CS225 Assignment 4

Group 18: Xu Ke, Wang Dingkun, Qian shuaicun, Hua Bingshen

March 2021

1 Assignment 4 Exercise 1

1.1

We first consider the worst case, we insert a very large element when building a maxheap (vice versa). Therefore, every time the element compares with its parent node, their values will be swapped until the root element has been compared. Obviously, the number of comparisons is the degree of the heap $log(n)$, i.e., the time complexity is $O(log(n))$.

1.2

From the bottom to the top, from the right to the left, we make altogether $log(n)$ comparisons for every pair of children and its parent, because the degree of the heap is $log(n)$. Consider the path along the elements have been swapped, i.e., the path from the root node to leaf node. The complexity for binary search is $O(log(n))$ and there are $log(n)$ elements on the path. Therefore, the complexity for finding the proper position for root node is $O(log(log(n)))$. Thus, $log(n) + O(log(log(n)))$ comparisons are required.

2 Assignment 4 Exercise 2

2.1

The terminology "associative array" is motivated by considering the elements $e \in S$ as elements of an array of length $|S|$. And from lecture, we know that hash table is an associative array consists in providing a function $h : K \to (0, ..., m-1)$ mapping keys to small natural numbers. With a hash table, we can sort a binary relation easily. Take a binary relation set $(a_1, b_1), (a_2, b_2), ..., (a_n, b_n)$ for example, we can firstly initialize the hash table considering "a_i" as the key. By choosing the length of set and doing mod operation, we can locate each pair in the hash table (e.g. as shown below), where a_3 mod length has the same value as a_5 mod length.

After finishing this, we need to do the judgement. It is clear that for a binary relation (a_i, b_i), if it is symmetric we must have (a_j, b_j) in the set with

$h(1)$	$h(2)$	$h(3)$	...	h(n)
(a_1, b_1)	(a_3, b_3)	(a_2, b_2)	...	...
	(a_5, b_5)	(a_4, b_4)	...	...
		(a_6, b_6)	...	...
		...	...	...

$a_i = b_j, a_j = b_i$. Thus, it must satisfy that if such (a_j, b_j) exist, it must locate in the column with $h(i) = b_i$ mod length. Here, we can use getlength() function that we already implemented for Doubly-Linked List to find out the depth. Thus if set is symmetric, for each binary relation in the set, we must can find its corresponding one before we traverse the column throughout the whole depth, otherwise the set is not symmetric.

2.2

Please see the code in the file attached.

3 Assignment 4 Exercise 4

Since for the McGee heaps, the implementations of the operations are the same as for Fibonacci heaps, except that insertion and union consolidate the root list as their last step, we can conclude that the there are at most only two factors that can lead to the differences of running times, that is Insertion and Union operation. Let's discuss them in detail :

3.1 Insertion operation:

For the insertion operation, we need to do the following step:

(1) Put the node in the list of roots;

```
HEAP-INSERT(H, x)
  degree[x] ← 0
  p[x] ← NIL
  child[x] ← NIL
  left[x] ← x
  right[x] ← x
  mark[x] ← FALSE
  concatenate the root list containing x with root list H
  if min[H] = NIL or key[x] < key[min[H]]
     then min[H] ← x
  n[H] ← n[H] + 1
```

(2) Consolidate the root list;

```
CONSOLIDATE(H)
```

```
for i ← 0 to D(n[H])
     do A[i] ← NIL
for each node w in the root list of H
     do x ← w
        d ← degree[x]
        while A[d] not = NIL
           do y ← A[d]
              if key[x] > key[y]
                 then exchange x ← y
              FIB−HEAP−LINK(H,y,x)
              A[d] ← NIL
              d ← d + 1
        A[d] ← x
min[H] ← NIL
for i ← 0 to D(n[H])
     do if A[i] not = NIL
           then add A[i] to the root list of H
                if min[H] = NIL or key[A[i]] < key[min[H]]
                   then min[H] ← A[i]
```

For step(1), its complexity is $O(1)$. And for step(2), its complexity is $O(log(n))$. Thus, the total complexity is $O(log(n))$.

