

This is publicly accessible
on

<https://www.cs.umd.edu/~gihan/resources/cmsc132/>

CMSC132: Week 02, Lab 1

2025-Feb-03, Gihan

Outline

- Project 01
 - Submissions
 - Test types – Public, release, private
 - Coding style
- Inheritance (Recap from Week 01, Lab 2)
- ArrayList
 - Initialization, add, get, set, remove, clear, iterations
- 2D Arrays
 - Initialization, ragged, iterations

Additional resources [Brian](#), [Angelyn](#)

CMSC132: Week 02, Lab 2

2025-Feb-05, Gihan

Outline

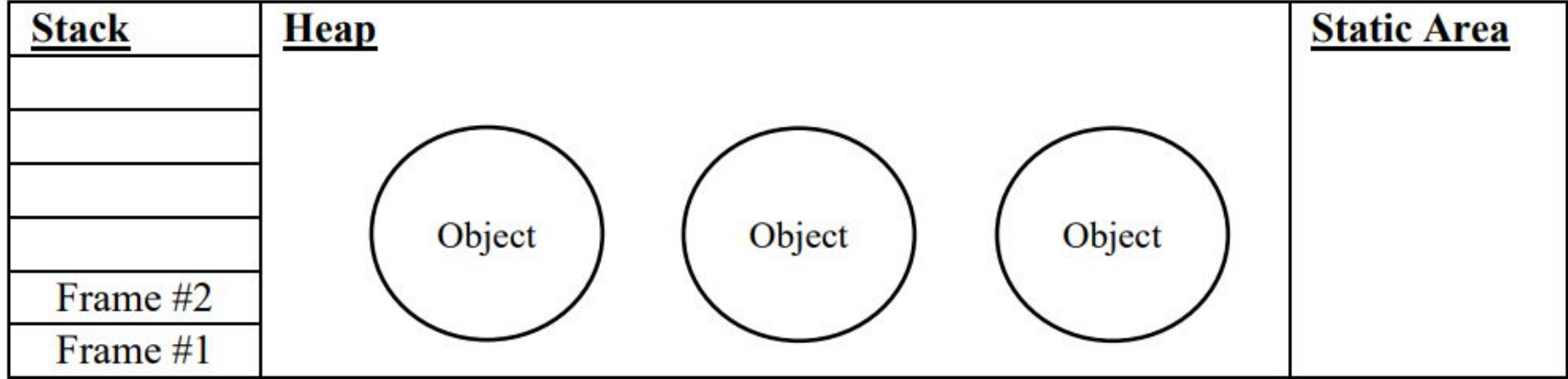
- Check – are you fine with submitting the Project 1?
- Doubts check – lecture material.
- Debugger
 - Breakpoints
 - Views
 - Step into, step over, step return
 - Colors for public, private, static, instance variables.
- Debugging instead of print()
 - Checking the value of variables, arrays, and ArrayLists.

CMSC132: Week 03, Lab 1

2025-Feb-10, Gihan

Outline

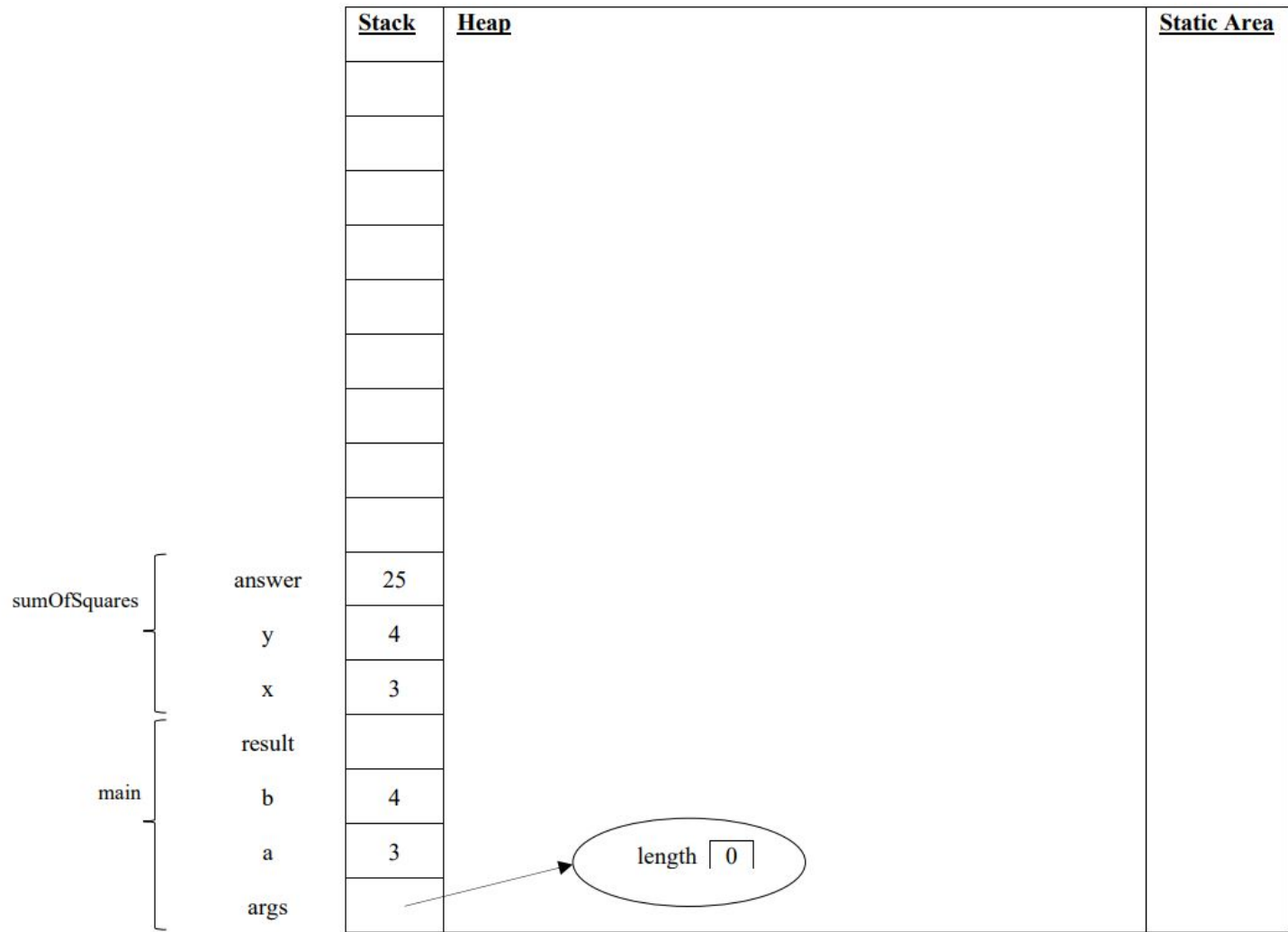
- How is Proj 1 going?
- Java assert
- JUnit
 - assertTrue, assertEquals, assertFalse
 - E.g. AuxMath, Zip archive
- Memory maps
 - Stack, heap, static, code
-



