□ 함수 설명 및 시간 복잡도

|  |  |
| --- | --- |
| **자료구조**  **함 수** | **Heap** |
| Insert | heap에 data삽입하는 메서드. 마지막 index에 삽입하고 percolation up진행  data의 우선순위가 가장 높을 경우 root까지 올라가므로 트리의 높이만큼 while문  반복하므로 O(lnN), data가 우선순위가 가장 낮은 경우 O(1) |
| Delete | heap에서 data삭제하는 메서드. 가장 앞에있는 index data는 우선순위가  가장 높으므로 해당 데이터를 지운 후 last index element를 root로 copy한 후  percolation down 진행. 왼쪽자식, 오른쪽자식 우선순위 비교는 O(1),  마지막 데이터를 옮겨왔으므로 lowest depth까지 갈 가능성 높다.  높이만큼의 while문 반복 O(lnN) |
| Top | heap에서 가장 앞에있는 element 출력  배열 1번째 접근하면 되므로 O(1) |
| Pop | heap에서 가장 앞에있는 element 삭제  heap ADT delete메서드 호출, delete메서드와 시간복잡도 동일 |
| Push | heap에 data 추가  heap ADT Insert메서드 호출. Insert메서드와 시간복잡도 동일 |
| Print | heap BFS 순회하며 출력  모든 원소 접근하므로 O(N) |
| Heapification | 정렬되지 않은 배열을 Inplace방식으로 Min-heap으로 변환  leaf node는 trivial heap이므로 n/2번째부터 heapification 진행.  n/2번째 원소부터 부모와 우선순위 비교해가며 진행(for문 n/2 ~ 1까지 반복O(N)) 후 1번째부터 다시 n/2번째까지 진행(for문 1~n/2까지 반복O(N))  시간복잡도 O(N) |
| HeapSort | Min-heap된 배열을 이용하여 내림차순으로 정렬  1번째 원소 pop하고 1번방에 last element copy한 후 root부터 percolation down. 1번째 원소는 배열에 맨 마지막에 pop된 원소 앞에 넣는다.  cIdx에 pIdx의 자식들 중 높은 우선순위 index를 선택해서 부모와 비교하며  진행. 시간복잡도는 while문으로 데이터 개수만큼 반복하므로 O(N)\*percolationO(ln(N))이므로 O(NlnN) |