Example Final Questions

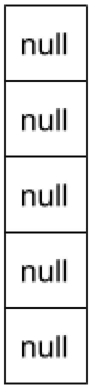
Consider the Hashtable code below.

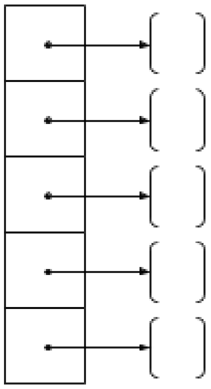
```
public class Hashtable {
      private LinkedList table;
      private int size; // current number of key value pairs
      private int tablesize; // number of array elements to allocate for table

      // This hashtable uses String keys to store integer values
      protected class Entry {
            String key;
            int value;
      }
      …
      public Hashtable(int tablesize) {
            this.tablesize = tablesize;
            size = 0;
            table = new LinkedList[tablesize];
      }
}
```

In the questions below, entries are denoted with **{ }** (e.g. {apple, 1} means the entry consisting of the key "apple" and the value 1, and lists are denoted with **( )** (e.g. ( ) is the empty list, and ({apple, 1}) is a list consisting of one entry).

1. After executing the line:
`Hashtable ht = new Hashtable(5);`
what does the table field of ht look like?

A. null

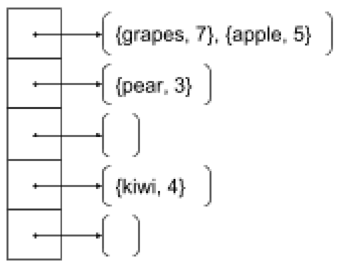
B. 

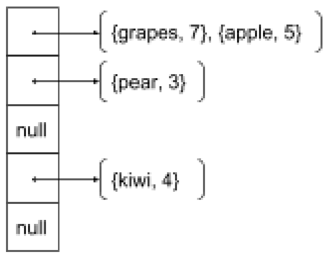
C. 

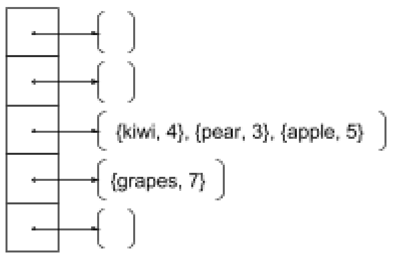
2. In this question, we're adding the following key/value pairs to the hashtable: {apple, 5}, {pear, 3}, {grapes, 7}, {kiwi, 4}. The hash value for each key is:
`hashCode(key) % tablesize`
where the hashCode(key) is given in the table below.

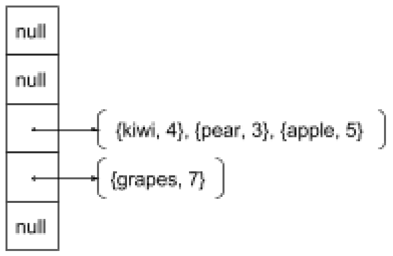
Key	**Hashcode**
apple	10
pear	11
grapes	15
kiwi	13

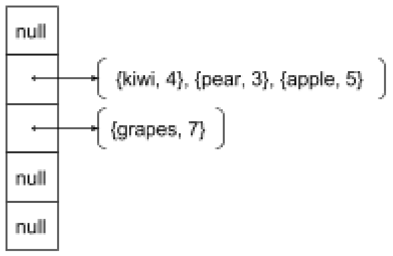
Which diagram below indicates the state of the hashtable after inserting these four elements?

A.


B.


C.


D.


E.


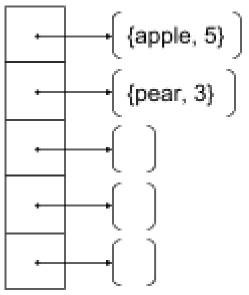
3. Is the algorithmic complexity (big-O) of delete() and get() on a hashtable the same or different?

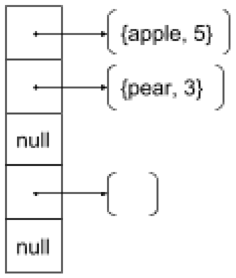
A. Same B. Different

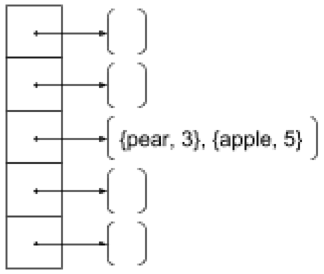
4. Assuming that in your hash tables there are *no collisions*, what is the worst-case complexity of get()?