Example #1 (Static Methods)

Draw a memory map for the following program at the point in

```
public class Driver {  
    public static int sumOfSquares(int x, int y) {  
        int answer;  
  
        answer = x * x + y * y;  
  
        /* HERE */  
        return answer;  
    }  
  
    public static void main(String[] args) {  
        int a = 3, b = 4, result;  
  
        result = sumOfSquares(a, b);  
        System.out.println("Answer: " + result);  
    }  
}
```



Example #2 (Methods/Objects)

Draw a memory map for the following program at the point in the program execution indicated by the comment **/*HERE*/**.

```
public class Person {
    public static double MINIMUM_SALARY = 1000.00;
    private String name;
    private double salary;
    private StringBuffer calls;

    public Person(String name, double salary) {
        this.name = name;
        this.salary = salary;
        calls = new StringBuffer("Calls: ");
    }

    public Person(String name) {
        this(name, MINIMUM_SALARY);
    }

    public Person increaseSalary(double delta) {
        salary += delta;

        /* Returning reference to current object */
        return this;
    }

    public void addCall(String newCall) {
        calls.append(newCall);
    }

    public String toString() {
        return name + ", $" + salary + ", " + calls;
    }
}
```

```
public class Driver {

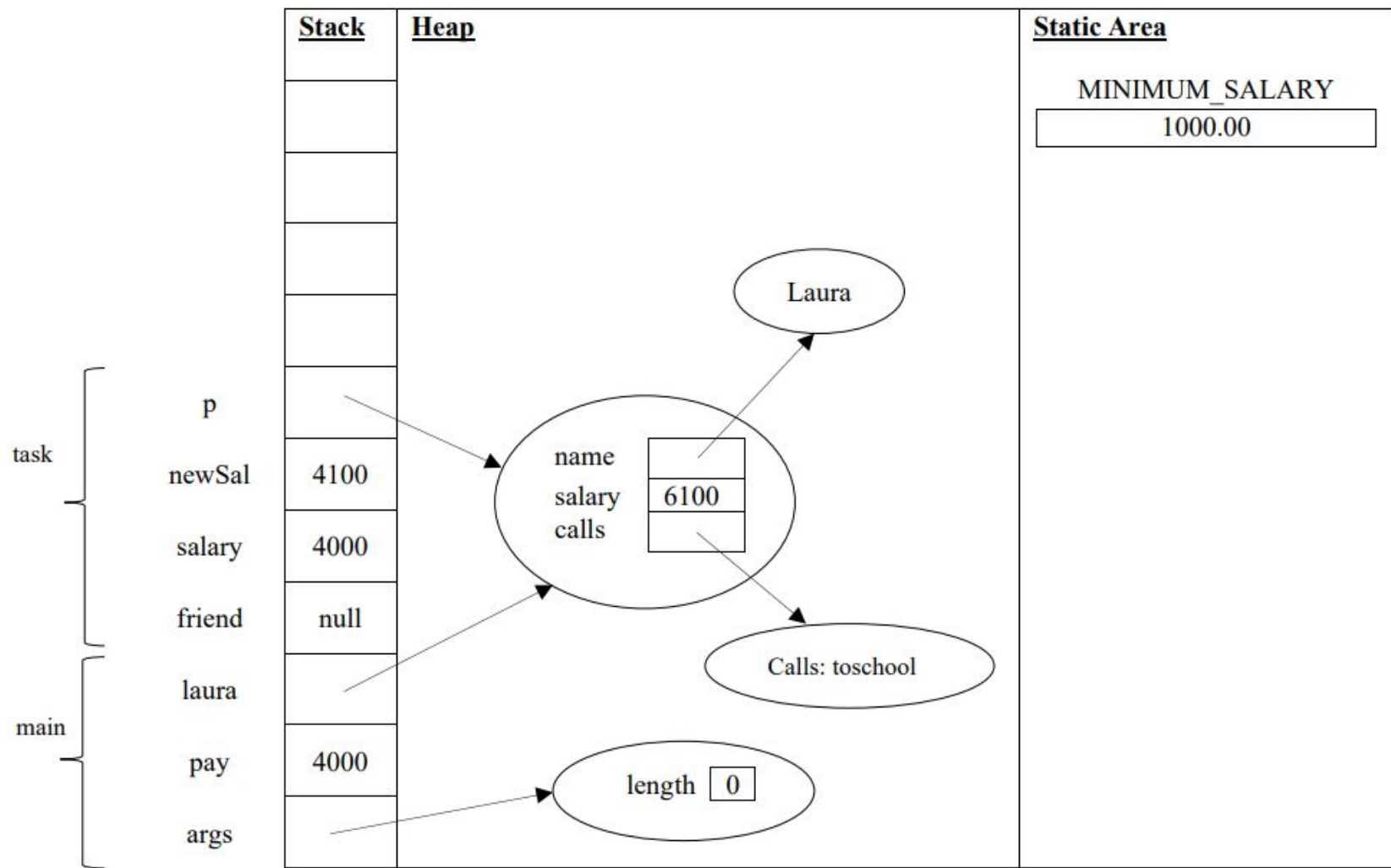
    public static void task(Person friend, double salary) {
        double newSal = salary + 100;
        Person p = friend;

        p.increaseSalary(newSal);
        friend.addCall("toschool");
        friend = null;

        /* HERE */
    }

    public static void main(String[] args) {
        double pay = 4000;

        Person laura = new Person("Laura", 2000);
        task(laura, pay);
        System.out.println(laura);
    }
}
```



