2️⃣

[2주차] Sorting Algorithm

기간

2024.01.15(월) ~ 2024.01.20(토)

(이사 문제로 인해 금, 토 진행 불가)

알고리즘 개념

정렬

선택 정렬(Selection Sort)

- 현재 위치에 들어갈 값을 찾아 정렬하는 배열
- 최소 선택 정렬(Min-Selection Sort, 오름차순), 최대 선택 정렬(Max-Selection Sort, 내림차순)
- **Logic**
  1. 정렬 되지 않은 index의 맨 앞에서부터, 이를 포함한 그 이후의 배열값 중 가장 작은 값을 찾아간다.
  2. 가장 작은 값을 찾으면, 해당 값을 현재의 index 값과 바꿔준다
  3. 다음 index에서 위 과정을 반복한다.
- 시간 복잡도 : O(n^2), 전체 비교를 진행하게 됨
- 공간 복잡도 : O(n), 하나의 배열에서만 진행

삽입 정렬(Insertion Sort)

- 현재 위치에서, 그 이하의 배열들을 비교하여 자신이 들어갈 위치를 찾아 해당 위치에 삽입하는 알고리즘
- **Logic**
  1. 두 번째 index부터 시작하여, 현재 index는 별도의 변수에 저장해주고, 비교 index를 현재 index - 1로 잡는다.
  2. 별도로 저장해 둔 삽입을 위한 변수와, 비교 index의 배열값을 비교한다.
  3. 삽입 변수의 값이 더 작으면 현재 index로 비교 index의 값을 저장해주고, 비교 index를 -1하여 비교를 반복한다.
  4. 삽입 변수의 값이 더 크면, 비교 index + 1에 삽입 변수를 저장한다.
- 시간 복잡도 : O(n) → 이미 정렬이 되어있는 경우, O(n^2) → 최악의 경우
- 공간 복잡도 : O(n)

버블 정렬(Bubble Sort)

- 매번 연속된 2개의 index를 비교하여, 정한 기준의 값을 뒤로 넘겨 정렬하는 방법
- 오름차순으로 정렬할 경우, 비교시마다 큰 값이 뒤로 이동하여, 1바퀴를 돌 경우 가장 큰 값이 맨 뒤에 저장된다.
  - 맨 마지막에 비교하는 수들 중 가장 큰 값이 저장되므로, (전체 배열의 크기 - 현재까지 순환한 바퀴 수)만큼 반복하여 진행한다.
- **Logic**
  1. 두번째 index부터 시작하여, 현재 index 값과 바로 이전의 index값을 비교한다.
  2. 이전 index가 더 큰 경우, 현재 index와 바꿔준다.
     (현재 index가 더 클 경우 교환하지 않고 다음 두 연속된 배열값을 비교한다.)
  3. (전체 배열의 크기 - 현재까지 순환한 바퀴수) 만큼 반복한다.
- 시간 복잡도 : O(n^2)

합병 정렬(Merge Sort)

- 분할 정복(Divide and conquer) 방식으로 설계된 알고리즘
  - Divide and Conquer : 큰 문제를 반으로 쪼개어 문제를 해결해 나가는 방식
  - 배열의 크기가 1보다 작거나 같을 때 까지 반복한다.

- 입력으로 하나의 배열을 받을 때, 연산 중에 2개의 배열로 계속 쪼게어 나간 뒤, 합치면서 정렬해 최후에는 하나의 정렬을 출력
- 분할 과정
  1. 현재 배열을 절반으로 나눈다.
     a. 배열의 시작 위치와, 종료 위치를 입력 받아 둘을 더한 후 2를 나누어 해당 위치를 기준으로 나눈다
  2. 나눈 배열의 크기가 0이나 1일 때까지 반복한다.
- 합병 과정
  1. 두 개의 배열 A, B의 크기를 비교한다.
  2. i에는 A 배열의 시작 index, j에는 B 배열의 시작 index를 저장한다.
  3. A[i], A[j]를 비교한다
     a. 오름차순의 경우 이중에 작은 값을 새로운 배열 C에 저장한다.
     (A[i]가 더 작은 경우, 해당 값을 배열 C에 저장한다)
     b. i나 j 중 하나가 각자 배열의 끝에 도달할 때까지 반복한다.
     c. 끝까지 저장을 못한 배열의 값을, 순서대로 전부 C에 저장한다.
     d. C 배열을 원래의 배열에 저장해준다.
- 시간 복잡도 : O(nlogn) → 분할 : O(n), 합병 : O(logn)

퀵 정렬(Quick Sort)

- Divide and Conquer 사용
  - 분할과 동시에 정렬을 진행
- pivot point라고 기준이 되는 값을 하나 설정하여, 해당 값을 기준으로 작은 값은 왼쪽, 큰 값은 오른쪽으로 옮기는 방식
  - 해당 과정을 반복하여 분할된 배열의 크기가 1이 되면 배열이 모두 정렬된 것
- **Logic**
  1. pivot point로 잡을 배열의 값 하나를 정한다. 보통 맨 앞이나 맨 뒤, 혹은 전체 배열 값 중 중간값이나나 랜덤 값으로 정한다.

2. 분할을 진행하기에 앞서, 비교를 진행하기 위해 가장 왼쪽 배열의 인덱스를 저장하는 left 변수, 가장 오른쪽 배열의 인덱스를 저장한 right변수를 생성한다.
3. right부터 비교를 진행한다. 비교는 right가 left보다 클 때만 반복하며. 비교한 배열값이 pivot point보다 크면 right를 하나 감소시키고 비교를 반복한다.
    a. pivot point보다 작은 배열 값을 찾으면, 반복을 중지한다.
4. 그 다음 left부터 비교를 진행한다. 비교는 right가 left보다 클 때만 반복하며. 비교한 배열값이 pivot point보다 작으면 left를 하나 증가시키고 비교를 반복한다.
    a. pivot point보다 큰 배열 값을 찾으면, 반복을 중지한다.
5. left 인덱스의 값과 right 인덱스의 값을 바꿔준다.
6. 3,4,5 과정을 left<right가 만족 할 때 까지 반복한다.
7. 위 과정이 끝나면 left의 값과 pivot point를 바꿔준다.
8. 맨 왼쪽부터 left - 1까지, left+1부터 맨 오른쪽까지로 나눠 퀵 정렬을 반복한다.

- 시간 복잡도 : O(nlogn) → 일반적, O(n^2) → 최악의 경우
    - 최악의 경우를 방지하기 위해 전체 배열 중 중간값이나 랜덤 값으로 pivot point를 지정한다.
- 일반적으로 퀵정렬이 합병정렬보다 20% 빠르다.

javascript의 정렬 시간 복잡도 : nlogn

공통 문제

- 백준-1377 : 버블 소트

  백준-1377 : 버블 소트 정리

- 백준-10814 : 나이순 정렬

  백준-10814 : 나이순 정렬 정리

- 백준-24062 : 병합 정렬 3

  백준-24062 : 알고리즘 수업 - 병합 정렬 3 정리

개인 문제