**정렬 알고리즘 결과 보고서**

정보시스템공학과 20181028 백선혜

본 프로그램은 6개의 sorting algorithm을 비교 분석하기 위한 프로그램으로, 아래에서는 측정한 실행시간을 바탕으로 그래프로 비교 및 분석해본다. 분석할 알고리즘은 O(n^2) 알고리즘 3개(Exchange Sort, Insertion Sort, Selection Sort)와 O(nlogn) 알고리즘 3개(Merge Sort, Quick Sort, Heap Sort)이다.

모든 참고는 ‘Richard E.Neapolitan, [알고리즘 기초 5판], 도경구 역, 홍릉과학출판소, 2017’했으며, 이하 교과서라고 칭하겠다.

## **Ⅰ. 각 알고리즘의 간단한 소개**

교환 정렬(Exchange Sort)은 처음 데이터와 다음 데이터를 비교하는 연산을 마지막 바로 이전의 데이터와 마지막 데이터를 비교할 때까지 반복하여 data[0:n-1]을 정렬한다. 교과서 7 페이지의 알고리즘 1.3 교환 정렬을 참고했다.

삽입 정렬(Insertion Sort)은 리스트로부터 요소를 하나씩 취한 다음 마치 돈을 지갑에 넣는 방식과 비슷하게 이들을 올바른 위치에, 새로 정렬된 리스트에 삽입함으로써 data[0:n-1]을 정렬한다. 교과서 272 페이지의 알고리즘 7.1 삽입정렬을 참고했다.

선택 정렬(Selection Sort)은 최소값을 찾고 값을 최초 위치와 바꿔친 다음 리스트의 나머지 부분에 대해 이 과정을 반복하여 data[0:n-1]을 정렬한다. 교과서 275 페이지의 알고리즘 7.2 선택정렬을 참고했다.

합병 정렬(Merge Sort)은 정렬되지 않은 리스트를 절반으로 잘라 비슷한 크기의 두 부분 리스트로 나눈 후, 각 부분 리스트를 재귀적으로 합병 정렬을 이용해 정렬하고, 이 두 부분 리스트를 다시 하나의 정렬된 리스트로 합병하여 data[0:n-1]을 정렬한다. 교과서 61 페이지의 알고리즘 2.5 합병 정렬을 참고했다.

빠른 정렬(Quick Sort)은 리스트 가운데서 하나의 원소를 골라 이를 피벗이라 칭하고, 피벗을 기준으로 분할된 두 개의 작은 리스트에 대해 재귀적으로 반복하여 data[0:n-1]을 정렬한다. 교과서 62 페이지의 알고리즘 2.5 빠른정렬을 참고했다.

힙 정렬(Heap Sort)은 n개의 노드에 대한 완전 이진 트리를 구성하여 최대 힙을 구성하면, 가장 큰 수를 가장 작은 수와 교환하는 과정을 반복하여 data[1:n]을 정렬한다. 교과서 293 페이지의 알고리즘 7.5 힙정렬을 참고했다.

## **Ⅱ. 컴퓨터 사양**

* COMPUTER : LG gram 15Z980-HA76K
* CPU : Intel Core i7-8550U @ 1.80GHz
* RAM : 16GB
* OS : Mircrosoft Windows 10 Home
* IDE : Mircrosoft Visual Studio 2018

## **Ⅲ. 측정된 시간에 대한 table 작성**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **n** | **1,000** | **5,000** | **10,000** | **50,000** | **100,000** | **500,000** | **1,000,000** |
| Exchange Sort | 0.0093 | 0.1841 | 0.6357 | 18.4695 | 74.1329 | 1556.09 | 3035.48 |
| Insertion Sort | 0.0044 | 0.0491 | 0.1568 | 4.4919 | 17.3803 | 229.745 | 653.81 |
| Selection Sort | 0.0064 | 0.0765 | 0.2302 | 6.8156 | 26.891 | 371.899 | 1141.832 |
| Merge Sort | 0.0036 | 0.0074 | 0.0109 | 0.0413 | 0.0898 | 0.4018 | 0.7768 |
| Quick Sort | 0.0026 | 0.0051 | 0.0085 | 0.0278 | 0.0616 | 0.2852 | 0.5805 |
| Heap Sort | 0.0023 | 0.004 | 0.0064 | 0.0194 | 0.0406 | 0.1838 | 0.4091 |