Sample Memory Maps

Please use the format described by the examples below when asked to draw memory maps / diagrams. Each example below covers typical diagrams we will ask you to draw. Regarding the diagrams:

1. When asking to draw a map, we need to set a stop point, so you can draw the contents of the stack, heap, and static area up to that point in the execution (otherwise the stack would be empty as the program would have finished execution). We will represent that stop point with the comment `/* HERE */`.
2. When drawing an object, we draw the instance variables associated with the object. For arrays, we draw the length property and a row of entries representing the array entries.
3. For non-static methods, the “this” current object reference will be drawn as the first entry in the stack (it can be seen as an implicit parameter).
4. Although the name “static method” may imply that static methods live in the static area, that is not the case. The code for both static and non-static methods resides in the code area. Both static and non-static methods use the stack for execution. The only difference between static and non-static methods, is that a static method does not require an object in order to be executed.
5. Any entry in the stack that has not been assigned a value will be left blank.
6. You should draw variables in the stack as you encounter them during code execution. This will allow you to verify your work easily.
7. You will see that **args** is always the first entry in the **main** method’s frame. The **args** parameter represents command line arguments. We will not provide any command line arguments in our examples, so we will always draw **args** as a reference to an array of length 0. You will get points for drawing this **args** entry.
8. For simplicity, loop variables defined inside of a loop (e.g., `for (int i = 0 ...)`) will not be drawn unless the `/* HERE */` marker is within the scope of the variable.
10. We will use the following symbol to represent a stack frame.



CMSC132: Week 04, Lab 1

17-Feb-2025, Gihan

- Project 2 questions
- Snow day content
 - Comparator, Comparable
 - Lists
 - Vector
 - ArrayList
 - Stack

CMSC132: Week 05, Lab 1

24-Feb-2025, Gihan

Outline

- Proj 1, Proj 2
- How is project 3 going?
- Tying some loose ends from last week.
 - Difference between **final** and immutable.
 - Exceptions (Checked and unchecked)
 - Autoboxing and unboxing
- Enhanced switch
- Enums
- Annotations
- Varargs

Checked and unchecked Exceptions

Examples of Checked Exceptions:

- `IOException`
- `SQLException`
- `FileNotFoundException`
- `InterruptedException`

Examples of Unchecked Exceptions:

- `NullPointerException`
- `ArrayIndexOutOfBoundsException`
- `ArithmeticException`
- `IllegalArgumentException`

Autoboxing and Unboxing

```
public class AutoboxingExample {  
    public static void main(String[] args) {  
        int primitiveInt = 10;  
        Integer wrapperInt = primitiveInt; // Autoboxing  
        System.out.println(wrapperInt); // Output: 10  
    }  
}
```

```
public class UnboxingExample {  
    public static void main(String[] args) {  
        Integer wrapperInt = 20;  
        int primitiveInt = wrapperInt; // Unboxing  
        System.out.println(primitiveInt); // Output: 20  
    }  
}
```

