

Name: \_\_\_\_\_ RUID: \_\_\_\_\_

1. Given

```
class Student;
class Class;
std::unordered_map<Student, std::set<Class> > studentClass;
```

There are  $S$  students and an average of  $C$  classes per student.

- What is the big-O of printing all students alphabetically and for each student, listing each of their classes alphabetically? Explain your answer.

– Answer

\* ChatGpt

1. The big-O of printing all students alphabetically and for each student, listing each of their classes alphabetically is  $O(S \log S + SC \log C)$ . *This is because we need to iterate over all  $S$  students and for each student, we need to print their name and iterate over their  $C$  classes and print them alphabetically. Sorting the list of students alphabetically takes  $O(S \log S)$  time, and sorting the list of classes for each student takes  $O(C \log C)$  time. Therefore, the overall time complexity is  $O(S \log S + SC \log C)$ .*

\* SOE

- Assuming a student potentially can take as many number as whole  $C$  instead of a few or constant number of classes
- $O(S \log S + S \cdot C \log C)$

- What is the big-O of determining who's taking ECE512? Explain your answer.

– Answer

\* ChatGpt

2. The big-O of determining who's taking ECE512 is  $O(S + C \log C)$ . This is because we need to iterate over all  $S$  students and check whether they are taking the class ECE512. For each student, we need to look up their set of classes in the `unordered_map`, which takes  $O(1)$  time on average, and check whether ECE512 is in the set. Checking whether ECE512 is in the set takes  $O(\log C)$  time because the set is sorted alphabetically. Therefore, the overall time complexity is  $O(S + C \log C)$ .

- \* SOE
    - Assuming inner set is not sorted, same as question 1
    - Time complexity is  $O(S \cdot C)$
- What is the big-O of determining if Joe Smith is taking ECE512? Explain your answer.
  - Answer
    - \* ChatGpt
      3. The big-O of determining if Joe Smith is taking ECE512 is  $O(\log C)$ , where  $C$  is the average number of classes per student. This is because we need to look up Joe Smith's set of classes in the `unordered_map`, which takes  $O(1)$  time on average, and check whether ECE512 is in the set. Checking whether ECE512 is in the set takes  $O(\log C)$  time because the set is sorted alphabetically. Therefore, the overall time complexity is  $O(\log C)$ .
- 2. Given a randomly ordered array of  $N$  integers, what is the big-O of the most efficient algorithm for determining if any single number in the array makes up more than 50% of the array? Explain your answer.

3. Given a binary search tree, what is the algorithm to print it out in reverse order? What is the big-O of this algorithm? Explain your answer. What is the big-O of the algorithm if the tree is not bushy (e.g. if the values were inserted into the tree in alphabetical order) Explain your answer.

4. If you were to perform a pre-order traversal of this binary tree and insert each node, in the order it is processed, into a closed hash table of size 7, show the resulting hash table. Explain your answer.

```
digraph g {
node [shape = record,height=.1];
node1[label = "<f0>|<f1>10|<f2>"];
node2[label = "<f0>|<f1>7|<f2>"];
node3[label = "<f0>|<f1>12|<f2>"];
node4[label = "<f0>|<f1>3|<f2>"];
node5[label = "<f0>|<f1>11|<f2>"];
node6[label = "<f0>|<f1>100|<f2>"];
node7[label = "<f0>|<f1>3|<f2>"];

"node1":f0 -> "node2":f1;
```

```
"node1":f2 -> "node3":f1;
"node2":f0 -> "node4":f1;
"node4":f0 -> "node7":f1;
"node3":f0 -> "node5":f1;
"node3":f2 -> "node6":f1;

}
```

5. What is the big-O of the following algorithm? Explain your answer.

```
void func(int arr[], int n){
    set<int> s;
    for (int i=0; i<n; i++){
        for (int j=0; j<i; j++)
            s.insert(arr[i] * arr[j]);
    }
}
```

6. Given the following unsorted array of integers, show it after it has been heapified using the efficient heap-sort algorithm that performs heapification in-place. Explain your answer.

1	2	3	4	5	6	7	8	9	10
1	9	100	3	77	5	29	30	88	33

7. You need to sort an array of integers which is already almost entirely sorted. Should you use the insertion sort, the merge sort, or the heapsort? Explain your answer.

8. You need to build a table to quickly look up student records by the students' last names; the set of students will grow and shrink each year as new students are enrolled and others graduate. Should you use a sorted array, a closed hash table, or a binary search tree? Explain your answer.

9. Write a function called `swapFirstTwo` which swaps the first two items in a linked list (if there are at least two items).

```
struct NODE{  
    int val;  
    NODE *next;  
};
```

```
struct NODE{  
    int val;  
    NODE *next;  
};
```

10. What does this program print? Explain your answer.

```
class MagicPower{  
public:  
    MagicPower() { cout << "MP\n"; }  
    ~ MagicPower() { cout << "~MP\n"; }  
};
```

```
class Wand{  
public:
```

```

        Wand() { cout << "Wand\n"; }
        ~Wand() { cout << "~Wand\n"; }
private:
        MagicPower myPower;
};

class Wizard{
public:
        Wizard() { cout << "Wizard\n"; }
        ~Wizard() { cout << "~Wizard\n"; }
private:
        Wand myWand;
        Wand *secondWand;
};

int main(){
        Wizard *tom;
        tom = new Wizard;
        Wizard david[2];
}