## **Ⅳ. 측정된 시간에 대하여 graph 작성**

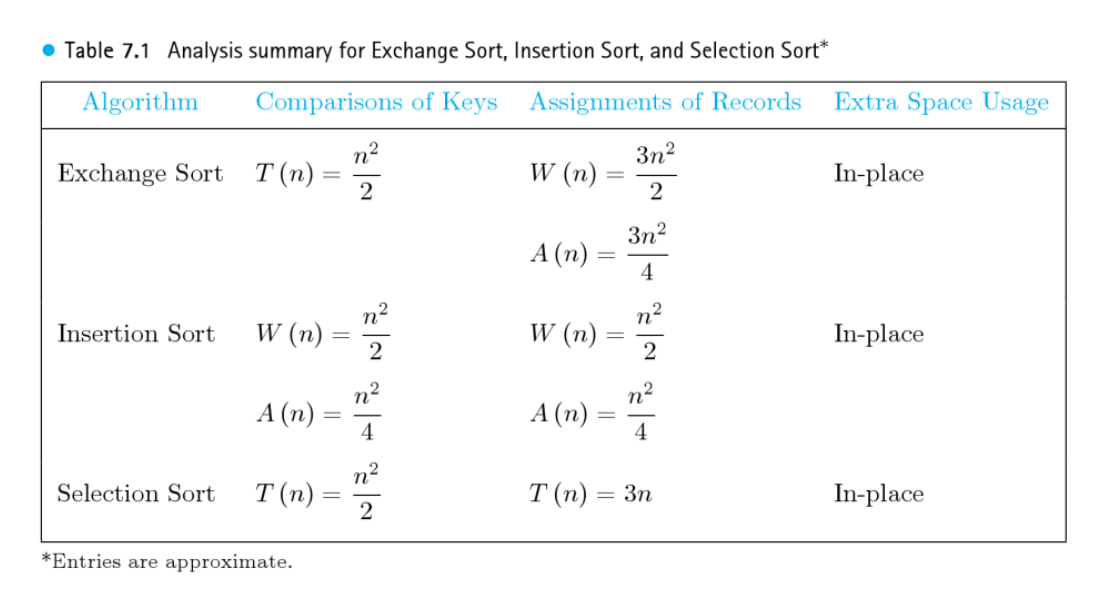
성능을 비교하는 그래프와 O(n^2) 알고리즘 3개와 O(nlogn) 알고리즘 3개의 성능을 각각 비교해보는 그래프와 이를 합쳐 전체를 비교해보는 그래프를 작성했다.

6개의 정렬 알고리즘을 integer타입의 숫자가 각각 1000개, 5000개, 10000개, 50000개, 100000개, 500000개, 1000000개가 주어졌을 때 10번씩 정렬하면서 걸린 시간의 평균을 성능 비교를 위한 기준으로 삼았다. 이 기준을 10단위의 log scale에 따라 그래프를 만들었다.

## **Ⅴ. 분석 결과 / 느낀점**

먼저, O(logn) < O(n) < O(nlogn) < O(n^2) < O(n^3) < O(2^n) < O(n!)의 순서로 시간복잡도가 느리므로, O(n^2) 알고리즘 3개와 O(nlogn) 알고리즘 3개를 비교했을 때, O(nlogn)이 월등히 낮음을 볼 수 있다. 데이터가 1,000,000개가 주어졌을 때, O(n^2)이 최대 3000초가 넘게 걸리는 반면, O(nlogn)같은 경우, 1초가 넘기지 않는다. 되도록이면 정렬해야 할 데이터가 많을수록 Exchange Sort, Insertion Sort, Selection Sort 알고리즘들을 사용하지 말아야겠다고 느꼈다.

O(n^2) 알고리즘들끼리 비교해보자.