Autoboxing and Unboxing continued

- Autoboxing in ArrayList
- Unboxing in primitive operations

Considerations

	Autoboxing	Unboxing
Memory usage?	Increase	Decreases
Equality checks?	Method calls	==
Null initialization?	Null	0
	Slow	Fast

Other presentation and code

- Enhanced switch
- Enums
- Annotations
- Varargs

CMSC132: Week 05, Lab 2

26-Feb-2025, Gihan

CMSC132: Week 07, Lab 1

10-Mar-2025, Gihan

Outline

- Project 3
- Search
 - Linear
 - Binary
- Sort
 - Bubble
 - Selection

CMSC132: Week 07, Lab 2

12-Mar-2025, Gihan

Outline

- Recap : LinkedList
- Java Collections Framework
 - LinkedList<E>
 - Deque<E>
 - Iterator<E>
 - ListIterator<E>

CMSC132: Week 08, Lab 1

24-Mar-2025, Gihan

Outline

- How is Project 5 going?
- Week 7>>> LListRecursion >>>
 - **delete()**
 - **findMax()**
- Insertion sort
- Cloning
 - Copy constructor?
 - Cloneable interface
 - clone()
 - Deep/shallow?

CMSC132: Week 08, Lab 2

26-Mar-2025, Gihan

Outline

- Clone – code
- Kahoot
- Wildcards
- Functional interfaces

Wildcard Usage Summary

Wildcard Type	Can Accept	Can Read	Can Write
? (Unbounded)	Any type (<code>List<?></code>)	✔ as <code>Object</code>	✗ Cannot add (except <code>null</code>)
? extends <code>Type</code> (Upper Bounded)	<code>Type</code> or any subclass	✔ as <code>Type</code>	✗ Cannot add (except <code>null</code>)
? super <code>Type</code> (Lower Bounded)	<code>Type</code> or any superclass	✗ Only as <code>Object</code>	✔ Can add <code>Type</code> and its subtypes

When to Use Which Wildcard?

- Use ? (Unbounded) when you don't care about the specific type but just need to **iterate** or **print**.
- Use ? extends `T` when you need to **read data** (e.g., process numbers).
- Use ? super `T` when you need to **write data** (e.g., add numbers).

Step 1: Define a Functional Interface

```
java                                                                    Copy Edit

@FunctionalInterface
interface Greeting {
    void sayHello(String name);
}
```

- ✓ The `@FunctionalInterface` annotation ensures that only one abstract method is defined.

Step 2: Implementing with an Anonymous Class

Before Java 8, you would implement the interface using an **anonymous class** like this:

```
java                                                                    Copy Edit

public class AnonymousClassExample {
    public static void main(String[] args) {
        Greeting greeting = new Greeting() {
            @Override
            public void sayHello(String name) {
                System.out.println("Hello, " + name + "!");
            }
        };

        greeting.sayHello("Alice");
    }
}
```

- ✓ Works fine but involves **boilerplate code**.

Step 3: Implementing with a Lambda Expression (Java 8+)

Lambda expressions provide a more concise way to implement functional interfaces.

```
java                                                                    Copy Edit

public class LambdaExample {
    public static void main(String[] args) {
        Greeting greeting = (name) -> System.out.println("Hello, " + name + "!");

        greeting.sayHello("Bob");
    }
}
```

- ✓ **Shorter and cleaner** compared to an anonymous class.

CMSC132: Week 09, Lab 1

26-Mar-2025, Gihan

Outline

- Stability of Sorting Algorithms (from last week)
- How is project 5 going?
- 1:00PM - 1:30PM
 - You have to implement the following,
 - `public static String mostFrequentWord(List<String> words)`
 - `public Set<T> removeInRange(boolean ordered, T lowerBound, T upperBound)`
 - `private Node removeInRangeAux(Node headAux, T lowerBound, T upperBound, Set<T> newSet)`
- 1:30PM - 1:50PM
 - We will go through the solution

Stability of Sorting Algorithms

Sorting Algorithms	In - Place	Stable
Bubble Sort	Yes	Yes
Selection Sort	Yes	No
Insertion Sort	Yes	Yes
Quick Sort	Yes	No
Merge Sort	No (because it requires an extra array to merge the sorted subarrays)	Yes
Heap Sort	Yes	No

