



# University of Maryland College Park

## Department of Computer Science

### CMSC132 Summer 2023

### Exam #3

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g. 123456789):

#### Instructions

- Please print your answers and use a pencil.
- This exam is a closed-book, closed-notes exam with a duration of 50 minutes and 100 total points.
- **Do not remove the exam's staple.** Removing it will interfere with the scanning process (even if you staple the exam again).
- Write your directory id (e.g., terps1, not UID) at the bottom of pages with **DirectoryId**.
- Provide answers in the rectangular areas.
- Do not remove any exam pages. Even if you don't use the extra pages for scratch work, return them with the rest of the exam.
- Your code must be efficient and as short as possible.
- If you continue a problem on the extra page(s) provided, make a note on the particular problem.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must write your name and id at this point (we will not wait for you after time is up).**
- You must stop writing once time is up.

#### Grader Use Only

Part #1 (Short Answer)	20
Part #2 (Code)	80
Total	100

## **Part 1 (Short Answers) – 20 pts – 2 pts each**

1. A typical \_\_\_\_\_ does two things: generates a hash code and compresses it.
2. Linear probing is an example of \_\_\_\_\_ addressing.
3. Which Java Collection Framework Set implementation is best to use if you want your `String` data to be iterated over in alphabetical order? \_\_\_\_\_
4. Notation used to say that  $f(n)$  is in both  $O(g(n))$  and  $\Omega(g(n))$ ? \_\_\_\_\_
5. In hashing, the ratio of *(# of entries in the hash table) / (size of the hash table)* is known as the \_\_\_\_\_
6. Given the pseudocode below:

```
System.out.println("hi");  
for (int i = 0; i < n; i++){  
    System.out.println("hi");  
    for (int j = 0; j < n/2; j++){  
        System.out.println("hi");  
        for (int k = 0; k < 1000; k++) {  
            System.out.println("hi");  
        }  
    }  
}
```

Assume input size  $n$ . Write  $T(n)$ , the sum of the number of times the `println` method executes. Provide the answer with an expression in terms of  $n$ . What is the overall best upper bound of the code above (in terms of Big O)?

$T(n) =$  \_\_\_\_\_

$T(n)$  is in  $O(\quad)$

7. Given the pseudocode below:

```
for (int i = 0; i < n; i++){  
    System.out.println("hi");  
}  
for (int j = 1; j < n; j*=2){  
    System.out.println("hi");  
}
```

Assume input size  $n$  (for simplicity  $n$  is a power of 2). Write  $T(n)$ , the sum of the number of times the `println` method executes. Provide the answer with an expression in terms of  $n$ . What is the overall best upper bound of the code above (in terms of Big O)?

$T(n) =$  \_\_\_\_\_

$T(n)$  is in  $O(\quad)$

Based on the diagram convention used in the lecture slides, assume the following hash table using linear probing. Use it to answer questions 8 to 10.

<i>NeverUsed</i>	<i>NeverUsed</i>	<i>NeverUsed</i>	<i>Kiwi</i>	<i>Removed</i>	<i>Banana</i>
<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>

8. If **Apple** hashes to index 3, this would be an example of \_\_\_\_\_
9. If **Apple** hashes to index 3, you would place it in index \_\_\_\_\_
10. Assume **Apple** (hashes to index 3) was never added to the table and you are trying to find it. Given the state of the table above, after visiting which index can you stop the search as you know it is not in the table? \_\_\_\_\_

## **Part 2 (Code) – 80 pts**

Assume the give code below (all in the same package and all necessary imports):

```
public class Driver {
    public static void main(String[] args) {
        WordList newList = new WordList();

        newList.add("if").add("java").add("else").add("for").add("if");
        newList.add("java").add("switch").add("java").add("java").add("case");

        System.out.println(newList);
        newList.nodeAt(0); //first is if
        newList.nodeAt(6); //7th word is switch
        newList.nodeAt(10); //out of bounds

        newList.showSubLists("java");
        System.out.println(newList.makeNewList());

        newList.showSubLists("else");
        System.out.println(newList.makeNewList());

        newList.showSubLists("cmsc");
        System.out.println(newList.makeNewList());
    }
}
```