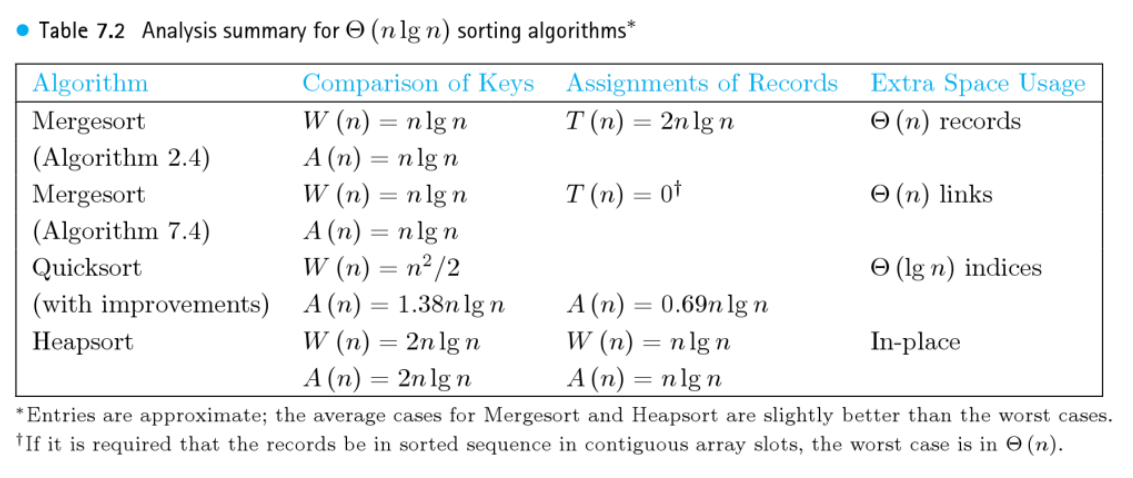


위 표를 빅O 표기법으로 간략화한 표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Time Complexity** | | | **Space Complexity** |
|  | **Best** | **Average** | **Worst** | **Worst** |
| **Exchange Sort** | O(n^2) | O(n^2) | O(n^2) | O(1) |
| **Selection Sort** | O(n^2) | O(n^2) | O(n^2) | O(1) |
| **Insertion Sort** | O(n) | O(n^2) | O(n^2) | O(1) |

위 표(교과서 표 7.1)에 따르면, 키 비교횟수를 기준으로 삽입정렬이 교환정렬과 선택정렬보다 항상 빠르다. 레코드 저장횟수를 기준으로 봤을 때 선택정렬보다 실행속도가 훨씬 빠르고 둘 다 제자리 정렬 알고리즘이므로 최종적으로 이 세 알고리즘 중에서 항상 제일 빠른 알고리즘이 된다. 선택정렬은 교환정렬의 레코드 저장의 단점 중 하나를 보완하여 만들어진 알고리즘이기 때문에, 키 비교횟수는 같지만, 레코드 저장횟수가 교환정렬은 O(n^2), 선택정렬은 O(n)이기에 선택정렬은 교환정렬보다 빠른 알고리즘이 된다.

O(nlogn) 알고리즘들끼리 비교해보자.



위 표를 빅O 표기법으로 간략화한 표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Time Complexity** | | | **Space Complexity** |
|  | **Best** | **Average** | **Worst** | **Worst** |
| **Merge Sort** | O(nlogn) | O(nlogn) | O(nlogn) | O(n) |
| **Quick Sort** | O(nlogn) | O(nlogn) | O(n^2) | O(logn) |
| **Heap Sort** | O(nlogn) | O(nlogn) | O(nlogn) | O(1) |

위 표(교과서 표 7.2)에 따르면, 키 비교횟수와 레코드의 저장횟수를 기준으로 힙 정렬이 삽입정렬이 평균적으로 빠른 정렬보다 나쁘고, 추가 공간 사용량이 O(logn)으로 최소이기 때문에 빠른 정렬이 힙 정렬보다 빠르다. 합병정렬의 경우 전체 입력 레코드 배열 크기만큼의 배열을 추가적으로 사용(O(n))하고 빠른 정렬보다 레코드 저장 횟수가 3배가량 더 많기 때문에, 빠른 정렬의 키 비교횟수가 조금 더 많음에도 빠른 정렬이 합병정렬보다 더 빠르다.

그러나 필자가 작성한 힙정렬의 경우, 힙구조를 만들어서 정렬한 것이 아니라, 기존 데이터 H.S 대신 data배열과 heapsize 대신 n을 이용하여 시간복잡도가 크게 줄어들었다. Heapsize 대신에 n을 사용하면서 removekeys()에서는 for (int i = n; i >= 1; i--) data[i] = Root(n--);으로, Root()에서는 data[1] = data[n]; SiftDown(n - 1, 1); 를 사용하였다.