Before you start:

Homework Files

You can download the starter files for coding as well as this *tex* file (you only need to modify *homework2.tex*) on canvas and do your homework with latex. Or you can scan your handwriting, convert to pdf file, and upload it to canvas before the due date. If you choose to write down your answers by hand, you can directly download the pdf file on canvas which provides more blank space for solution box.

Submission Form

A pdf file as your solution named as VE281_HW2_[Your Student ID]_[Your name].pdf uploaded to canvas
Estimated time used for this homework: **3-4 hours.**

0 Student Info

Your name and student id:

Solution: Gong Zimu 520370910037

1 Choices (16 points)

1. For hashing the array (7,34,55,25,64,46,20,10), if H (K)=K%9 is used as the hash function, how many pairs of elements are hashed with address 1?

   A. 1

   B. 2

   C. 3

   D. 4

   Solution: D

2. Let the length of the hash table be 14, the hash function be H(key)=key%11, and the existing data in the table have four key words: 15,38,61,84. Now, the element with key word 49 is added to the table, and the conflict is resolved by two probing methods. Which position is the place to put in?

   (a) 8

   (b) 3

   (c) 5

   (d) 9

   Solution: (a) (d)

3. Which of the following statements about hash lookup is not true?

   (a) With the chain-address approach, it takes the same amount of time to find an element

   (b) When using the chain address method to handle conflicts, if the insertion rule is always at the beginning of the chain, the time to insert either element is the same

   (c) Using the chain address method to deal with the conflict will not cause the secondary aggregation phenomenon

   (d) The chain address method is used to deal with conflicts, which is suitable for uncertain table length

Solution: (a) (b)

4. Have a set of data (15,9,7,8,20, 1,7,4), established by screening method descending sort heap sort of initial heap for ()

   (a) -1,4,8,9,20,7,15,7
   (b) -1,7,15,7,4,8,20,9
   (c) -1,4,15,9,20,7,7,8
   (d) -1,4,7,8,20,15,7,9

Solution: (d)

2 Tree and Heap (34 points)

2.1 Leaf (2+3+3+3 points)

1. A proper binary tree has n 2-degree nodes, what is the number of leaf nodes?

Solution: $n + 1$

2. Try to prove that for a heap with n nodes, the index of the leaves goes from $floor(n/2) + 1$ to n

Solution:
First, the index of the leaves cannot be greater than n.
Second, consider the case with most leaves,
where the heap is a perfect binary tree with n nodes.
Let h be the height, then $n = 2^{h+1} - 1$, and there are 2^h leaves.
Substitute h with n, we have $2^h = (n + 1)/2$, then the index of the first leave is:
$n - (n + 1)/2 + 1 = n/2 + 1/2 = floor(n/2) + 1$, since n is odd.
Thus, the index of the leaves goes from $floor(n/2) + 1$ to n.

3. Try to prove that for a heap with n nodes, there are at most $ceiling(\frac{n}{2^{(h+1)}})$ nodes with height h.

Solution:
Consider the extreme case where the heap is a perfect binary tree with n nodes.
Let H be the height, then $n = 2^{H+1} - 1$, and there are 2^{H-h} nodes with height h.
Substitute H with n, we have $2^{H-h} = \frac{n+1}{2^{(h+1)}} = ceiling(\frac{n}{2^{(h+1)}})$, since n is odd.
Thus, there are at most $ceiling(\frac{n}{2^{(h+1)}})$ nodes with height h.

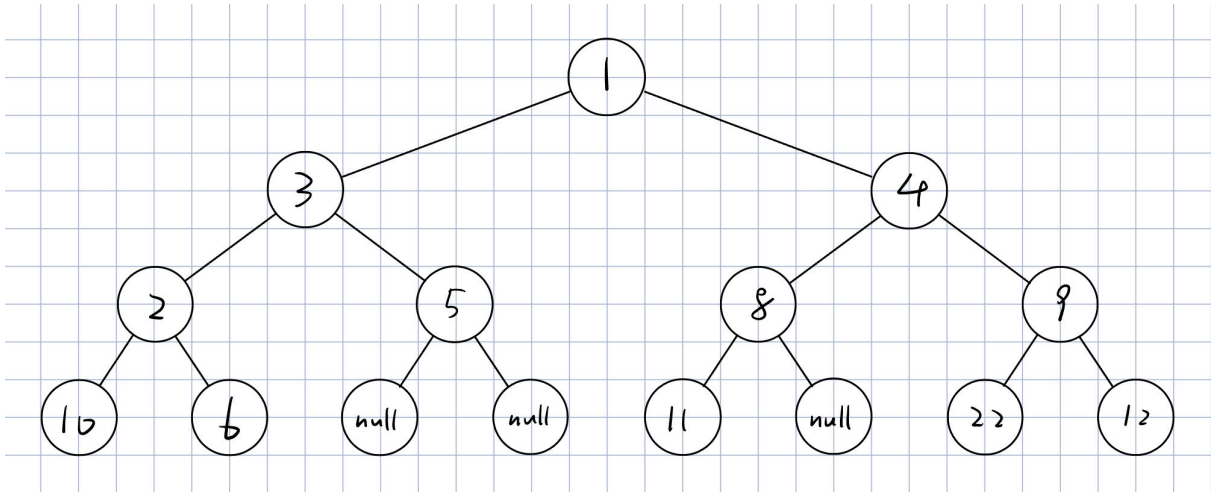
4. What's the number of leaf nodes for a complete binary tree with n nodes? (Elaborate your solution)

Solution:
Suppose there are n_0 nodes with degree 0, n_1 nodes with degree 1, and n_2 nodes with degree 2, then we have $n = n_0 + n_1 + n_2$
The total degrees of the tree X equals to the number of edges of the tree, then we also have
$x = 0 \times n_0 + 1 \times n_1 + 2 \times n_2 = n - 1$
Combine these two equaions we can get $n_2 = n_0 - 1$
We also know that $n_1 = 0$ or 1 in a complete binary tree, then we have
number of leaf nodes $n_0 = floor((n+1)/2)$

2.2 Traversal (5 points)

Given the pre-order traversal $\{1, 3, 2, 10, 6, 5, 4, 8, 11, 9, 22, 12\}$
and in-order traversal $\{10, 2, 6, 3, 5, 1, 11, 8, 4, 22, 9, 12\}$, draw the tree and write down the post-order and level-order traversal (null node marked).

Solution:



post-order traversal $\{10, 6, 2, 5, 3, 11, 8, 22, 12, 9, 4, 1\}$
level-order traversal $\{1, 3, 4, 2, 5, 8, 9, 10, 6, 11, 22, 12\}$

2.3 Initial(8 points)

If we want to initialize a minHeap with n elements. Recall that the min heap initialization algorithm can reduce the time complexity from $O(n \log n)$ to $O(n)$. Now lets us apply this algorithm to insert the following integers. 100, 76, 50, 12, 21, 42, 44, 93, 91, 51, 26, 29, 99, 88, 64, 70, 78, 19, 31. What is the total number of function call percolateDown and the number of swaps operation in it?

Solution:
number of function call percolateDown: 9
number of swaps operation: 12

2.4 Structure (10 points)

What is the number of possible proper binary tree structures giving the fact that it has 13 nodes? (Elaborate your solution. This question is **difficult**!)

Solution:
Height = 3, $n_3 = \binom{3}{4} = 4$
Height = 4,
Distribution 1-2-1-2, $n = 1 \times 2 \times 1 \times 2 = 4$
Distribution 1-1-2-2, $n = 1 \times 2 \times 1 \times \binom{2}{4} = 12$
Distribution 1-2-2-1, $n = 1 \times 1 \times \binom{2}{4} \times \binom{1}{4} = 24$
$n_4 = 4 + 12 + 24 = 40$
Height = 5,
Distribution 1-2-1-1-1, $n = 1 \times 1 \times 4 \times 2 \times 2 = 16$
Distribution 1-1-2-1-1, $n = 1 \times 1 \times 2 \times 4 \times 2 = 16$
Distribution 1-1-1-2-1, $n = 1 \times 1 \times 2 \times 2 \times 4 = 16$
Distribution 1-2-1-1-1, $n = 1 \times 1 \times 2 \times 2 \times 2 = 8$
$n_5 = 16 + 16 + 16 + 8 = 56$
Height = 6,
$n_6 = 1 \times 2 \times 2 \times 2 \times 2 \times 2 = 32$
$N = n_3 + n_4 + n_5 + n_6 = 4 + 40 + 56 + 32 = 132$

3 Hashing Zoo (20 points)

Suppose Prof. Blue Tiger is using a hash table to store information about the grades of his students. The keys are strings and the values are integers. Furthermore, he uses a very simple function t where the hash code of a string is its length. For example:

- t("Blue Tiger") = 10
- t("Red Flamingo") = 12
- t("Glass Frog") = 10

One of his TA gives another function t where the hash code of a string is the integer representing its last letter. For example:

- t("Blue Tiger") = 17
- t("Red Flamingo") = 14
- t("Glass Frog") = 6

And we have:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

3.1 Hash Function (3 points)

Which function would be better? State your reason.

Solution:
Second is better.
The second hash table allows 26 different hash keys, while most people may have a similar length of names, especially in Chinese, which may cause severe collisions.

Prof. Blue Tiger decides to use the second function and work with a hash table of size 13. The hash function is $h(t) = t \ \% \ 13$. This means that "Red Flamingo" would hash to 14, but ultimately fall into bucket $14 \ \% \ 13 = 1$ of our table. For this problem, you will determine where each of the given name lands after inserting a sequence of values using three different collision resolution schemes:

- linear probing
- quadratic probing
- double hashing with $h_i(t) = h(h(t) + ((13 - t) \ \% \ 10) * i)$

For each of these three collision resolution schemes, determine the resulting hash table after inserting the following (*key*, *value*) pairs in the given order:

1. ("Blue Tiger", 100)
2. ("Red Flamingo", 88)
3. ("Rainbow Horse", 80)
4. ("Honeydew Alligator", 70)
5. ("Pink Elephant", 101)
6. ("Gold Monkey", 65)
7. ("Glass Fish", 96)
8. ("Yellow Dog", 87)

Every incorrect value counts for 0.5 point.

3.2 Linear Probing (4 points)

Please use the **linear probing** collision resolution method to simulate the given insertion steps, and then show the final position of each *value* inside the related buckets below.

Solution:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value		88			100	80	70	101	96	87		65	

3.3 Quadratic Probing (4 points)

Please use the **quadratic probing** collision resolution method to simulate the given insertion steps, and then show the final position of each *value* pair inside the related buckets below.

Solution:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value		88			100	80	101	96	70		87	65	

3.4 Double Hashing (4 points)

Please use the **double hashing** collision resolution method to simulate the given insertion steps, with the double hash function $h_i(t) = h(h(t) + ((13 - t) \ \% \ 10) * i)$, and then show the final position of each *value* pair inside the related buckets below.

Solution:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value	80	88			100		101	96	87		70	65	

3.5 Possible Insertion Order (5 points)

Suppose you have a hash table of size 11 storing Blue Tiger's family members' favorite letter which shares the same t(key) and h(t). It uses open addressing with linear probing. After entering six values into the empty hash table, the state of the table is shown below.

Index	0	1	2	3	4	5	6	7	8	9	10
Key	a	l	x	z	d	w					

How many insertion orders are possible? Explain your answer clearly.

Solution:
10 possible orders
a-l-x-z-d-w, a-l-z-x-d-w, a-z-l-x-d-w, z-a-l-x-d-w
a-l-z-d-x-w, a-z-l-d-x-w, z-a-l-d-x-w
a-z-d-l-x-w, z-a-d-l-x-w
z-d-a-l-x-w

4 Binomial heap (30 points)

A binomial heap is a data structure that acts as a priority queue but also allows pairs of heaps to be merged. A binomial heap is implemented as a set of binomial trees (compare with a binary heap, which has a shape of a single binary tree), which are defined recursively as follows
1. binomial tree of order 0 is a single node.
2. binomial tree of order k k k has a root node whose children are roots of binomial trees of orders $k-1, k-2, ..., 2, 1, 0$(in this order).

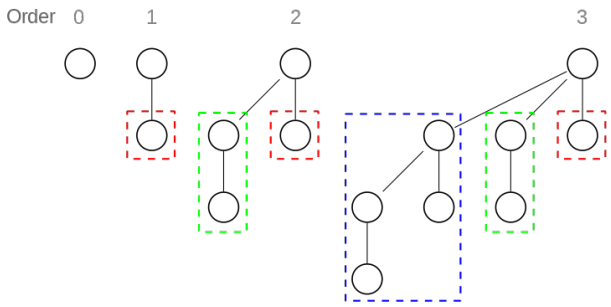


Figure 1: Binomial heap

A binomial heap is implemented as a set of binomial trees that satisfy the binomial heap properties:
1. Each binomial tree in a heap obeys the maximum-heap property: the key of a node is less than or equal to the key of its parent.
2. There can be at most one binomial tree for each order, including zero order.

4.1 Merge (8 points)

Here we provide a pseudo code to merge trees.

```
Input : p,q as two trees
Output : a merged new tree
function mergeTree(p, q)
    if p.root->key < q.root->key
        return q.addSubTree(p);
    else
        return p.addSubTree(q);
```

Here we provide a pseudo code to check whether a tree is empty.

```
Input : p as a tree
Output : whether p is empty
function isEmpty(p)
    if p.root==NULL
        return true;
    else
        return false;
```

Please complete the pseudo code for merge two heaps. Each blank may have more than one statement.

```
Input : p,q as two heaps (a list of trees sorted from order 0 to order k)
Output : heap as merged heap
function mergeHeap(p, q)
    heap=new Heap();
    pIt = p.head();
    qIt = q.head();
    heapIt = heap.head();
    while ([1])
        tree = mergeTree(*pIt,*qIt);
        if ([2])
            tree = mergeTree(tree, *heapIt);
        [3]
        pIt++;
        qIt++;
    return heap
```

Solution:

```
function mergeHeap(p, q)
    heap=new Heap();
    pIt = p.head();
    qIt = q.head();
    heapIt = heap.head();
    while (!(pIt==p.end()&&qIt==q.end()))
        tree = mergeTree(*pIt,*qIt);
        heapIt = heap.find(tree.degree)); # find the position for the tree with
             same degree with the tree merged by p and q in heap
        if (!isEmpty(heapIt))
            tree = mergeTree(tree, *heapIt);
            heap.delete(heapIt);
        heap.addTree(tree); # automatically add the tree to its degree position
             in heap
        pIt++;
        qIt++;
    return heap
```

4.2 Delete (22 points)

Please write a pseudo code to delete an element by its pointer. Hint: You can write $dequeMax$(10 points) and $increaseKey$(7 points) as helpers.

Solution:

```
Input: h as a binomial heap
Output: the node with maximum key
function dequeMax(h)
    heapIt = h.head();
    heapIt_prev = h.beforeHead(); # beforeHead points to the beginning of the
        heap
    max = heapIt.key;
    maxIt = h.head();
    maxIt_prev = h.beforeHead();
    heapIt++;
    while (!isEmpty(heapIt)) # find the position of maximum key in heap
        if (heapIt.key > max)
            max = heapIt.key;
            maxIt = heapIt;
            maxIt_prev = heapIt_prev;
        heapIt++;
        heapIt_prev++;
    maxIt_prev.next = maxIt.next; # delete the tree with maximum key
    x = maxIt.child; # delete the maximum node and make the rest a new heap
    if (!isEmpty(x))
        heap = new Heap();
        heap.head = x;
        newHeapIt = heap.head();
        newHeapIt++;
        y = x.sibling;
        x.sibling = null;
        while (!isEmpty(y))
            newHeapIt = y;
            y = y.sibling;
            newHeapIt++;
        mergeHeap(h, heap); # merge two heaps to get a new heap;
    return x;

Input: n as a node, key as the key we want to increase to
Output: nothing
function increaseKey(n, key)
    n.key = key; # set the key of the node to defined key
    x = n;
    y = n.parent;
    while (!isEmpty(y) && y.key < x.key) # switch the nodes to maintain the
        properties of a tree
        x.key = y.key;
        y.key = key;
        x = y;
        y = x.parent;

Input: h as a binomial heap, n as the node we want to delete
Output: nothing
function delete(h, n)
    increaseKey(n, val); # val is a value larger than any key in the heap
    dequeMax(h);
```