### **Driver Output**

```
" if java else for if java switch java java case "
Node [word=if]
Node [word=switch]
null
Here are the subLists starting with java:
    " java else for if "
    " java switch "
    " java "
    " java case "
" else for if switch case "
Here are the subLists starting with else:
    " else for if java switch java java case "
" for if java switch java java case "
Here are the subLists starting with cmsc:
    No subList
null
```

**Directory ID:**

```

public class WordList {

    private class Node {
        private String word;  //assume it will not be null
        private Node next;

        private Node(String word) {
            this.word = word;
            next = null;
        }

        @Override
        public String toString() {
            return "Node [word=" + word + "]";
        }
    }

    private Node head, tail;
    private int size;
    private ArrayList <WordList> subLists;

    public WordList() {
        head = tail= null;
        size = 0;
        subLists = null;
    }

    /* Adding word (assume not null) at the end of the list */
    public WordList add(String word) {
        Node newNode = new Node(word);

        if(head == null) {
            head = tail = newNode;
        }
        else {
            tail.next = newNode;
            tail = newNode;
        }
        size++;
        return this;
    }

    public String toString() {
        String result = "\" ";
        Node curr = head;

        while (curr != null) {
            result += curr.word + " ";

            curr = curr.next;
        }

        return result + "\"";
    }

    //returns the reference of the node at index – start at 0
    private Node nodeAtHelper(int index) {
        // YOU WRITE THIS METHOD
    }

    public void nodeAt(int index){
        System.out.println(nodeAtHelper(index));
    }

    private void makeSubLists(String target) {
        // YOU WRITE THIS METHOD
    }
}

```

```

public void showSubLists(String target){

    this.makeSubLists(target); //calls private helper you wrote

    System.out.println("Here are the subLists starting with "+ target+ ":");

    if(subLists.isEmpty())
        System.out.println("\t No subList");

    for(WordList wl: subLists) {
        System.out.println("\t"+ wl);
    }

}

public WordList makeNewList()
{
    if(subLists==null||subLists.isEmpty())
        return null;
    else {
        WordList newList = new WordList();

        //arguments are list to be made, index of subLists, index of node in current sublist
        return makeNewListAux(newList, 0,1);
    }
}

private WordList makeNewListAux(WordList newList, int indexInList, int indexOfNode){
    // YOU WRITE THIS METHOD
}

}

```

1. Write the code for `nodeAtHelper` that will return the node at the passed in “index”. The head is at “index” 0, the second node is at “index” 1, and so forth. If the index is out of bounds, return `null`. Otherwise, return the reference to the node. When you develop your code, you can use all fields of the `WordList` class necessary, but do not call any methods. Your code should not be recursive.

```

private Node nodeAtHelper(int index) {

```

2. Write the code for `makeSubLists` to assign values to the elements of the `subLists` field. First make an empty `ArrayList` to assign to `subLists`. Traverse the current `WordList` and every time the `target` is found make a new `WordList` object with `target` as the head and assign to the next available element of `subLists`. Keep adding nodes to the `WordList` object you just created by calling the `WordList` `add` method until a new instance of `target` is found, at which point you will make a new `WordList` object as described to be assigned to the next element of `subLists`. The nodes in your sublists must be independent of the nodes in the current object (i.e. call the `add` method of `WordList` to make new nodes). When you develop your code, you can use all fields and methods of the `WordList` class as necessary and the following Java library methods: `ArrayList` default constructor, `get`, `size`, and `add`. You can also use the `equals` method of the `String` class. Based on your code, you might not need all allowed library methods. Your code should not be recursive.

```
private void makeSubLists(String target) {
```

**//More room if you need it**

3. Write the code for `makeNewListAux` which will return a new `WordList` object made up of non-head nodes in the `subLists` field. The nodes in the return `WordList` must be independent of the nodes of the `WorldList` that are part of `subLists` field. When you develop your code, you can use all fields and methods of the `WordList` class as necessary and the following Java library `ArrayList` methods: `get` and `size`. **Your code MUST be recursive.** Notice that the code for the `public makeNewList` is fully written for you and will call the method you are writing.

```
private WordList makeNewListAux(WordList newList, int indexInList, int indexOfNode){
```