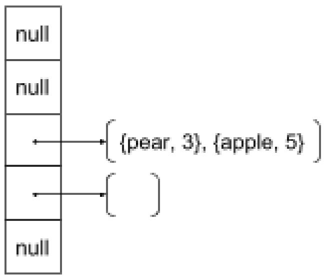
A. O(1) B. O(log n) C. O(n) D. O(n log n) E. O(n^2)

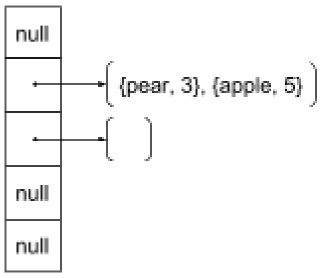
5. Given the table resulting from question 2, what is the result of deleting the keys **grapes**, and **kiwi**?

A.


B.


C.


D.


E.


6. Assuming there *are collisions*, what is the worst-case complexity of get()?

A. O(1) B. O(log n) **C.** O(n) D. O(n log n) E. $O(n^2)$

7. Assuming there *are collisions*, and that inserts into a bucket are done at the front of the linked list, what is the worst-case complexity of put()?

A. O(1) B. O(log n) **C.** O(n) D. O(n log n) E. $O(n^2)$

Consider the following sorting algorithm.

```
void swap(int *i, int *j) {
   int temp = *i;
   *i = *j;
   *j = temp;
}

void someSort(int *array, int length) {
   for(int i = 1; i < length; i++)
      for(int j = 0; j < length - i; j++)
         if (array[j] > array[j+1])
            swap(&(array[j]), &(array[j+1]));
}
```

8. What is the complexity of this sorting algorithm?

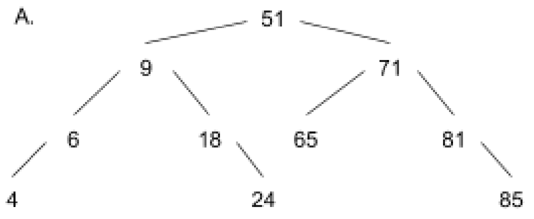
A. O(1) B. O(log n) C. O(n) D. O(n log n) **E.** O(n^2)

9. Which sorting algorithm is this?
A. Selection sort
B. Merge sort
C. Insertion sort
D. Bubble sort

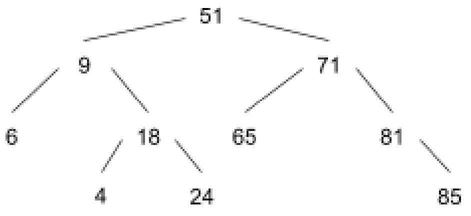
10. Is this sorting the array into ascending order (largest value will be at the highest index) or descending order (largest value will be at the smallest index)?
A. Ascending order
B. Descending order

11. What BST results from inserting the following integers into an empty BST: 51, 4, 6, 81, 18, 71, 24, 85, 9, 65

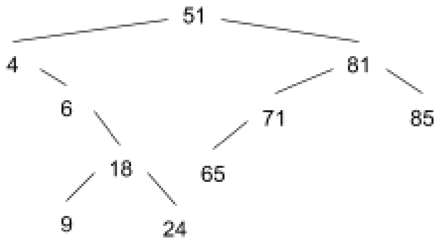
A.



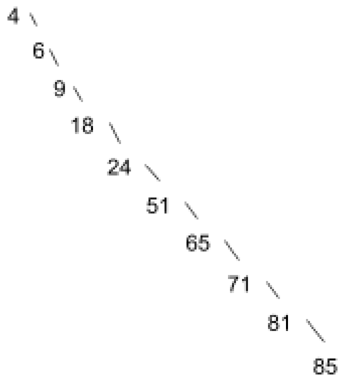
B.



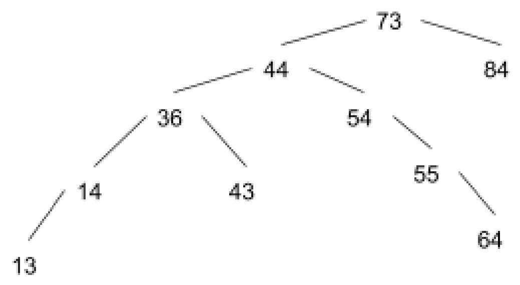
C.



D.