CMSC132: Week 09, Lab 2

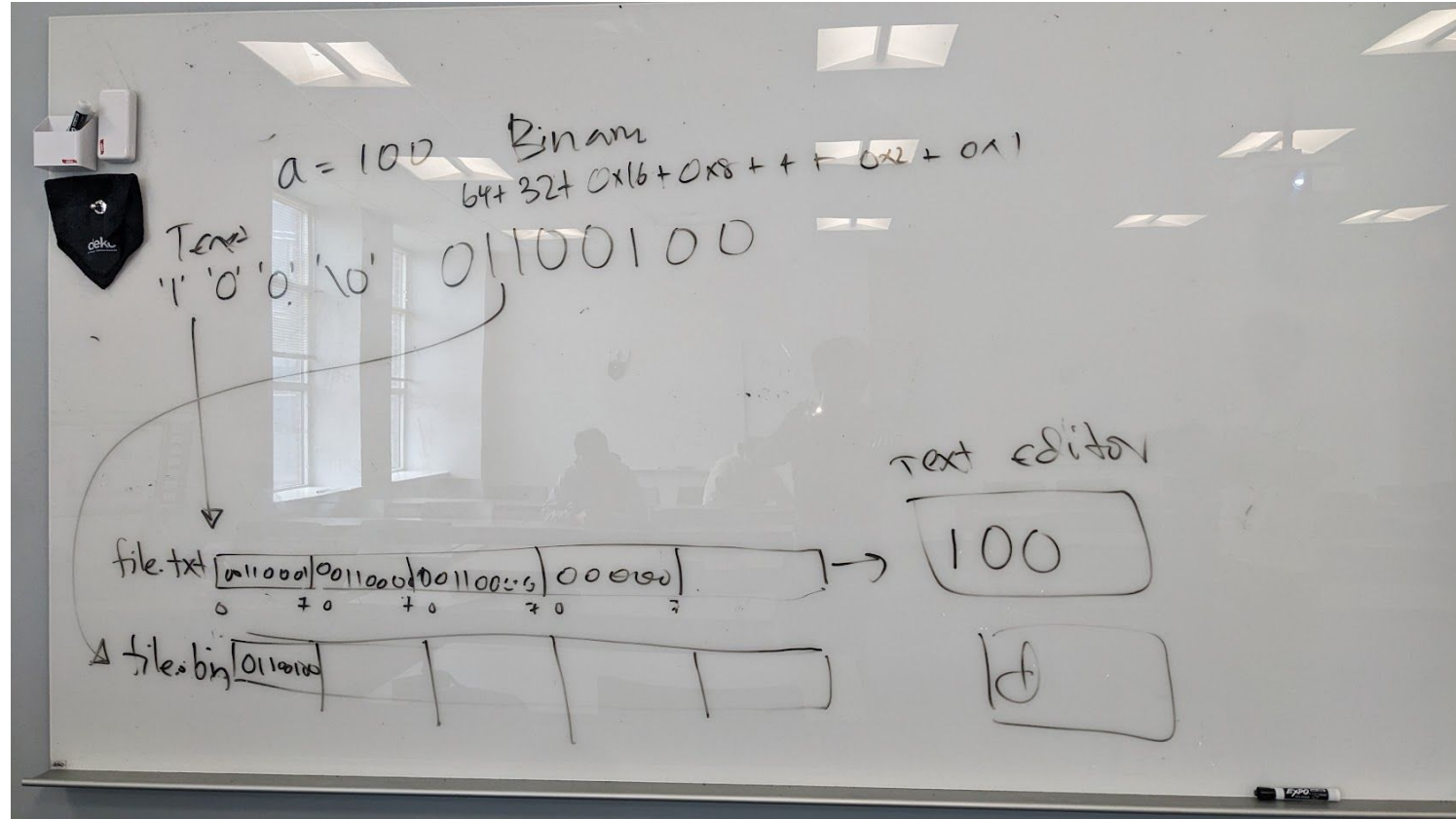
2-Apr-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

Outline

- Quick look at mostFrequentWord() from the previous class.
- Questions about Project 5? — (Deadline Thurs, 03-Apr-2025 11:30 pm)
 - DoublyLinkedList, Deque, Stack, Queue
- Binary Files
- Text Files
 - File
 - FileReader, BufferedReader
 - FileWriter, BufferedWriter
 - PrintWriter

Difference between binary files and text files



CMSC132: Week 10, Lab 1

7-Apr-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

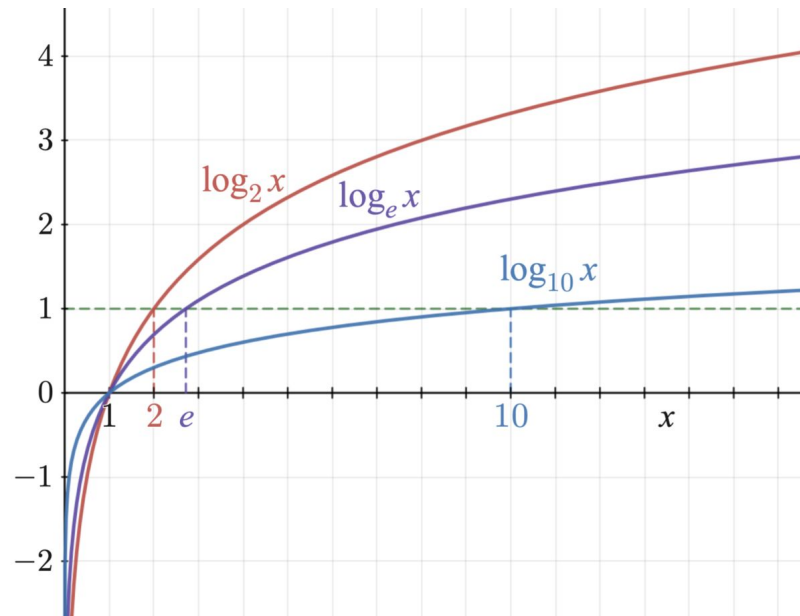
Outline

- How is Project 6 going?
- Logarithms (might be on the exam)
- Quiz 03

Logarithms

$$a^x = y$$
$$\log_a(y) = x$$


Logarithmic Properties	
Product Rule	$\log_a(xy) = \log_a x + \log_a y$
Quotient Rule	$\log_a\left(\frac{x}{y}\right) = \log_a x - \log_a y$
Power Rule	$\log_a x^p = p \log_a x$
Change of Base Rule	$\log_a x = \frac{\log_b x}{\log_b a}$
Equality Rule	If $\log_a x = \log_a y$ then $x = y$



