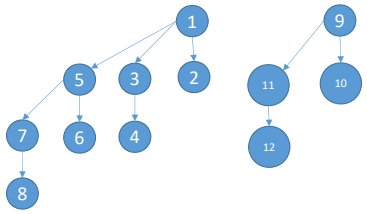
EC 504
Spring, 2021
HW 3

Due Monday, March 1, 8PM on Gradescope.

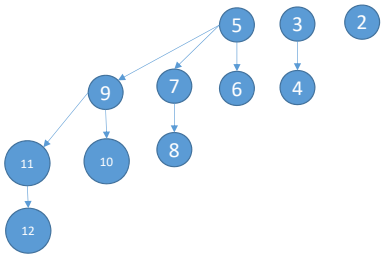
1. (15 pts) Draw the Binomial heaps that result at the end of each of the following sequence of operations:

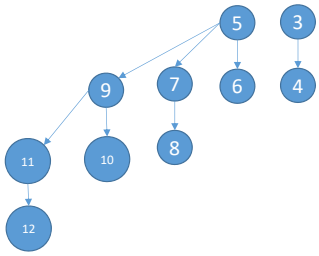
   - Insert 1,2,3,4,5,6,7,8,9,10,11,12 using non-lazy inserts (merging when necessary).
   - Then delete 1 (note it is the minimum, so this is extract min)
   - Then delete 2. (again, this is extract min)

   Solution: See pictures below.



Solution for Problem 1, part 1





Solution for parts 2 and 3.

2. (15 pts) Draw the Fibonacci heaps that result from the following sequence of operations. Don't bother drawing the circular linked lists at each level of the trees, only at the root level.

   - insert 1,2,3,4,5,6,7,8,9,10,11,12.
   - then delete 1 (extract min);
   - then change 8 to 0 (reduce key);
   - then change 7 to 1.

   Solution: See picture below, which shows the heap after insert all, after delete 1, after changing 8 to 0, and at the end. The green node is "marked".

3. (15 pts) Draw the rank-pairing heaps that result from the following sequence of operations.

   - insert 1,2,3,4,5,6,7,8,9,10,11,12.
   - then delete 1 (extract min) with recursive merging;
   - then change 8 to 0 (reduce key);
   - then change 7 to 1.
   - then extract min (delete 0) with recursive merging;

   Solution:

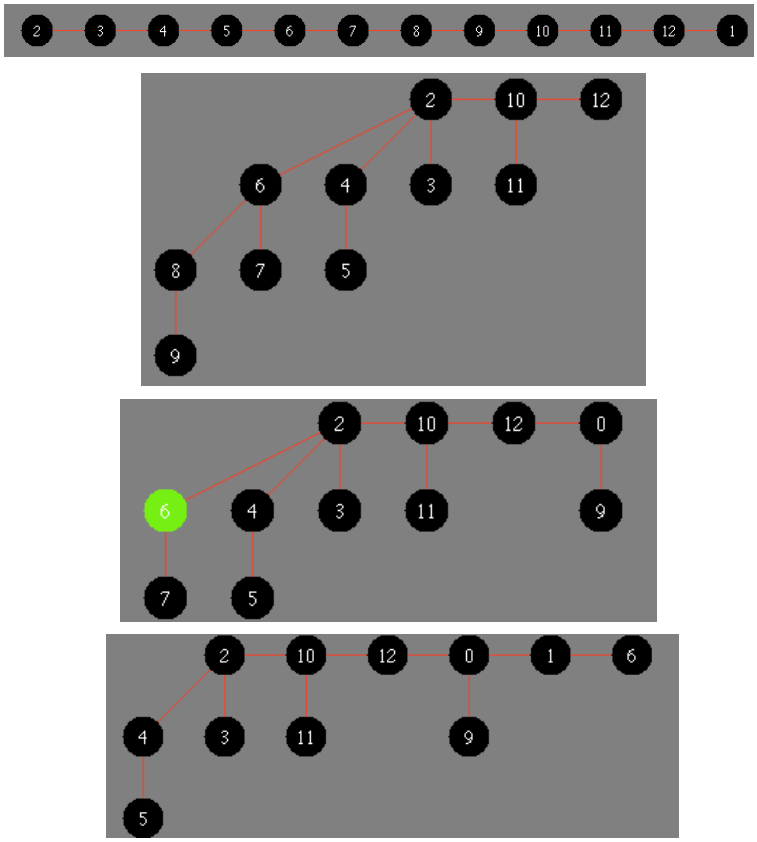
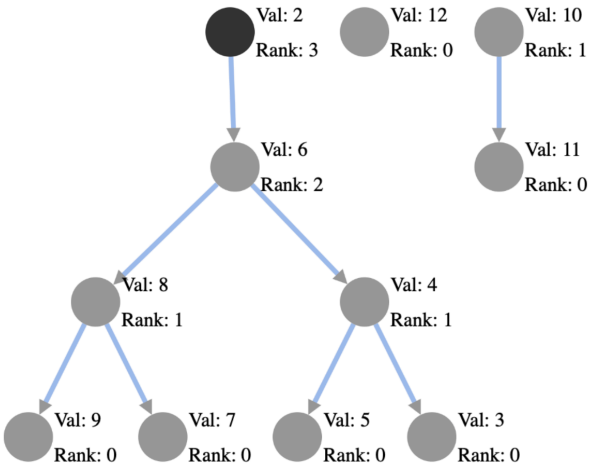
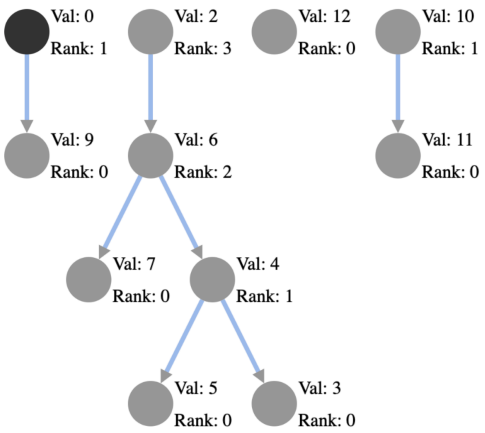


