MergeSort Report

2015-11323
박성민

설명의 편의를 위해 16개의 Array로 나뉜 경우로 한정해서 말하며, 16개 미만으로 나뉜 경우는 boolean 배열을 통해 적절히 처리하였다.

먼저 MergeSort로 나눠진 16개의 배열을 합치기 위해 heap을 이용할 때 단순히 기존 Array를 활용한다면 heap 과정에서 기존의 16개의 배열의 index를 보존하기 힘들기 때문에 LinkedList를 이용하여 기존 16개의 Array를 16개의 LinkedList로 바꿨다. 따로 선언한 LinkedList Class는 사실 의미상 Node Class에 가깝고, LinkedList의 배열을 통해 minimum 값을 O(1)로 구하므로 head는 따로 구현하지 않았다. 16개의 LinkedList를 만들 때 각각 가장 처음 선언한 Node를 heap array에 각 index별로 넣고, 모든 배열이 LinkedList로 변환이 끝난 뒤 heapify를 통해 minheap으로 만들었다. heapsort가 아닌 단순히 minheap 구조로 만들기만 하면 되므로, heapify는 heapsize(현재 케이스에선 16, 나뉜 Array의 수)를 2로 나눈 값부터 root까지 percolateDown을 반복하고, percolateDown에서는 불린 Node의 왼쪽 자식 Node가 존재할 경우 만약 오른쪽 자식 Node도 존재하고 오른쪽 자식 Node의 값이 더 작을 경우 오른쪽 자식 Node를 비교 대상으로, 그 외에는 왼쪽 자식 Node를 비교 대상으로 두었다. 이 비교 대상과 불린 Node의 값을 비교하여 불린 Node의 값이 더 큰 경우 두 Node 값을 바꾼 뒤, 바뀐 자식 Node에서 recursive하게 percolateDown을 부르는 식으로 구현했다. 이 과정을 heapify를 통해 전체 크기의 절반에 대해서 아래에서 위 순서로 시행하면 완전 정렬은 아니지만 heap 구조, 즉 부모 Node가 자식 Node들 보다 항상 작은 구조가 완성되고, 16개의 정렬된 Array의 가장 첫 값을 넣었으므로 minheap의 root Node가 항상 16개의 Array중 가장 작음이 자명하다.

이후 heap의 크기가 0이 될 때까지 heap의 root Node의 값을 반환할 배열에 넣어주는 행동을 반복하는데, LinkedList를 사용했기 때문에 heap[0].getNext()가 존재한다면 heap[0] 값을 heap[0].getNext() 값을 넣어주고 rootNode 이외에는 모두 minheap 구조를 만족하기 때문에 root만 percolateDown을 호출하면 다시 모든 node에서 minheap 구조를 만족한다. 만약 한 LikedList의 모든 값이 반환할 배열에 들어간다면, heap[0].getNext()는 null을 반환하고, 이 경우 현재 heap의 가장 끝 값과 root Node를 바꾼 뒤 heapsize를 하나 줄이고, 다시 rootNode에서 percolateDown을 호출한다. 이 과정을 heapIndex가 0이 될 때까지 반복하면 16개의 Array를 효율적으로 한번에 합칠 수 있다.

구현 과정에서 일관된 하나의 함수로 16개로 나뉘는 경우와 16개 미만으로 나뉘는 경우를 처리하려 하니 존재 가능 여부를 check하고, Array들로 나누는 등의 오버헤드가 들어갔지만 측정 결과 단순히 16개로 나누고 합쳤을 때보단 2배 가량 빠르고, 단순히 2개로 나누고 합쳤을 때보단 비슷하거나 1.3배정도 느렸다. Input이 Array로 한정되어 있지만 만약 Input이 임의 접근이 가능한 LinkedList로 들어온다면 변환하는 overhead가 줄어 16개로 나눈 경우가 더 빠를 수 있을 것이다.