CMSC132: Week 10, Lab 2

9-Apr-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

Outline

- Did you prove the logarithm rules?
- How is Proj 6, HashTables going?
 - Chained hash tables, Open addressed hash tables, HashMaps, HashSets
- Java Input/Output - Binary Files

FileInputStream	FileOutputStream	
BufferedInputStream	BufferedOutputStream	
DataInputStream	DataOutputStream	

- Standard streams in JAVA
 - System.in, System.out, System.err
- (Prove the logarithm rules if time permits)

file.txt

"100"

String "100"

Integer.parseInt()

file.bin

write

100

int

Read

JAV

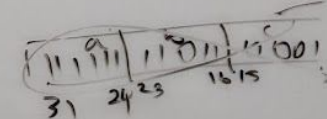
System.in

File Output Stream

Hard Disk

System.

"a"



output.dat

characters

random

System

10100111

5.67x10²

~~1111~~ a



bo.bin

0...10

Buffer

Buffered Output Stream

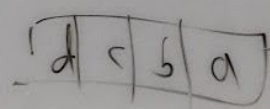
data.bin

12345
3145
the
in



1111

72



b₁
b₃
b₄ = d

CMSC132: Week 11, Lab 1

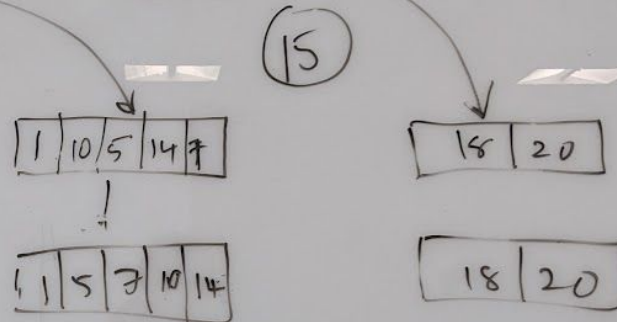
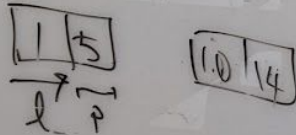
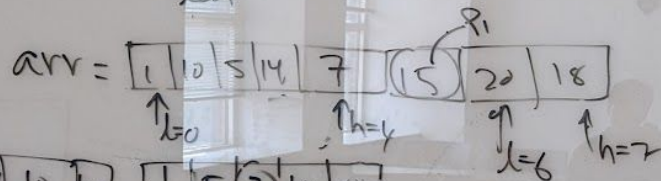
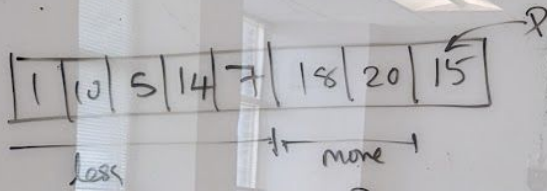
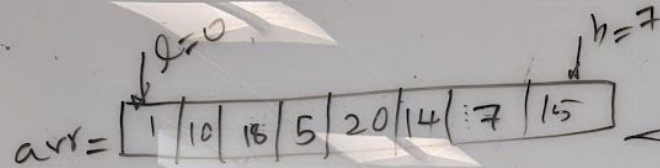
14-Apr-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

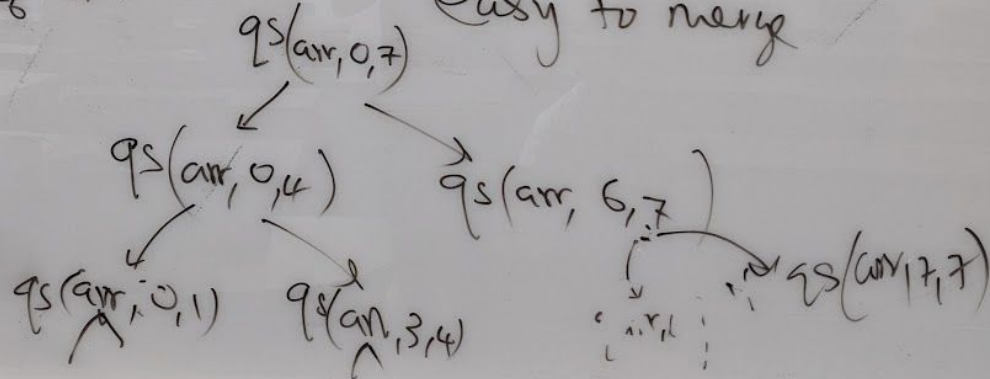
Outline

- How is Project 6 going?
- Quicksort
 - Divide and conquer – partitioning, subproblems
 - Time complexity
 - Memory complexity