The next three questions use the following tree



12. What is the in-order traversal of this tree?

A. 73 44 36 14 13 43 54 55 64 84

B. 13 14 36 43 44 54 55 64 73 84

C. 13 14 43 36 64 55 54 44 84 73

13. What is the pre-order traversal of this tree?

A. 73 44 36 14 13 43 54 55 64 84

B. 13 14 36 43 44 54 55 64 73 84

C. 13 14 43 36 64 55 54 44 84 73

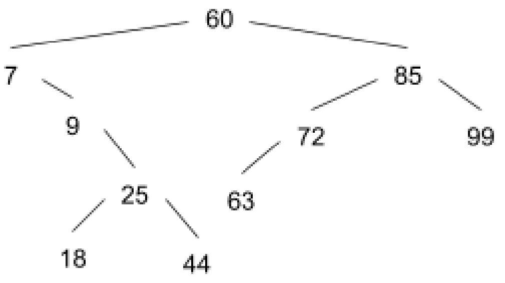
14. What is the post-order traversal of this tree?

A. 73 44 36 14 13 43 54 55 64 84

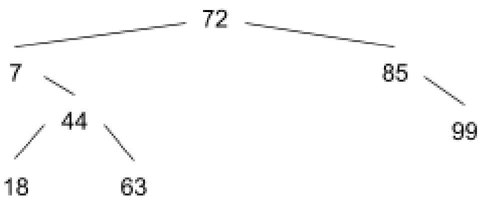
B. 13 14 36 43 44 54 55 64 73 84

C. 13 14 43 36 64 55 54 44 84 73

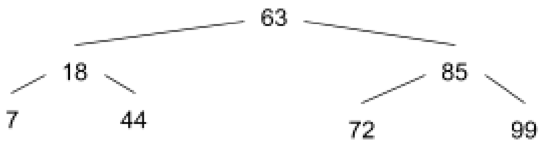
15. Given this tree, what tree results from deleting 25, 9 and 60 in that order. Like the BST assignment you did, we're using the in-order successor when necessary during deletions.



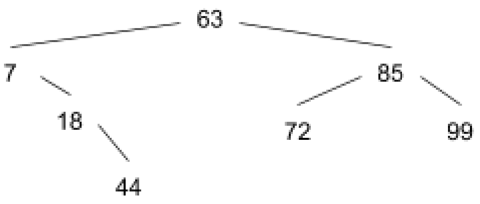
A.



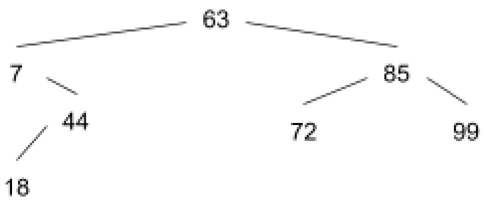
B.



C.



D.



16. Which tree above supports the most efficient lookup operation?
A. **B.** C. D.

17. Assuming a balanced tree, what is the algorithmic complexity of `contains()` on a BST?

A. O(1) **B.** O(log n) C. O(n) D. O(n log n) E. O(n^2)

18. Assuming that the tree is not necessarily balanced, what is the worst-case complexity of `contains()` on a BST?

A. O(1) B. O(log n) **C.** O(n) D. O(n log n) E. O(n^2)

19. Whether a tree is balanced or not, what is the algorithmic complexity of doing a traversal (doesn't matter what order) of a BST?

A. O(1) B. O(log n) **C.** O(n) D. O(n log n) E. O(n^2)

Given the C program below, answer the following questions. Comments in the code tell you the memory addresses for variables if appropriate.

```
#include<stdio.h>
#include<stdlib.h>

void f(int **p, int x, int y){
  *p = malloc(sizeof(int)); // *p = 50000
  **p = x*y;
  printf("**p = %d\n", **p);
}

void g(int* p, int* q){
  int *v;
  f(&v, *p, *q); // &v = 300
  printf("*v = %d\n",*v);
  *p = (*p) * 10;
  *q = (*q) + 10;
}

int main(void){
  int a=2, b=3;
  g(&a, &b); // &a = 200, &b = 204
  printf("a = %d, b = %d\n", a, b);
  return(EXIT_SUCCESS);
}
```

20. What is the value of `**p` at the end of f()?

A. 50000 B. 40800 **C.** 6 D. NULL

21. What is the value of `*v` at the end of g()?

A. 50000 B. 300 C. 40800 D. NULL **E.** 6

22. What is the value of `a` at the end of main()?

A. 20 B. 20000 C. 40800 D. NULL E. 6

23. What is the value of `b` at the end of main()?

A. 6 **B.** 13 C. 40800 D. NULL E. 214

Consider the NodeObj typedef below for a linked list.

```
typedef struct NodeObj {
     int item;
     struct NodeObj* next;
} NodeObj;

typedef NodeObj* Node;
```