3.2 Union operation:

For the union operation, we need to do the following step:

(1) Union two heaps;

```
HEAP−UNION(H1,H2)
   H ← MAKE−HEAP()
   min[H] ← min[H1]
   concatenate the root list of H2 with the root list of H
   if (min[H1] = NIL) or (min[H2] not = NIL and min[H2] < min[H1])
   then min[H] ← min[H2]
   n[H] ← n[H1] + n[H2]
   free the objects H1 and H2
   return H
```

(2) Consolidate;

```
CONSOLIDATE(H)
for i ← 0 to D(n[H])
     do A[i] ← NIL
for each node w in the root list of H
```

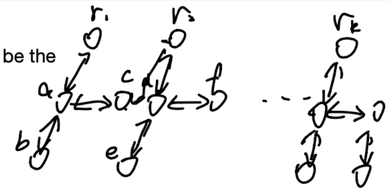
```
      do x ← w
         d ← degree[x]
         while A[d] not = NIL
            do y ← A[d]
               if key[x] > key[y]
                  then exchange x ← y
               HEAP-LINK(H,y,x)
               A[d] ← NIL
               d ← d + 1
         A[d] ← x
min[H] ← NIL
for i ← 0 to D(n[H])
   do if A[i] not = NIL
      then add A[i] to the root list of H
         if min[H] = NIL or key[A[i]] < key[min[H]]
                  then min[H] ← A[i]
```

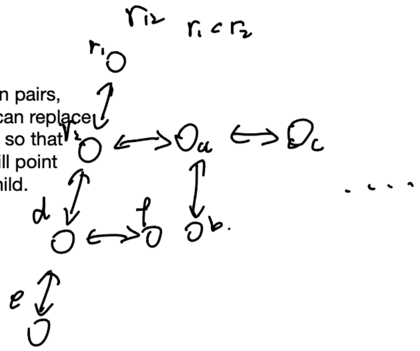
For step(1), its complexity is $O(1)$. And for step(2), its complexity is $O(log(n))$. Thus, the total complexity is $O(log(n))$.

Thus, we can conclude that the worst-case running times of operations on McGee heaps may cost at most $O(log(n))$.

4 Assignment 4 Exercise 3

i)
1.The deletemin is delete the top root and its children will be the top roots then it will remain k subtrees.

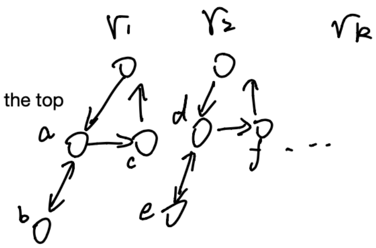




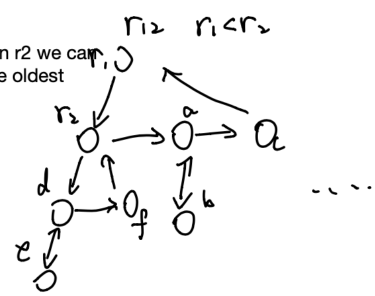
2.Combines the subtrees in pairs, if r1 is smaller than r2 we can replace the oldest child of r1 to r2, so that r1 will point to the r2, r2 will point to the r1 and r1's oldest child.

3.Combines the r12, r34,....... from right to left one by one until left 1 new heaps. The time complexity is O(logn).

ii)
1.The deletemin is delete the top root and its children will be the top roots then it will remain k subtrees.



2.Combines the subtrees in pairs, if r1 is smaller than r2 we can replace the oldest child of r1 to r2 and r2 point to the oldest child of r1.



3.Combines the r12, r34,....... from right to left one by one until left 1 new heaps. The time complexity is O(logn).