Conceptual example

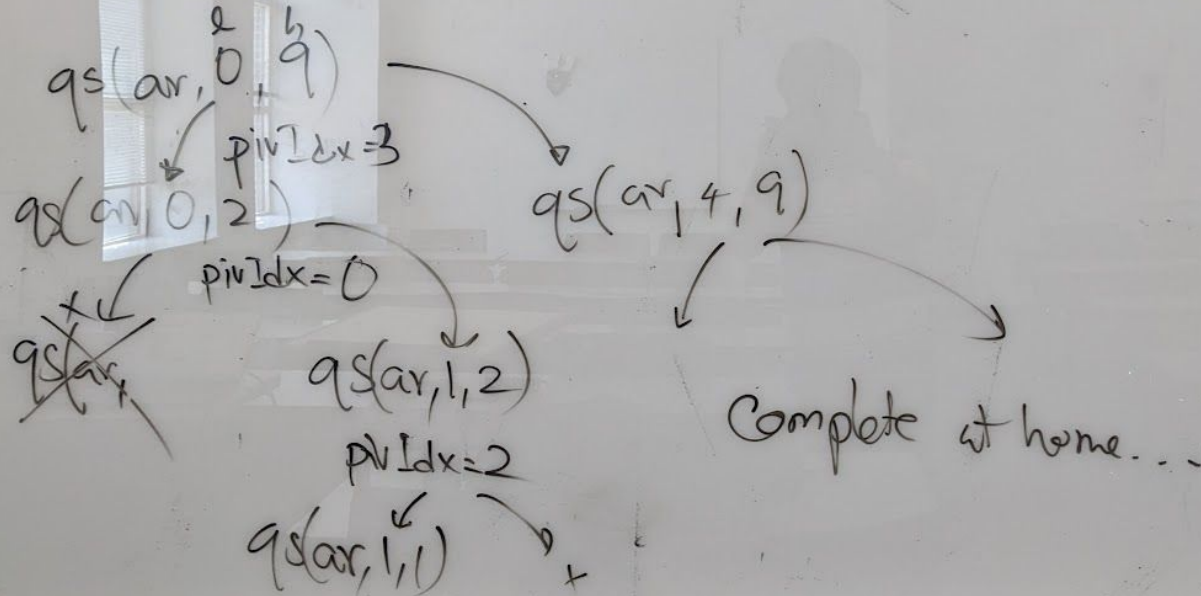


easy to merge



Example from sample code

9	3	7	6	2	8	5	1	10	4
0	1	2	3	4	5	6	7	8	9
1	2	3	4	9	8	5	7	10	6



CMSC132: Week 11, Lab 2

16-Apr-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

Outline

- How is the lectures going? Do you have any questions to discuss in this class?
- Recap – quick sort
- Merge sort - slides
- Merge sort - code

merge sort

Quick sort

0	1	2	3	4	5	6
8	3	1	7	0	10	2

split (heavy)

10	1	2	3	7	8	10
----	---	---	---	---	---	----

3. merge (heavy)

merge easy

1	0	(2)	8	3	7	10
---	---	-----	---	---	---	----

split (easy)

0	2	10
---	---	----

2. sort

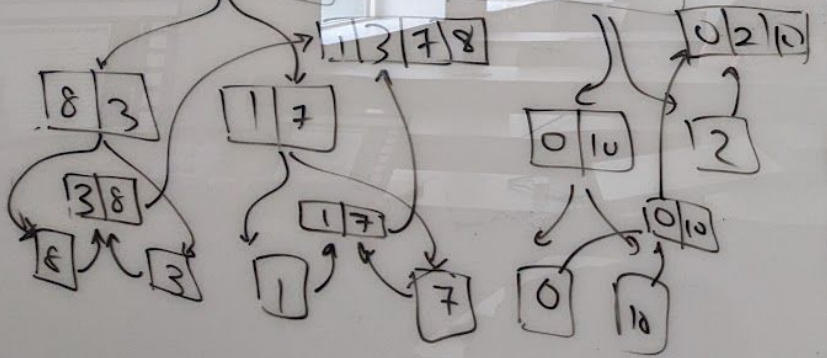
1	3	7	8
8	3	1	7

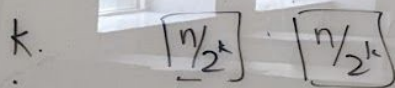
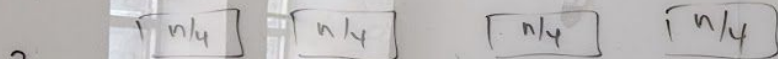
2. sort

split (easy)

0	10	2
---	----	---

2. sort





$k \rightarrow m$
 $\log_2 n$



$$2^m = n$$

$$\log_2(2^m) = \log_2 n$$

$$m \log_2 2 = \log_2 n$$

$$m = \log_2 n$$

$$2 \mid n/2$$

$$4 \mid n/4$$

$$2^k \mid n/2^k$$

$$n \mid 1$$

work

$$O(n)$$

$$O(n)$$

$$O(n)$$

$$O(n \log_2 n)$$



Extra space

n

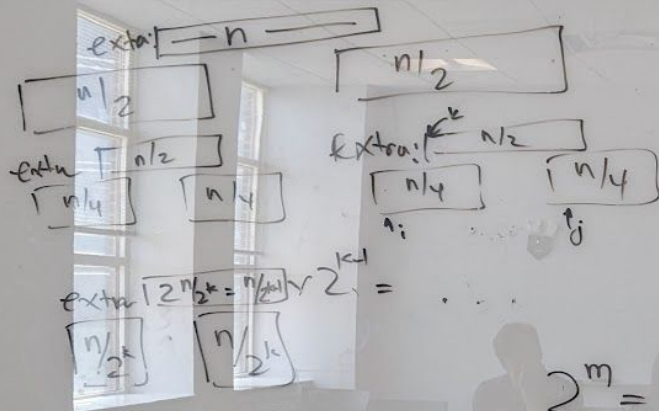
1.

n

2.

n

k.