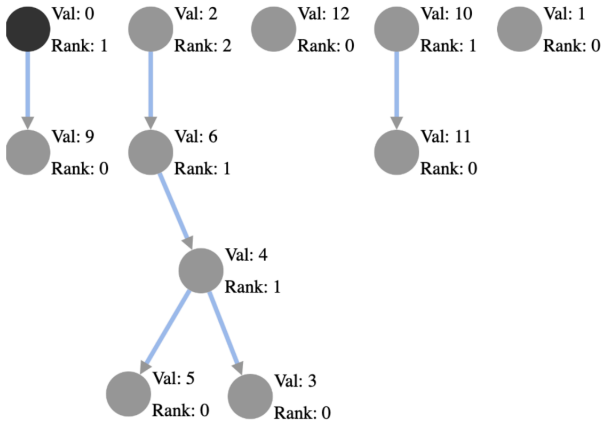
Figure 1: Solution to Problem 2



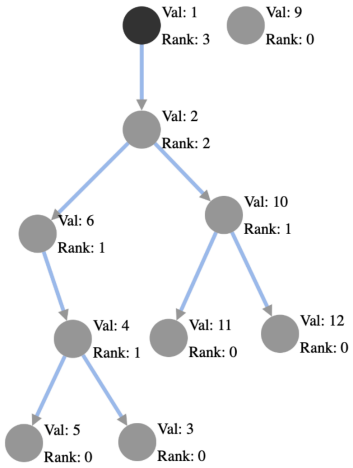
Solution to Problem 3(a)



Solution to Problem 3(b)



Solution to Problem 3(c)



Solution to Problem 3(d)

4. (10 pts) Given input 4371, 1323, 6173, 4199, 4344, 9679, 1989 and a hash function $h_1(x) = x \mod 10$, and a second hash function $h_2(x) = (x/10) \mod 10$, show the following:

   (a) The result of inserting these into a hash table with 10 entries with hash function $h_1(x)$ using linear probing.

   (b) The result of inserting these into a cuckoo hash table with two tables of length 10, with hash functions $h_1(x), h_2(x)$ respectively.

