병합 정렬과 기본 퀵 정렬 알고리즘

20201015 컴퓨터공학과 최유림

1. 병합 정렬과 기본 퀵 정렬 알고리즘 소개

두 알고리즘은 기본적으로 분할 정복 알고리즘을 기반으로 한다. 분할 정복 알고리즘이란 문제를 쪼개 하위의 문제들로 나누고, 하위의 문제들의 답을 합쳐 나가며 문제를 해결해 나가는 것을 뜻한다.

A. 병합 정렬

병합 정렬은 배열을 반으로 분할하고 분할된 배열마다 정렬하는 과정을 재귀적으로 반복하여 합치고 이를 통해 정렬된 배열을 얻어내는 알고리즘이다.

B. 기본 퀵 정렬

병합 정렬처럼 배열을 나누고 분할된 배열마다 재귀적으로 다시 분할하고 정렬하며 합치는 방식은 동일하지만, 배열을 나누는 기준이 다르다. 병합 정렬은 배열을 정확히 반으로 분할하는 과정을 반복하지만 퀵 정렬은 pivot이라고 하는 기준 원소를 기준으로 pivot보다 작은 원소와 큰 원소 배열로 분할한다. 기본 퀵 정렬은 보통 배열의 첫 번째 원소를 pivot으로 삼는다.

2. 두 정렬 방법에 따른 birthday.in 정렬 결과 비교

병합 정렬

기본 퀵 정렬

두 정렬 방법 모두 월별 기준으로 잘 정리된 것을 확인할 수 있다.

3. 병합 정렬 선택 이유

병합 정렬을 선택한 이유는 바로 퀵 정렬의 최악의 시간복잡도 측면을 고려했기 때문이다. 기본 퀵 정렬의 피봇은 항상 배열의 첫 번째 요소가 되는데, 만약 해당 피봇보다 큰 데이터들이 많거나 작은 데이터들이 많은 경우에 $o(n^2)$의 시간복잡도를 가지게 된다. 그러나 병합 정렬은 데이터 크기에 무관하게 항상 일정한 속도를 가진다. 따라서 입력으로 들어올 데이터의 개수를 알 수 없는 문제에는 일정한 속도를 보장할 수 있는 병합 정렬이 적절할 것으로 생각했다.

4. 같은 생일자를 찾는 알고리즘

[정렬 전]

정렬 전 배열에서는 순차 탐색 알고리즘을 사용할 수 있다. 따라서 배열의 첫 번째 요소부터 마지막 요소까지 차례대로 모든 요소를 비교하여 탐색하면 된다. 첫 번째 배열 요소와 같은 생일자를 찾기 위해 전체 배열을 탐색하는 순차 탐색 알고리즘을 실행하고, 이를 마지막 요소까지 차례대로 전체 배열 탐색을 실행하면 된다.

[정렬 후]

정렬 후의 배열에서는 이분 탐색 알고리즘을 사용할 수 있다. 생일이 모두 순서대로 정렬되어 있기 때문에 첫 번째 배열 요소부터 같은 생일자를 찾기 위해 전체 배열의 중앙값과 비교한다. 중앙값보다 데이터 값이 작다면 중앙값 기준 왼쪽 배열에서 탐색을 진행하고, 크다면 중앙값 기준 오른쪽 배열에서 탐색을 진행해 나가면 된다. 마지막 배열 요소까지 해당 연산을 반복한다.

5. 알고리즘 correctness 및 efficiency 증명

각각의 알고리즘은 p라는 함수에서 배열 A의 길이 n을 인자로 정상적으로 실행된다고 가정한다.

① 정렬 전 데이터에 대해 같은 생일자를 찾는 알고리즘

[Correctness] Statement : p(n)은 같은 생일자를 찾으면 true를 리턴한다.

Base Case: p(0)은 false를 리턴한다. 0칸의 배열에서 같은 생일자를 찾을 수 없기 때문이다.
Inductive Step: p(n-1)이 성립한다고 가정한다. 배열 내에 같은 생일자가 있다면 true를 return한다.
Inductive Hypothesis: p(n)이 성립한다. 이전의 p(n-1)에서 이미 같은 생일자를 찾았기 때문에 가능하다.
Conclusion: n개의 배열 내에서 같은 생일자를 찾아낼 수 있다.
[Efficiency : $o(n^2)$]
순차탐색하기 위해 각각의 요소는 나머지 배열을 모두 탐색해야 한다. 따라서 이중 for문을 통해 구현되기 때문에 n^2의 시간 복잡도를 가진다.