```

11. Show the copy constructor for the stomach class.

```

class Stomach{
public:
        Stomach(string items[], int n){
                count = n;

                if (n <= 3){ // if <= 3 items then use array
                        for (int i=0;i<n;i++)
                                contents[i] = items[i];
                }
        }
}

```

```

        }else{           // otherwise use new to alloc array
            moreThanThree = new string[n];
            for (int i=0;i<n;i++)
                moreThanThree[i] = items[i];
        }
    }

    ~Stomach(){
        if (count > 3)
            delete [] moreThanThree;
    }

private:
    string contents[3];
    string *moreThanThree;
    int count;
};

```

12. Define a constructor for the NightClub class so it works as specified below:



Sam's joke should be: "A man walks into a bar... ouch!" and he should tell it count times.

David's joke should be: "A fish swims into a wall... Damn!" and he should tell it twice as many times as Sam.

```
class BadComedian{
public:
    BadComedian(const string &joke, int times){
        myJoke = joke;
        numTimes = times;
    }
    void tellJoke() const {
        for (int i=0; i<numTimes; i++)
            cout << myJoke << endl;
    }
private:
    string myJoke;
    int numTimes;
};

class NightClub{
public:
    NightClub(int count) { }

    void doShow() {
        sam.tellJoke();
        david.tellJoke();
    }
private:
    BadComedian sam;
    BadComedian david;
};
```

13. Write a recursive function called `freeList` that accepts a pointer to a doubly linked list node. The function must print all of the items in order and also delete all nodes except for the first in the linked list.

```
struct NODE{
    int val;
    NODE *next, *prev;
};

void main(){
    NODE *head; // ptr points to the first node (e.g. head) of the linked list
    head = createSomeLinkedList();
    freeList(head);
}
```

14. Create a set of C++ classes using inheritance:

All Sea Animals can make noise using a `makeNoise` function that takes no arguments and returns nothing

- You must specify a Sea Animal's weight when its born

- You can get a Sea Animal's weight by calling its `weight()` function which returns its weight.
- All Sea Animals can `eat(...)` another Sea Animal (passed in by pointer) and gain the weight of the consumed animal (the eaten animal should be deleted)
- Otters have a `makeNoise` method that prints "Bark".
- Squid have a `makeNoise` method that prints "Squeak".
- Giant Squid, a species of Squid, burps after it eats another animal by printing "Burp" to the screen.
- Giant Squid always weight 1000 pounds.

15. Write a recursive function called `addOnes` that finds all nodes containing a value of 0 in a linked list and adds a new node after each with a value of 1.

16. Write a function called `Balanced` that determines whether or not a parenthesized expression is properly balanced. Your function should process strings with the following types of delimiters: `( ) { } [ and ]`. Your function should accept two arguments: a string argument and a reference to an integer. The first, string argument contains the input expression that should be evaluated. The second, integer reference should have its value set to the maximum “nesting” depth of the expression. The function should return a bool: true if the expression is properly parenthesized, and false otherwise.

Here are examples of valid, balanced strings:

<code>bletch</code>	<code>MaxDepth = 0</code>
<code>{}</code>	<code>MaxDepth = 1</code>
<code>[( ) ( )]</code>	<code>MaxDepth = 2</code>
<code>{goober[{face}]}</code>	<code>MaxDepth = 3</code>
<code>(a(b((c)))d)[(ef{g})]</code>	<code>MaxDepth = 4</code>

Here are examples of invalid strings:

```
Snitch[
[fe[fi[fo}fum]ack)
((start())()
```

17. A “binary tree” is a data structure that employs a special type of linked list node. In a binary tree, each linked list node has two next pointers. Here is an example binary tree node:

```
struct binodes
{
    int  datavalue;
    btnode *left;
    btnode *right;
};
```

Write a member function called findValue that accepts a pointer to the head of the tree and an int parameter and returns an int. The function must search through each node of the binary tree and count the number of times the value was found in the tree, then return this value.

- First implement your function using a queue.
- Then implement it with recursion.

```
digraph g {
    node [shape = record,height=.1];
    nodeH[label = "<f0> Head"];
    node0[label = "<f0> |<f1> 1|<f2> "];
    node1[label = "<f0> |<f1> 7|<f2> "];
    node2[label = "<f0> |<f1> 2|<f2> "];
    node3[label = "<f0>nullptr |<f1> 10|<f2> nullptr "];
    node4[label = "<f0> |<f1> 20|<f2> "];
    node5[label = "<f0>nullptr |<f1> 20|<f2>nullptr "];
    node6[label = "<f0>nullptr |<f1> 50|<f2>nullptr "];
    node7[label = "<f0>nullptr |<f1> 77|<f2> nullptr "];
    node8[label = "<f0>nullptr |<f1> 2|<f2>nullptr "];
    "nodeH":f1 -> "node0":f1;
    "node0":f2 -> "node4":f1;
    "node0":f0 -> "node1":f1;
    "node1":f0 -> "node2":f1;
    "node1":f2 -> "node3":f1;
    "node2":f2 -> "node8":f1;
    "node2":f0 -> "node7":f1;
    "node4":f2 -> "node6":f1;
    "node4":f0 -> "node5":f1;
}
```