**Lab 3**

1. **How would you represent a d-ary heap in an array? (10%)**

A d-ary heap can be represented in an array similar to a binary heap. In a binary heap, each non-leaf node i has its children at indices 2\*i + 1 and 2\*i + 2. For a d-ary heap, each non-leaf node i will have its children at indices d\*i + 1, d\*i + 2, ..., d\*i + d.The parent of node i (where i > 1) is stored at index floor((i-1)/d) + 1.

1. **What is the height of a d-ary heap of n elements in terms of n and d? (10%);**

The height h of a d-ary heap with n elements can be approximated using the formula for the sum of a geometric series:

*n*≤1+*d*+*d*2+⋯+*dh*=*(d(h*+1))−1​)/(d-1),namely *h*≤log*d*​(*n*(*d*−1)+1)−1

Thus,the height of the heap can be O(log\_d n),using the upper bound.

**3. Given an efficient implementation of EXTRACT-MAX in a d-ary max-heap.**

**Analyze its running time in terms of d and n. (20%);**

EXTRACT-MAX involves replacing the root (maximum element) with the last element of the heap, reducing the heap size, and then "heapifying" down from the root to restore the heap property.

Running Time Analysis:

The replacement and size reduction are *O*(1) operations.

Heapifying down from the root involves traversing the tree from top to bottom,taking *O*(*h*) time.Since *h*=*O*(log*d*​*n*),the overall running time of EXTRACT-MAX is *O*(log*d*​*n*).

**4. Given an efficient implementation of INSERT in a d-ary max-heap. Analyze its**

**running time in terms of d and n. (20%);**

INSERT involves appending the new element to the end of the heap array and then "heapifying" up from the new node to restore the heap property.

Running Time Analysis:

Appending the new element is an *O*(1) operation.Heapifying may traverses the tree

from bottom to top,also taking *O*(*h*) time.Therefore, the overall running time of INSERT is *O*(log*d*​*n*).

**5. Give an efficient implementation of INCREASE-KEY(A, i, k), which first sets A[i]**

**max(A[i], k) and then updates the d-ary max-heap structure appropriately. Analyze**

**its running time in terms of d and n. (20%);**

INCREASE-KEY first updates the key at index i to be the maximum of the current key and k, and then "heapifies" up from index i to restore the heap property.

Running Time Analysis:

Updating the key is an *O*(1) operation.Heapifying may traverses the tree from index i to top, taking *O*(*h*) time.Thus, the overall running time of INCREASE-KEY is *O*(log*d*​*n*).

**6.Doucument(20%)**

The code is in the document named “d-ary heap.cpp”.