② 정렬 후 데이터에 대해 같은 생일자를 찾는 알고리즘
[Correctness] Statement : p(n)은 같은 생일자를 찾으면 true를 리턴한다.
Base Case: p(0)은 false를 return한다. 0칸의 배열에서 같은 생일자를 찾을 수 없기 때문이다.
Inductive Step: p(k)(k<n)이 성립한다고 가정한다. 배열 내에 같은 생일자가 있다면 true를 return한다.
Inductive Hypothesis : mid는 배열의 중앙값을 의미한다.
Case1 : A[mid]에 같은 데이터가 있다면 true를 바로 반환한다.
Case2: 찾는 데이터의 값이 A[mid]값보다 작다면 데이터는 A[0] ~ A[mid-1] 내에 있다. 따라서 (mid-1)개의 배열 내에서 같은 생일자를 찾아내는 것은 위의 가정에 의해 성립함을 알 수 있다.
Case3: 찾는 데이터의 값이 A[mid]값보다 크다면 데이터는 A[mid+1] ~ A[n-1] 의 위치에 있다. 따라서 (n-m-1)개의 배열 내에서 같은 생일자를 찾아내는 것은 위의 가정에 의해 성립함을 알 수 있다.
Conclusion: n개의 배열 내에서 같은 생일자를 찾아낼 수 있다.
[Efficiency : $o(nlogn)$]
첫 번째 요소와 같은 데이터를 찾기 위해 이진 탐색을 실행하면 한 개의 요소당 $logn$의 시간 복잡도를 가진다. 모든 요소에 대해 이진탐색을 실행하게 되면 $nlogn$의 시간 복잡도를 가지게 된다.

③ 병합 정렬
[Correctness] Statement : p(n)은 n개의 배열을 정렬한다. 정렬이 성공하면 true를 리턴한다.
Base Case: p(1)은 항상 true를 리턴한다. 1개의 배열은 정렬되어 있기 때문이다.
Inductive Step: p(n/2)가 true를 리턴한다고 가정한다. A[0]<A[1]<…<A[2/n-1]과 A[2/n+1] < … < A[n-1]이 성립한다.
Inductive Hypothesis : p(n)은 true를 리턴한다. 위의 가정에 따라 성립함을 알 수 있다.
Conclusion: 배열은 해당 함수를 통해 정렬될 수 있다.
[Efficiency : $o(nlogn)$]
n개의 데이터를 분할한 배열에서 두 개씩 합치는 단계가 총 $logn$번 발생하고 해당 단계에서 비교하는 연산이 n번 발생하므로 $o(nlogn)$이 된다.

④ 기본 퀵 정렬
[Correctness] Statement : p(n)은 n개의 배열을 정렬한다. 정렬이 성공하면 true를 리턴한다.
Base Case: p(1)은 항상 true를 리턴한다. 1개의 배열은 정렬되어 있기 때문이다.
Inductive Step: p(k)(k<n)가 true를 리턴한다고 가정한다. 따라서 A[0]<A[1]<…<A[k]가 성립한다.
Inductive Hypothesis : pivot 값을 기준으로 pivot보다 작은 값으로 이루어진 k_1 길이의 배열을 정렬하려면 p(k_1)이고, pivot보다 큰 값으로 이루어진 k_2 길이의 배열을 정렬하면 p(k_2)이다. k_1, k_2는 모두 n보다 작으므로 함수는 true를 리턴한다. 따라서 p(n)이 성립됨을 확인할 수 있다.
Conclusion: 배열은 해당 함수를 통해 정렬될 수 있다.
[Efficiency : $o(nlogn)$]
Pivot이 적절한 중간값으로 선정된 경우, n개의 데이터를 pivot을 기준으로 분할한 배열에서 두 개씩 합치는 단계에서 $logn$의 연산이 소요되고 분할된 데이터를 합치는 단계에서 비교하는 연산이 n번 발생하므로 $o(nlogn)$이 된다.