Solution:

Part (a):

```
9679
4371
1989
1323
6173
4344
xxxx
xxxx
xxxx
4199
```

Part (b):

```
xxxx    xxxx
4371    xxxx
xxxx    1323
6173    xxxx
4344    xxxx
xxxx    xxxx
xxxx    xxxx
xxxx    9679
xxxx    xxxx
1989    4199
```

5. (15 pts) A min-max heap is a complete binary tree containing alternating min (or even) and max (or odd) levels. Even levels are for example 0, 2, 4, etc, and odd levels are respectively 1, 3, 5, etc. We assume that the root element is at the level 0. This heap maintains the following heap order: if a key is in position k at an even level, it is less than or equal to the keys of its children. If a key is in position k at an odd level, it is greater than or equal to the keys of its two children.

   We represent a min-max heap with N elements as an array[0:N-1]. To add an element to a min-max heap perform following operations: Append the required key to (the end of) the array representing the min-max heap, denoted as position k. This will likely break the min-max heap properties, therefore we need to adjust the heap. Then, do a modified upheap:

```
If k is not root:
    if k is on min-level:
        if h[k]> h[parent[k]]:
            temp = h[k]; h[k]=h[parent[k]]; h[parent[k]]=temp;k=parent[k];
            while k has grandparent and h[k] > h[grandparent[k]]:
```

```
            temp = h[k]; h[k] = h[grandparent[k]]; h[grandparent[k]]=temp;
            k=grandparent[k];
    else:
            while k has grandparent and h[k] < h[grandparent[k]]:
            temp = h[k]; h[k] = h[grandparent[k]]; h[grandparent[k]]=temp;
            k=grandparent[k];
else: // k is  on max level
        if h[k]< h[parent[k]]:
            temp = h[k]; h[k]=h[parent[k]]; h[parent[k]]=temp;k=parent[k];
            while k has grandparent and h[k] < h[grandparent[k]]:
            temp = h[k]; h[k] = h[grandparent[k]]; h[grandparent[k]]=temp;
            k=grandparent[k];
    else:
            while k has grandparent and h[k] > h[grandparent[k]]:
            temp = h[k]; h[k] = h[grandparent[k]]; h[grandparent[k]]=temp;
            k=grandparent[k];
```

(a) Show that, in min-max heaps, the root element has the smallest key in the heap, and one of the two elements in the level 1 is the largest key in the heap has the maximum key in the heap. Hence finding both min or max is O(1).

Solution:

Suppose the smallest key is inserted on a min-level other than 0. Then, its key will be smaller than its parent, so it will be bubbled up in an upheap move with its grandparent, and so recursively, until it reaches level 0. Hence, the smallest key will end up in position 0 if inserted into a min-level.

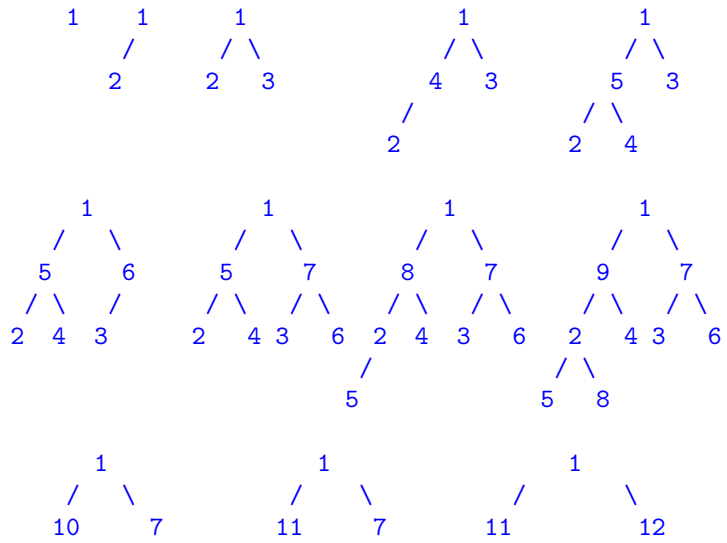
Suppose the smallest key is inserted into a max-level. Then, it will be smaller than its parent, so it will be swapped with its parent, and it will be in a min-level. From there, it will be swapped with its grandparent, recursively, until it reaches level 0. Thus, the smallest key will always end up in position 0.

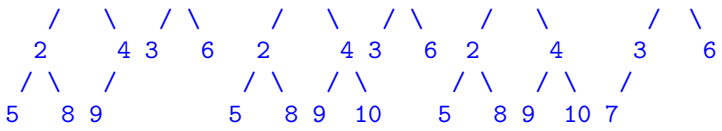
Suppose the largest key is inserted into a max-level. It will be larger than its parent, so it will be swapped with its grandparent recursively until it reaches level 1, where it no longer has a grandparent.

