Algorithm HW1

Randomize, Deterministic Select

2013-11381

강지원

* **구현할 내용**

선택 알고리즘(i번째 작은 숫자 선택) 중, 평균적으로 선형 시간이 소요되는 알고리즘과 최악의 경우에도 선형 시간이 소요되는 알고리즘을 구현한다 평균 선형 시간이 소요되는 선택 알고리즘의 경우, 퀵 정렬의 원리를 이용하고, 피봇은 랜덤으로 정한다. 최악의 경우에도 선형 시간이 소요되는 알고리즘의 경우 마찬 가지로 퀵 정렬의 원리를 이용한다. 피봇을 정할 때는 전체의 리스트를 5개로 나누어 각각의 중간 값들의 중간 값으로 정한다.

* **구현 방법**

구현 언어는 python을 사용하였다. 두 알고리즘에서 피봇을 정하는 함수만 따로 구현하였고 나머지는 모두 같다.

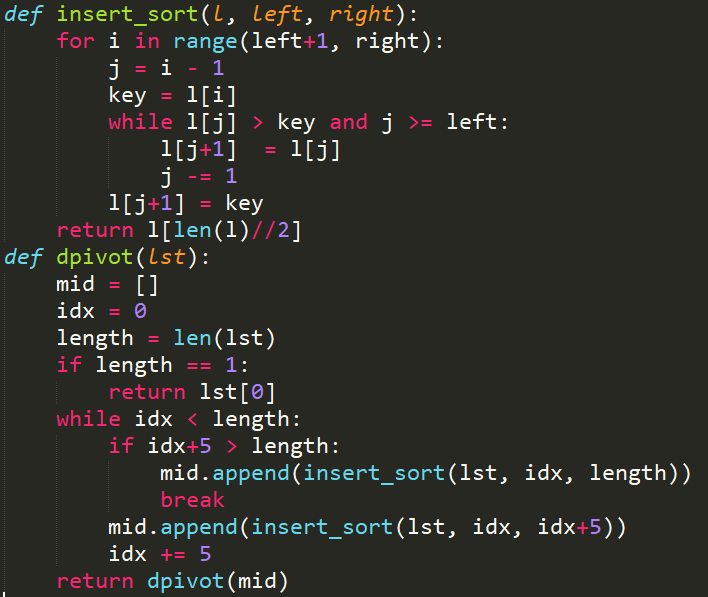
1. **Randomize Select**

피봇을 정하는 부분을, 파이썬의 기본 제공 모듈 random의 randint 함수를 사용하여 구현하였다.



1. **Deterministic Select**

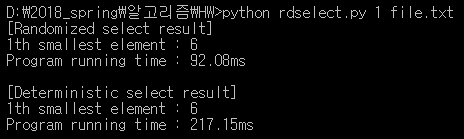
피봇을 정할 부분에서 리스트 전체를 5개의 리스트로 나누고, 각각의 중간 값의 중간 값을 재귀적으로 구하는 방법을 사용하였다. 이때, 5개의 중간 값을 구할 과정은 삽입 정렬 알고리즘을 이용하였다.



* **시간 측정**

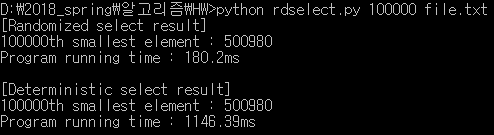
시간을 측정하기 위해서 파이썬 제공 모듈 중, time 모듈을 사용하였다. 또한 input으로 사용한 숫자들은 1에서 1000000 사이의 랜덤 숫자 200000개를 사용하였다.

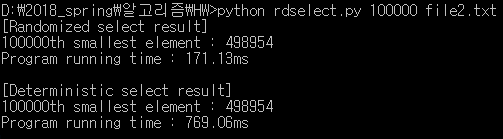
1. 첫번째 실행 – 가장 작은 수 찾기



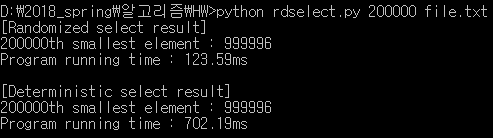


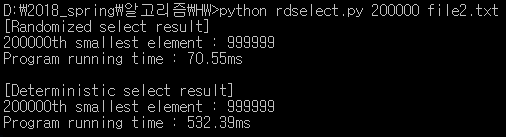
1. 두번째 실행 – 중간 값(100000번째) 찾기





1. 세번째 실행 – 마지막 값(200000번째) 찾기

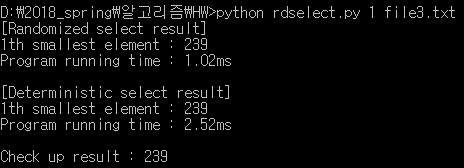


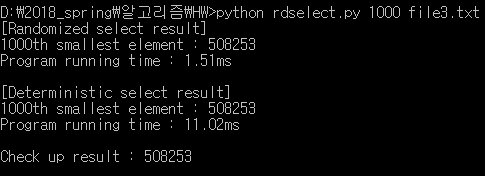


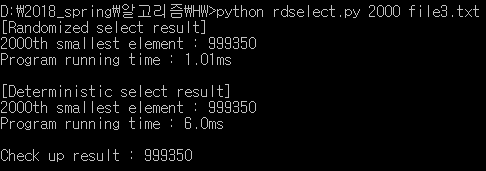
전체적으로 randomize select가 훨씬 빠른 모습을 보인다.

* **Checker Program**

Checker 프로그램은 만들어 놓은 삽입 정렬 알고리즘을 이용해서 구현 하였다. 삽입 정렬이 느려 200000개를 테스트하기는 어렵기 때문에, 2000개로 테스트 하였다.







* **결론**

Randomize select에서, 최악의 경우에 O(n^2)의 복잡도를 가지지만, 평균적으로 O(n)의 복잡도를 가진다. Deterministic select는 최악의 경우에도 O(n)의 복잡도를 가진다. 하지만 피봇을 정함에 있어서 삽입 정렬을 반복적으로 하는 등의 오버헤드가 많고, 숫자가 커지면 최악의 경우를 랜덤으로 선택할 확률이 엄청 낮기 때문에 평균적으로 Randomize select가 더 빠르게 동작하는 것으로 보인다.