1. A. Key = 9, mlist = [-23, -11, -7, -2, 1, 4, 5, 7, 12, 34, 56, 75]

ITERATION #	FIRST	LAST	MIDPOINT	mlist[Midpoint]
1	mlist[0]= -23	mlist[11]= 75	mlist[5]	4
2	mlist[6]= 5	mlist[11]= 75	mlist[8]	12
3	mlist[6]= 5	mlist[7]= 7	mlist[7]	7
4	mlist[7]= 7	mlist[7]= 7	mlist[7]	7 return False

1. B. Key = 9, mlist = [-23, -11, -7, -2, 1, 4, 5, 7, 8, 9, 12, 34]

ITERATION #	FIRST	LAST	MIDPOINT	mlist[Midpoint]
1	mlist[0]= -23	Mlist[11]= 34	Mlist[5]	4
2	mlist[6]= 5	mlist[11]= 34	mlist[8]	8
3	mlist[9]= 9	mlist[11]= 34	mlist[10]	12
4	mlist[9]= 9	mlist[9]= 9	mlist[9]	9 return True

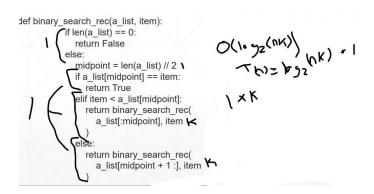
2. A. Key = 9, mlist = [-23, -11, -7, -2, 1, 4, 5, 7, 12, 34, 56, 75]

CALL#	MIDPOINT	mlist[ : Midpoint] or mlist[ Midpoint + 1 : ]
1	mlist[6]	[7, 12, 34, 56, 75]
2	mlist[2]	[7, 12]
3	mlist[1]	[7]
4	mlist[0]	[] return False

2. B. Key = 9, mlist = [-23, -11, -7, -2, 1, 4, 5, 7, 8, 9, 12, 34]

CALL#	MIDPOINT	mlist[ : Midpoint] or mlist[ Midpoint + 1 : ]
1	mlist[6]	[7, 8, 9, 12, 34]
2	mlist[2] =9	midpoint=Key return True

4. What is the upper bound for the recursive version of binary search that uses array slices? Show how you arrived at the result.



The upper bound for the recursive binary search that uses array slices is  $O(log_2(nk))$ . The function halves the list on each recursion call through the slice function. Since the function cuts the list size in half each time, it is  $log_2 n$ . But since it uses slices that are O(k) it becomes  $log_2(nk)$ .

5. Words = {moose, elephant, viper, tarantula, baboon, tiger, jaguar}

Ordinal value of:

moose =547 baboon =625 elephant =849 tiger =539 viper =550 jaguar =634

tarantula =972

a. Number of hash slots = 4

Math:

547%4=3, 849%4=1, 550%4=2, 972%4=0, 625%4=1, 539%4=3, 634%4=2

## Mod Remainder Word

0	tarantula,
1	elephant, baboon
2	viper, jaguar
3	moose, tiger

## b. Number of hash slots = 18

Math:

547%18=7, 849%18=3, 550%18=10, 972%18=0, 625%18=13, 539%18=17, 634%18=4

<u>Mod Remainder</u>	<u>Word</u>
0	tarantula
1	
2	
3	elephant
4	jaguar
5	
6	
7	moose
8	
9	
10	viper
11	
12	
13	baboon
14	
15	
16	
17	tiger

## Which one is a perfect hash - hash in part a. or part b.? Why?

The hash in part b is a perfect hash because there are no collisions. Part a is not perfect because it has several collisions.