Suppose the largest key is inserted into a min-level. It will be larger than its parent in a max-level, so it will be swapped with its parent, and then swapped with its grandparent recursively until it reaches level 1, where it no longer has a grandparent. In either case, the largest key winds up in level 1, which is one of positions 1, 2 in the heap.

(b) Show the sequence of min-max heaps that occur when you insert 1,2,3,4,5,6,7,8,9,10,11,12.

Solution:





6. (10 points) For the KMP algorithm, Compute the prefix function for the following string: "abracadabra".

7. (10 points) For the Aho-Corasick algorithm, compute the suffix trie that would be used to search for the following strings simultaneously:

```
"the"
"other"
"otter"
"often"
"threw"
"these"
"rocks"
```

Solution: See figure below

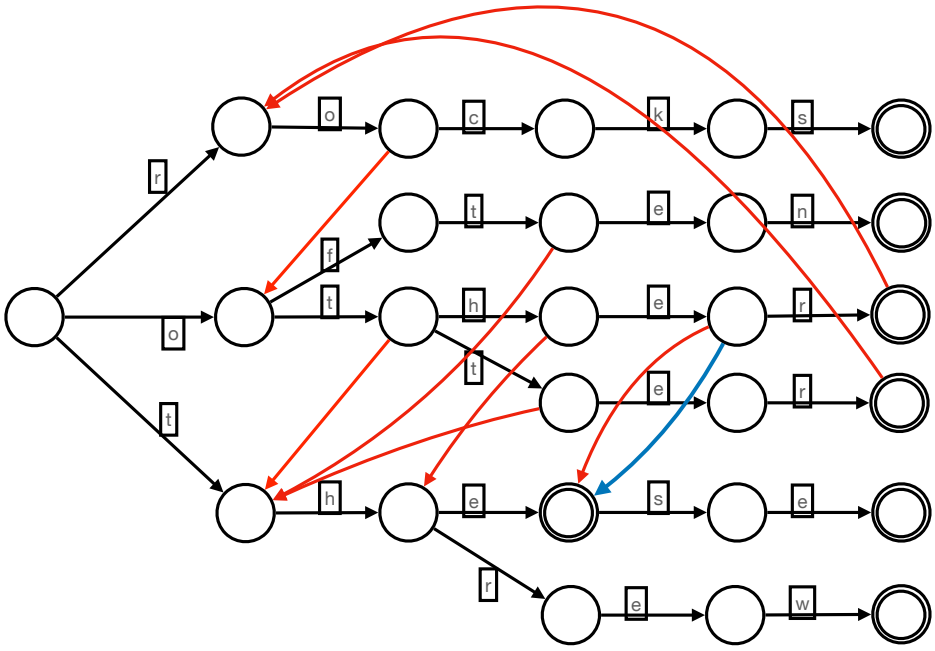
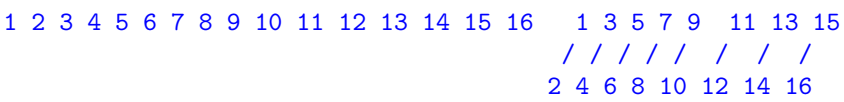


Figure 2: Solution for Problem 7. Suffix links to root not shown.

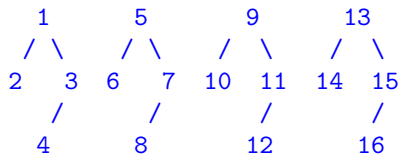
8. (10 points) Show the data structure that results by the disjoint set operations for the following programs.

   (a) Assume we will do union by rank, and that, when we do find, we will do path compression. Insert nodes $1, 2, 3, \ldots, 16$. Add relations between nodes 1-2, 3-4, ..., 15-16, and form the unions, where we merge by rank. When it is a tie of ranks, make the root the smaller of the two root numbers.
   Solution:



(b) add relations 1-3,5-7, 9-11, 13-16 and show the disjoint sets that result from the unions using these relations.

Solution:



(c) Add relations 4-8, 12-16, 6-16 in that order and show the disjoint sets that result.

Solution: