Design & Analysis of Algorithms

Midterm

Date: Thursday, Oct. 15, 2020

Name: ___Chris Grimes_____

- You have 150 minutes for this exam.
- It consists of 5 problems worth 50 points each, plus a problem 6 for extra 30 points credit.
- The extra credit will be recorded separately, so make sure you have answered all questions from first 5 problems before moving on to problem 6.
- You need to collect 200 points out of 280 to have an A.

Instructions

Work as many problems as possible. All problems have the same value, but subparts of a problem may have different values (depending on their difficulties, importance, etc). Provide a short preliminary explanation of how an algorithm works before running an algorithm or presenting a formal algorithm description, and use examples or diagrams if they are needed to make your presentation clear. Please be concise and give well-organized explanations. Long, rambling, or poorly organized explanations, which are difficult to follow, will receive less credit.

Problem 1 (50)	Problem 2 (50)	Problem 3 (50)	Problem 4 (50)	Problem 5 (50)	Extra Credit (30)	Total (200/280)

Problem 1 [Algorithm Analysis].

For each of the following algorithms, indicate their worst-case running time complexity using Big-Oh notation, and give a brief (2-3 sentences each) summary of the **worst-case** running time analysis.

(a) (10 points) Selection-sort on a sequence of size *n*.

Selection sort with n elements will have a worst-case run time of $O(n^2)$ as the sort must sort all n elements for each of the n elements present in the original sequence. Since, it does n operations per element and there are n elements the Big-Oh notation is O(n*n) or $O(n^2)$.

(b) (10 points) Merge-sort on a sequence of size *n*.

Merge sort with n elements will have a worst-case run time of O(n logn) as the sort breaks the original sequence into two sub-sequences, each about roughly half the size of the original sequence, and then it sorts the sub-sequences. The sort will recursively split all sequences until the only sequences left have zero or one elements. Once all sequences have 0 or 1 elements the sequences will be merged together by checking each element in each of the two sub-sequences and placing the element with the smallest value in the parent sequence, this continues until all elements are removed from the sub-sequences.

(c) (10 points) Bucket sort on a sequence of *n* integer keys, each in the range of [0*; N*]

Bucket sort with n elements will have a worst-case run time of O(n+N) where n is the number of elements in the original sequence and N is the range of the keys found in the original sequence. The sort will declare a second sequence of size N, where N is the range of the keys found in the original sequence, and begin reading the original sequence from beginning to end. Each element in the original sequence will be placed into the second sequence in the index whose value is equal to the elements key, in case of a collision a pointer is declared and points from the index to the second element, this is done for all subsequent collisions. Once the original sequence is empty, we read the second sequence from beginning to end placing the elements back into the original sequence in the order they are found, when an index that has pointers is found, we go through the pointers removing each element and putting it in the original sequence.

(d) (10 points) Find an element in a binary search tree that has *n* distinct keys.

Finding an element in a binary tree has a worst-case run time of O(h) where h is the height of said binary tree. The height of the binary tree is dependent upon how the

original sequence was inserted into the binary tree. If a sorted sequence is inserted the tree will have a height of n where n are the number of elements inserted into said tree. In such a case the root would only have one internal child and that child, itself, would only have one internal child. This would be true of n-1 of the inserted element, with the nth element having two leaf nodes as children.

(e) (10 points) Find an element in a red-black tree that has *n* distinct keys.

Finding an element in a red-black tree has a worst-case run time of O(log n) as the tree has the property of all nodes to the left of the root are less than or equal to said root, and all nodes to the right of the root are greater than or equal to said root. In addition to this property the red-black trees also have properties to ensure that all leaf nodes are black and the path from each leaf node to the root passes through the same amount of black nodes. With these properties we can ensure that the worst-case run time for finding an element in a red-black tree will have a Big-Oh notation of O(logn).

Problem 2 [Heap].

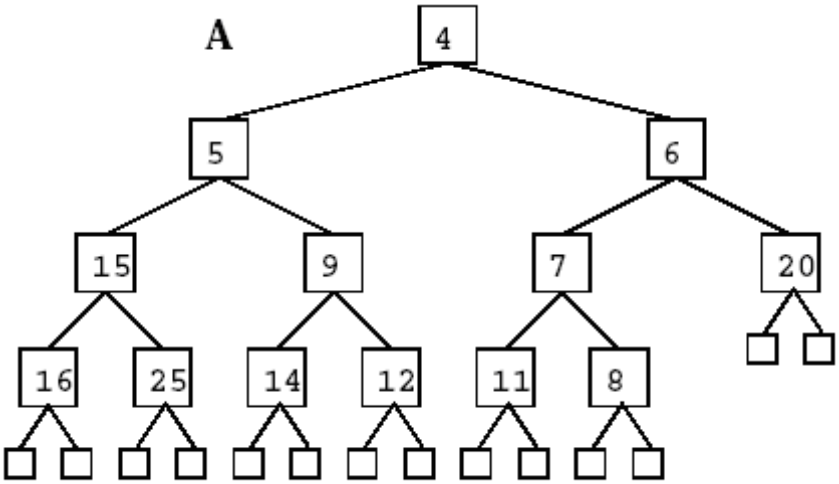
1- (5 points) Give the definition of a (min) heap.

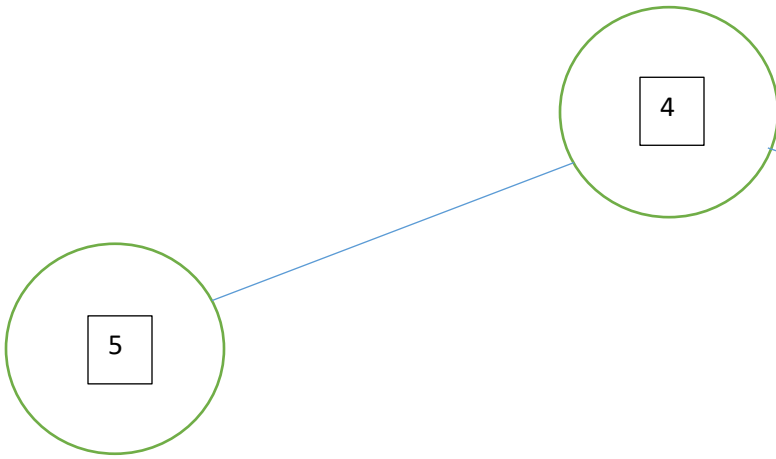
A min heap is a heap that for each node its children have values equal to or greater than it.

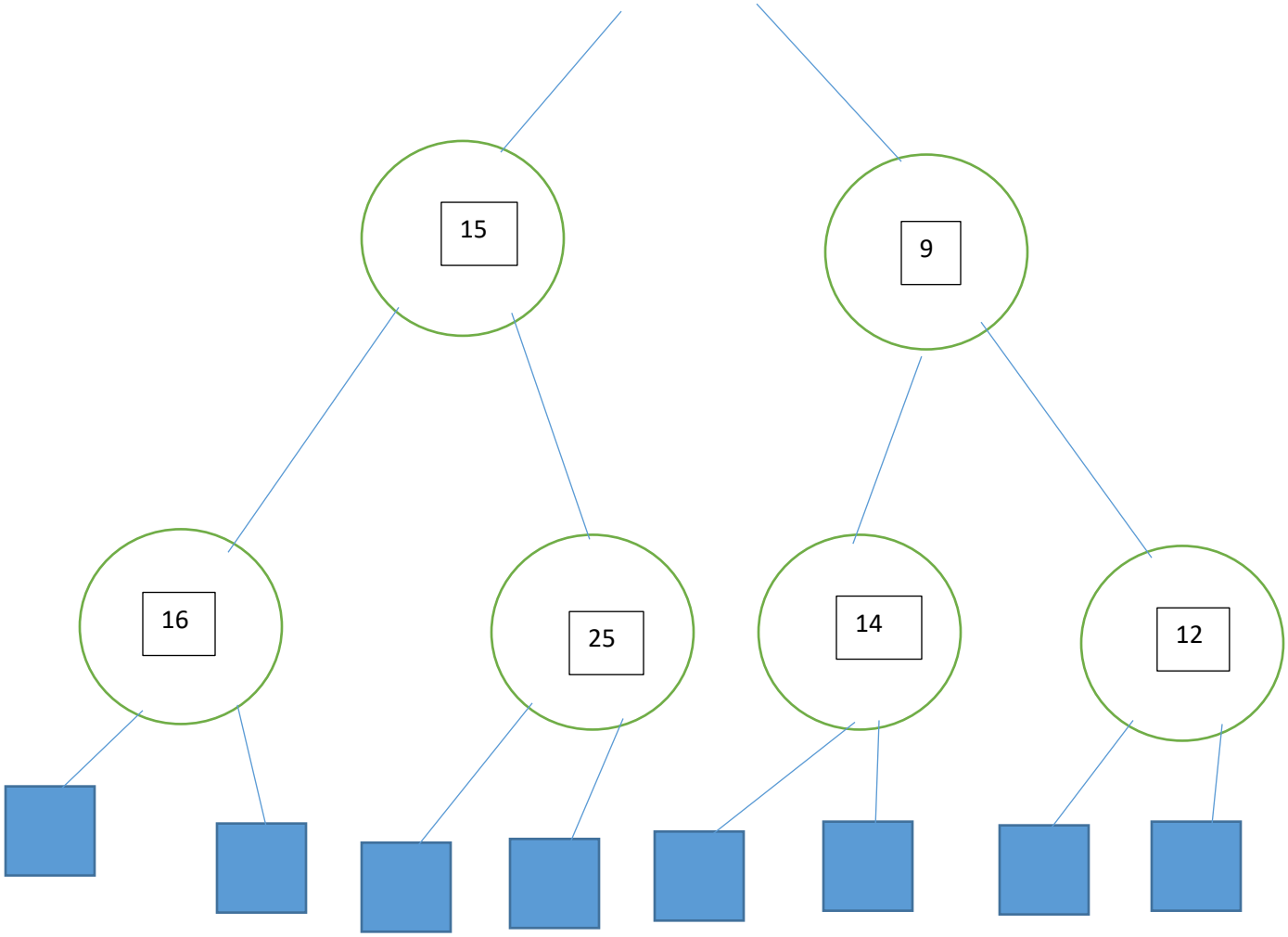
2- (10 points) What is the worst-case complexity of heap-sort? Explain.

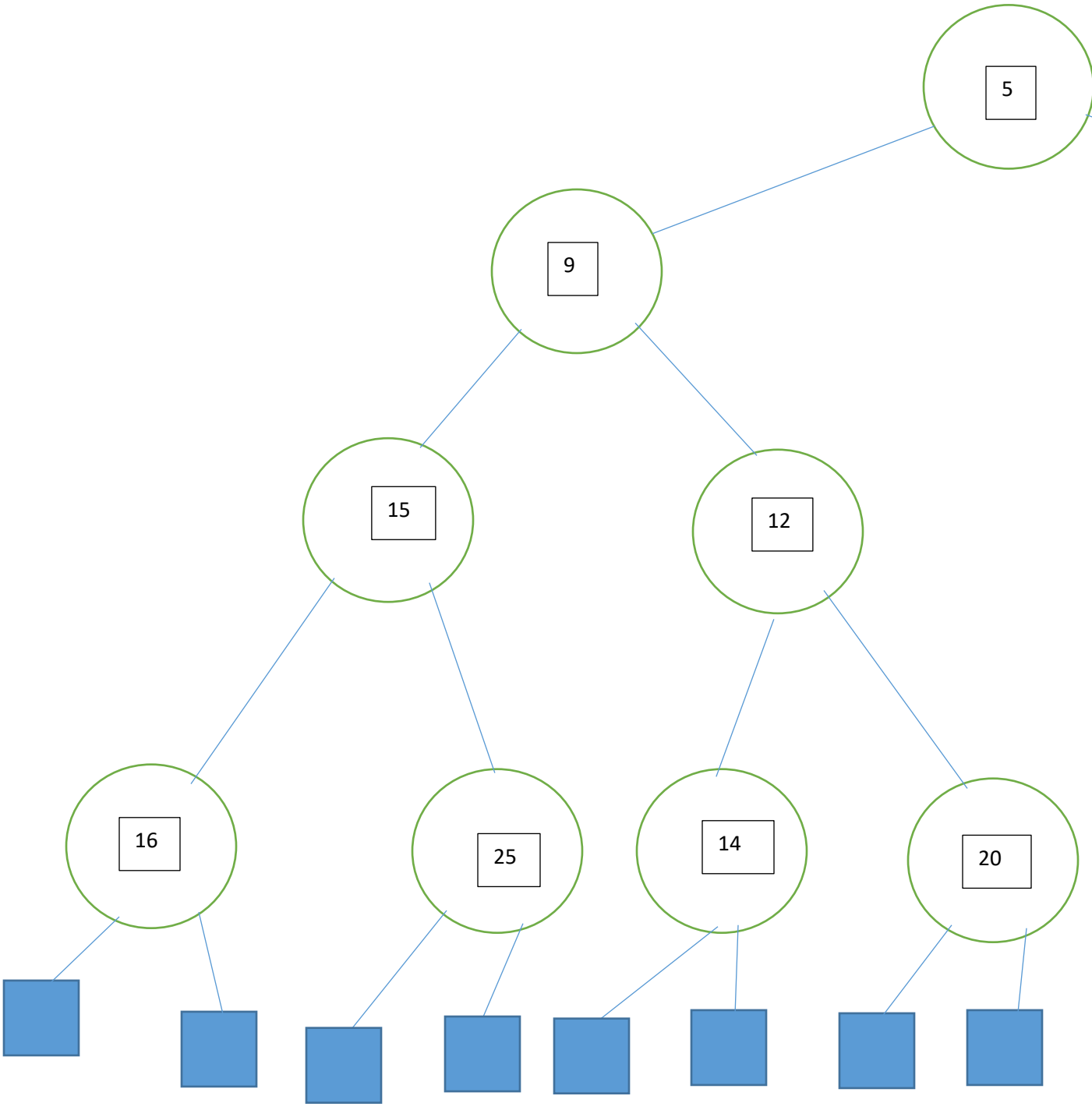
The worst-case complexity of a heap-sort is O(n logn) where n is the number of elements to be sorted and logn because a well kept heap will keep the height of its tree to logn.

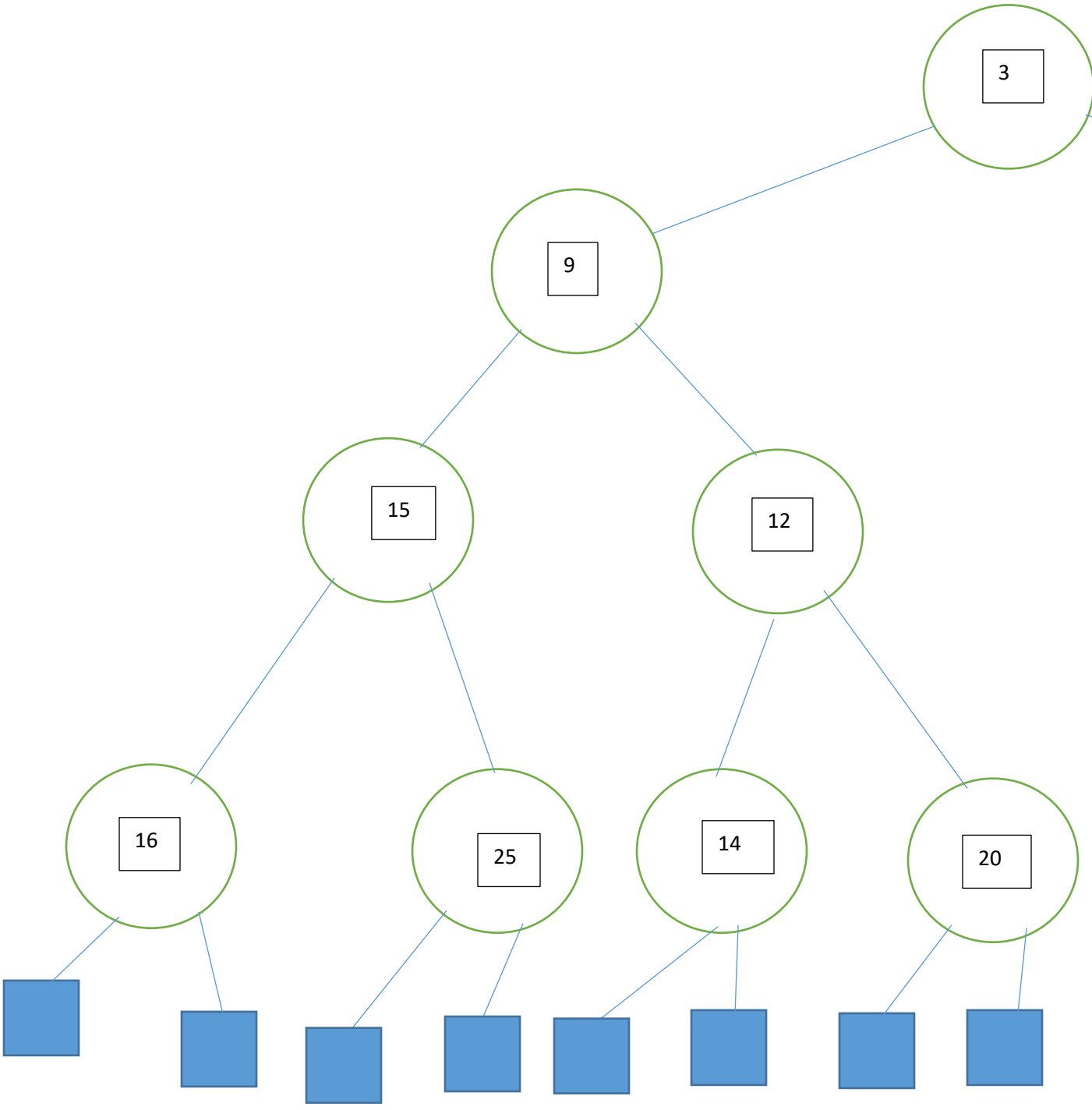
3- (30 pts) Perform sequence of ***insert(10), removeMin, insert(3)*** operations on heap ***A.*** Draw the heap after each operation.






15
9
16
25
14
12


5
9
15
12
16
25
14
20


3
9
15
12
16
25
14
20

Problem 3 [Hash Tables].

You have a hash table of size $m = 11$ and two hash functions $h1$ and $h2$. Here are some precomputed hash values:

word	ape	bat	bird	cow	dog	goat	hare	koala	mule	panda
h1	5	0	5	6	0	2	0	6	1	3
h2	2	5	5	7	1	2	2	4	3	8

1- Draw a picture of the resulting hash-table after inserting the following words in order: dog, hare, ape, mule, bat, panda, cow, koala, goat

Linear Probing

Index	0	1	2	3	4	5	6	7	8	9	10
word	Dog	Hare	Mule	Bat	Panda	Ape	Cow	Koala	Goat		
h1	0	0	1	0	3	5	6	6	2		
h2	1	2	3	5	8	2	7	4	2		

Double Hashing

Index	0	1	2	3	4	5	6	7	8	9	10
word	Dog	Mule	Hare	Panda		Ape	Cow	Koala	Goat		Bat
h1	0	1	0	3		5	6	6	2		0
h2	1	3	2	8		2	7	4	2		5

2- Identify cells that are probed when trying to find the word: bird. Do (1) and (2) using each of the following techniques:
   a. (25 points) Linear probing with *h*1 as your hash function.

      Index 5->6->7->8->9->10->0->1->2->3->4 not found

   b. (25 points) Double hashing with *h*1 as your first hash function and *h*2 as your second hash function.

      Index 5->10->4->9->3->8->2->7->1->6->0-> not found

Problem 4 [Binary Search Variants]. (50 points)

Let *A* be an array of *n* distinct integers sorted in ascending order, and let *x* be an integer which may or may not be in *A*. Describe an algorithm running in $O(\log n)$ time that determines the **number** of integers in *A* that are strictly less than *x*. Analyze the running time of your algorithm to justify that it runs in $O(\log n)$ time.

```
Algorithm findLessThan(A, x):
    Input: an array A, and an integer x
    Output: z the number of integers less than x
    z<-0
    i<- the first index in array A
    if(A[i]<x)
        ++z
    if(A[2*i] < A.size)
        if(A[2*i]=<x)
            findLessThan(A[2*i], x)
    if(A[2*i+1] <A.size)
        if(A[2*i+1]=<x)
            findLessThan(A[2*i+1], x)
    return z
```

Problem 5 [Partial Sort]. (50 points)

Suppose the entering freshman class at some university has n students. The information pertinent to each student in the class (name, identification number, employment status, etc.) can be found in some element of A, an array of records indexed from 1 to n. Assume the records are in some random order, and that we wish to rearrange the array **in-place** so that all the unemployed student records precede all the employed student records. Describe and give the **pseudocode** for an in-place algorithm running in $O(n)$-time which performs this "partial sort" on A. Assume "status" is a field in each record, with value "employed" or "unemployed".

Algorithm sortEmployment(A):

input: the array of records indexed from 1 to n A

output: the same array that was input, sorted based on employment status with unemployed first

int z<-0

```
for(int i=0; i< A.size; ++i)
        if(A[i].status[0] = ‘u’)
                swap(A[i], A[z])
                ++z
return array A
```

"Challenging" (extra credit) problem (30 pts): *A 'true' medieval story…*

Sir Arthur de Templar was paid 27 gold coins for the information he provided on the last known location of the Holly Grail. Soon, from the highly trusted sources, he has learned that one of the 27 gold coins is not fully gold (it has some admixture of iron and copper and hence its weight differs (lighter or heavier) from the weight of the other 26 true gold coins). Sir Arthur paid a visit to his closest friend drug-maker Mr. Drugless and in a few seconds they identified the fake coin. Legend says that they used just an old weighing device of Mr. Drugless (depicted in Figure above) and could identify the fake coin in only 4 weightings. Recall that the fake coin maybe lighter or heavier (that is unknown) than the true-gold one.

1. Repeat the procedure used by Sir Arthur de Templar and Mr. Drugless for identifying the fake coin among 27 gold coins in just 4 weightings. (15 pts)

2. Extend it to an algorithm for identifying a single fake coin among N given gold coins using minimum number of weightings. How many weightings do you need? (15 pts)

(A weighting is an operation of putting k coins on one plate of the device and k other coins on the other plate ($k=1,2,3,\ldots$) and checking if the device is in equilibrium, and if not, which plate is heavier, lighter.)