Exercises

n 1.

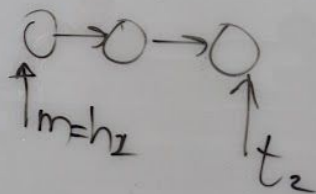
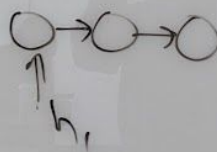
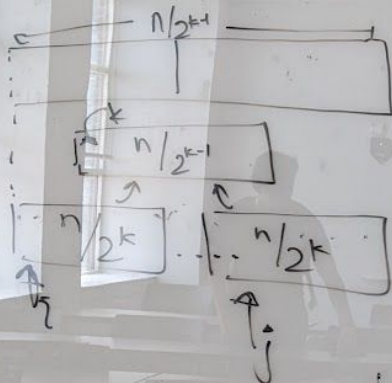
n 2.

n k .

$k \rightarrow m$
 $\log_2 n$



$$len(arr) = n$$



CMSC132: Week 11, Lab 2 (additional)

16-Apr-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

Outline

- Implementing mergesort and quicksort to work with Comparable data types.
- I am not going through this material in the discussion. Therefore, please watch the video on the following link:
 - <https://www.cs.umd.edu/~gihan/resources/cmsc132/>

CMSC132: Week 12, Lab 1

21-Apr-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

Outline

- How is project 7 going?
- Abstract data structure
 - Backing data structures
- Priority Queue
 - add / offer
 - peek
 - poll / remove
- The case for **offer** and **remove**

CMSC132: Week 12, Lab 2

23-Apr-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

Outline

- containsKey() exercise
- Quiz
- Tree traversal, depth (of nodes), height

```
/**  
 * Checks whether the given key exists in the BST.  
 *  
 * @param key the key to search for  
 * @return true if the key exists, false otherwise  
 * @throws IllegalArgumentException if the key is null  
 */  
public boolean containsKey(K key) {
```

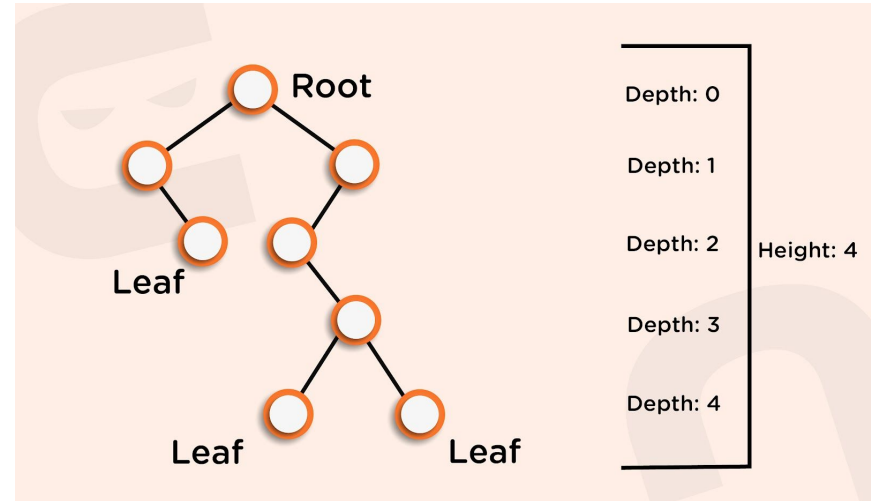
```
/**
 * Checks whether the given key exists in the BST.
 *
 * @param key the key to search for
 * @return true if the key exists, false otherwise
 * @throws IllegalArgumentException if the key is null
 */
```

```
public boolean containsKey(K key) {
    if (key == null) {
        throw new IllegalArgumentException("Key cannot be null");
    }

    Node current = root;
    while (current != null) {
        int comparison = key.compareTo(current.key);
        if (comparison == 0) {
            return true;
        } else if (comparison < 0) {
            current = current.left;
        } else {
            current = current.right;
        }
    }

    return false; // key not found
}
```

- Tree traversal orders
 - Pre-order: Visits: Root $\rightarrow$ Left $\rightarrow$ Right
 - In-order: Visits: Left $\rightarrow$ Root $\rightarrow$ Right
 - Post-order: Visits: Left $\rightarrow$ Right $\rightarrow$ Root
- Height of a tree and depth of a node.



CMSC132: Week 13, Lab 1

28-Apr-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

Outline

- JAVA Threads

CMSC132: Week 14, Lab 1

5-May-2025, Gihan

<https://www.cs.umd.edu/~gihan/resources/cmssc132/>

Outline

- Threads continued