시간복잡도

김수진

Bubble sort(stable sort)

- 두개의 인덱스를 비교하여 값을 정렬하는 방법
- 오름차순일 경우 두 인덱스끼리 비교하여 1바퀴 돌면 가장 큰 값이 맨 뒤에 저장됨.
- 최대 $n(n-1)$ / 2번 정렬을 수행, 즉 $O(n^2)$의 성능

$$\sum_{i=1}^{n-1} i = 1 + 2 + 3 + 4 + \cdots + (n-1) = \frac{n(n-1)}{2}$$

Selection sort(unstable sort)

- 최대 최소 모두 *n*(*n*–1) / 2번 정렬을 수행, 즉 항상 O(n^2)
- 버블 정렬보다 2배정도 빠름.(교환횟수 + CPU에서 동작하는 속도)
- Astrachan의 스트링 정렬 (with JAVA) 실험을 보면 버블 정렬은 최근의 CPU에서 삽입 정렬보다 대략 5배, 선택 정렬보다는 40% 정도 느렸다고 한다.

$$\sum_{i=1}^{n-1} i = 1+2+3+4+\cdots+(n-1) = \frac{n(n-1)}{2}$$

https://ko.wikipedia.org/wiki/선택_정렬#/media/File:Selection-Sort-Animation.gif

Insert sort(non-stable sort)

- K번째 원소를 (1 ~ K-1)까지 비교해 적절한 위치에 끼어 넣음
- 평균적으로 $O(n^2)$의 성능
- 정렬되어 있는 구조 -> O(n)
- 역순으로 정렬되어 있는 구조 -> O(n^2)

$$\sum_{i=1}^{n-1} i = 1 + 2 + 3 + 4 + \cdots + (n-1) = \frac{n(n-1)}{2}$$

```
void insertionSort(int[] arr)
{

   for(int index = 1 ; index < arr.length ; index++){

      int temp = arr[index];
      int aux = index - 1;

      while( (aux >= 0) && ( arr[aux] > temp) ) {

         arr[aux+1] = arr[aux];
         aux--;
      }
      arr[aux + 1] = temp;
   }
}
```

Merge Sort(stable sort)

- 마지막 한 개가 될 때 까지 자른 후 자른 순서를 역순으로 크기를 비교해 병합

  ex) A의 배열 크기 N1, B의 배열 크기 N2 -> O(N1+N2) = O(N)

  A의 배열 분할 과정 N1/2, N1/4 … lgN

  합병 단계가 log(2)n 번 만큼 존재

  하나의 합병 단계에서 최대 n번의 비교 연산 필요

  => O(NlogN)

https://ko.wikipedia.org/wiki/합병_정렬#/media/File:Merge-sort-example-300px.gif

Quick sort(unstable sort)

- 배열이 이미 정렬되어 있는 경우 분할이 N만큼 일어나 O(n^2)의 최악의 성능을 가짐
- *1) 나누어지는 족족마다 반씩 분할되는 경우 (the best)*
- *2) 나누어지는 족족마다 1개와 나머지로 분할되는 경우 (the worst)*
- 평균적으로 O(nlogn)의 성능을 가짐
- 평균적인 상황에서는 어떤 알고리즘보다 최고의 성능을 가짐 (퀵정렬의 내부 루프는 대부분의 컴퓨터 아키텍처에서 효율적으로 작동하도록 설계되어 있기때문)

Quick sort(unstable sort)

n 개의 원소인 배열을 정렬할 때 걸리는 수행 시간을 T(n)일때,

퀵 정렬은 재귀적인 방법으로 해결하고 반 씩 나누어 재귀 호출이 이루어지는데,
재귀 호출이 진행하기 전에 비교에 걸리는 시간을 S(n)이라고 한다.
n 개인 원소를 재귀 호출 전에 비교하는 횟수는 n번이므로 S(n)=n이다.

T(n) = 2*T(n/2) + n = n^2T(n/n^2) + 2*n = … = h*n
h = logn

따라서 o(nlogn)!

Heap sort(stable sort)

- 언제나 O(nlogn)을 가짐.
- n번 도는데 logn번 최대값을 찾기 위해 솔트함
- 시간복잡도 증명

10	4	8	5	12	2	6	11	3	9	7	1

최대힙만드는 시간 : O(n)

힙을 내려가면서 비교를 진행 : 2logn

즉, n개의 데이터에 대해 2logn번의 비교를 진행하면서 2nlogn번의 비교 연산이 진행되어 결과적으로 힙 정렬의 시간복잡도는 2nlogn+n이며 O(nlogn)이다.

Radix sort

- 시간복잡도는 O(dn)이다. (n은 데이터 수 , d는 가장 큰 데이터의 자리수)
- 자리수 만큼 포문을 돌고 거기에 데이터 수만큼 또 포문을 돌면서 큐에 집어넣기 때문이다.

Counting sort

- 시간복잡도는 O(n + k)이다. (n은 데이터의 수, k 는 데이터의 최대값, k가 작을수록 선형 시간의 효과)
- 예를 들어 A : 5 6 3 100 1 일 경우, 실제 데이터의 개수인 5가 n이고, 데이터의 최대값이 k이기 때문에 105가 시간복잡도 계산이 된다
- 그 이유를 설명하면 데이터의 개수인 n만큼 원소의 값을 count하고, 0~100만큼 배열을 돌리기 때문이다.

Shell sort

- 평균 시간 복잡도 O(n^1.5)
- 최악의 경우에는 삽입정렬과 똑같은 O(n^2)의 시간복잡도
- 부분적으로 정렬되어 있는 경우 유리