24. Which code below correctly sums the integers stored in a list composed of NodeObj?

A.
```
int sumList(Node h) {
     int sum = 0;
     while (h == NULL) {
          sum  = sum + h->item;
          h = h->next;
     }
     return sum;
  }
```

B.
```
int sumList(Node h) {
     if (h != null)
          return
sumList(h->next);
     else
          return 0;
  }
```

C.
```
int sumList(Node h) {
     int sum = 0;
     while(h != NULL) {
          sum = sum + h->item;
          h = h->next;
     }
     return sum;
  }
```

D.
```
int sumList(Node h) {
     int sum = 0;
     while(h != NULL) {
          sum = sum + *(h->item);
          h = *(h->next);
     }
     return sum;
  }
```

E. B & C

Consider the following class definition in Java.

```
class BSTNode() {
     String item;
     BSTNode left;
     BSTNode right;

     BSTNode(String s) {
          item = s;
          left = NULL;
          right = NULL;
     }
}
```

25. What is the correct equivalent code in C for defining the fields (the constructor is a separate question below)?

A. `typedef struct bstnode_struct {`
`    char* item;`
`    struct bstnode_struct left;`
`    struct bstnode_struct right;`
`} BSTNode;`
	B. `typedef struct bstnode_struct`
	`    {`
	`    char* item;`
	`    struct bstnode_struct* left;`
	`    struct bstnode_struct* right;`
	`} BSTNode;`

C. `typedef struct bstnode_struct`
`{`
`    String item;`
`    BSTNode left;`
`    BSTNode right;`
`} BSTNode;`
	D. A & C	E. B & C

26. What is the correct equivalent code in C for the constructor? For all the functions below, assume `stdlib.h` and `string.h` have been included.

A.
```
BSTNode make_node(char* s) {
    // use strlen() library function to get the length of s
    int len = strlen(s);
    char newString[len + 1]; // Allocate a new character array on the stack.
    // use strcpy() library function to copy s into newString
    strcpy(newString, s);
    Node newNode; // Allocate a new node on the stack.
    newNode.item = newString; // Set item to the new string
    newNode.left = NULL; // initialize left and right nodes to NULL
    newNode.right = NULL;

    return &newNode;
}
```

B.
```
BSTNode make_node(char* s) {
     // allocate BSTNode on heap with everything initialized to 0 (NULL)
    Node* newNode = calloc(1, sizeof(Node));
    newNode->item = s; // set item equal to the string
    return newNode; // return the new node
}
```

C.
```
BSTNode make_node(char* s) {
    // use strlen() library function to get the length of s

    int len = strlen(s);
    // Allocate BSTNode on heap with everything initialized to 0 (NULL)
    Node* newNode = calloc(1, sizeof(Node));
    // Allocate string on heap with everything initialized to 0 (NULL)
    newNode->item = calloc(len + 1, sizeof(char));
    // use strcpy() library function to copy s into newNode->item
    strcpy(newNode->item, s);
    return newNode;
}
```

D.
```
BSTNode make_node(char* s) {
    // use strlen() library function to get the length of s
    int len = strlen(s);
    // Allocate BSTNode on the heap with everything initialized to 0 (NULL)
    Node* newNode = calloc(1, sizeof(Node*));
    // Allocate string on heap with everything initialized to 0 (NULL)
    newNode->item = calloc(len + 1, sizeof(char*));
    // use strcpy() library function to copy s into newNode->item
    strcpy(newNode->item, s);
    return newNode;
}
```

Assume the NodeObj definition from question 24. Consider the following code in C.

```
int f(Node n, int i, int j) {
    if (n == NULL)
        return -1;
    else if (n->item == i)
     return j;
    else
     return f(n->next, i, j+1);
}
```

27. What does this function do?
A. Increment every integer stored in the list by 1.
B. Adds a new integer to the end of the list.
C. Returns the index at which i is found in the list.
D. Counts how many nodes are in the list.

Consider the following code

```
#include <stdlib.h>
char* make_string() {
   char* s = “hello”;
   return s;
}

int main() {
  char* s = make_string();
  free(s);
  return 1;  // Normal termination
}
```

28. How does main terminate?
A. Terminates normally, returning 1.
B. Crashes with a pointer-related error.

29. Consider a linked list and an array list implementation of the List interface. Assume that both lists have sorted data stored in them. We want to use binary search to quickly search the list, where binary search will call the get() method on the list. What can we say about using binary search in this way?

A. Binary search can efficiently search both lists.
B. Binary search can efficiently search the linked list but not the array list.
C. Binary search can efficiently search the array list but not the linked list.
D. Binary search can efficiently search neither list.