이에 그치지 않고 이번 과제에서 완벽하게 구현해 내지 못한 힙 정렬에 대한 부분을 조금 더 공부해 보도록 하겠다.