

Part 1—Paper Part

1. (6 points) Consider the following initial array configuration. In each question, assume we want to sort the array in **ascending** order (that is, from smallest to largest).

25	67	83	44	52	8	15	36	0	10
----	----	----	----	----	---	----	----	---	----

- a. (2 points) Suppose the **insertion** sort algorithm from class is applied to the **initial array above**. In the boxes below, show the state of the array after the **fourth** execution of the outer loop. **Clearly mark** the sorted portion and the unsorted portion separately. Note: the sorted part of the array initially contains 1 element.

25	44	52	67	83	8	15	36	0	10
----	----	----	----	----	---	----	----	---	----

sorted | unsorted
← →

- b. (2 points) Suppose the **selection** sort algorithm from class is applied to the **initial array above**. Show the state of the array immediately following the **third** execution of the outer loop. **Clearly mark** the sorted portion and the unsorted portion separately.

0	8	10	44	52	67	15	36	25	83
---	---	----	----	----	----	----	----	----	----

swap sorted not sorted
← →

Iteration 1 25 67 83 44 52 8 15 36 0 10
Iteration 2 0 67 83 44 52 8 15 36 25 10
Iteration 3 0 8 83 44 52 67 15 36 25 10

- c. (2 points) Suppose the **merge** sort algorithm from class is applied to the **initial array above**. Show the state of the two sub-arrays immediately before the final merge.

25	44	52	67	83
----	----	----	----	----

0	8	10	15	36
---	---	----	----	----

2. (6 points) Answer the questions below on the sorting and searching algorithms that we discussed in class.

- a. Match the algorithms below with the Big-O runtime in the **worst** case for each by drawing a clear line from between an algorithm and the answer.

Binary Search		$O(n^2)$
Selection Sort		$O(n)$
Merge Sort		$O(\log n)$
		$O(n \log n)$

- b. Match the algorithms below with the Big-O runtime in the **best** case for each by drawing a clear line from between an algorithm and the answer.

Insertion Sort		$O(n^2)$
Selection Sort		$O(n)$
Merge Sort		$O(n \log n)$
		$O(\log n)$

3. (3 points) Predict the output for the code snippet below.

```
System.out.print(fun(1, 1));

// elsewhere...
public static int fun(int x, int y){
    if(x == 0)
        return y + 1;
    else if(y == 0)
        return x + 1;
    else {
        int temp = fun(x, y-1);
        return fun(x-1, temp);
    }
}
```

Output:

3

Handwritten call stack for the recursive function `fun(1, 1)`:

- `fun(1, 1)` calls `fun(1, 0)` (since `y != 0` and `x != 0`)
 - `fun(1, 0)` returns `1 + 1 = 2` (since `y == 0`)
- `fun(1, 1)` then calls `fun(0, 2)` (since `x != 0` and `y != 0`)
 - `fun(0, 2)` returns `2 + 1 = 3` (since `x == 0`)

The final return value is 3, which is printed.

4. (9 points) Give the Big-O runtime for each of the code snippets below. Answers are worth 1 point each.

a.

```
public void funct1(int n) {  
    int result = 0;  
    for (int i = n; i >= 0; i--) {  
        result += i * i * i;  
        for (int j = 0; j < 2005; j++) {  
            result += j * i;  
        }  
    }  
}
```

Answer:

$O(n)$

b.

```
public static void funct2(int n) {  
    for (int i = 0; i < n; i++) {  
        System.out.println("Break!");  
        for (int j = 0; j < (n * n); j++) {  
            System.out.println("Value of j: " + j);  
        }  
    }  
    for (int i = 0; i < n; i++) {  
        System.out.println("Howdy");  
    }  
}
```

Answer:

$O(n^3)$

c.

```
public static void funct3(int n) {  
    for (int i = 0; i < n; i++) {  
        for (int j = i; j < n; j++) {  
            System.out.println("Whats up");  
        }  
    }  
}
```

Answer:

$O(n^2)$

d.

```
public void funct4(ArrayList<Integer> data, int value, int position) {  
    data.add(position, value); //add "value" at index "position."  
}
```

(i) Answer (Worst Case):

$O(n)$

(ii) Answer (Best Case):

$O(1)$

e.

```
//LinkedList is a singly linked list that has only a first/head Node  
//pointer (i.e., no last/tail Node pointer).  
public void funct5(LinkedList<Integer> data, int value, int position) {  
    data.add(value, position); //insert value in new Node at "position"  
}
```

(i) Answer (Worst Case):

$O(n)$

(ii) Answer (Best Case):

$O(1)$

f.

```
//LinkedList is a singly linked list that has only a first/head Node  
//pointer (i.e., no last/tail Node pointer).  
public void funct6(LinkedList<Integer> data, int position) {  
    data.remove(position); //delete Node at "position"  
}
```

(i) Answer (Worst Case):

$O(n)$

(ii) Answer (Best Case):

$O(1)$

5. (6 points) Consider the following **five** related declarations about various modes of transportation in a simulation of city streets (**Note: no constructors are provided, but when any of the objects below are created, speed=0.0 and direction is always="N"**):

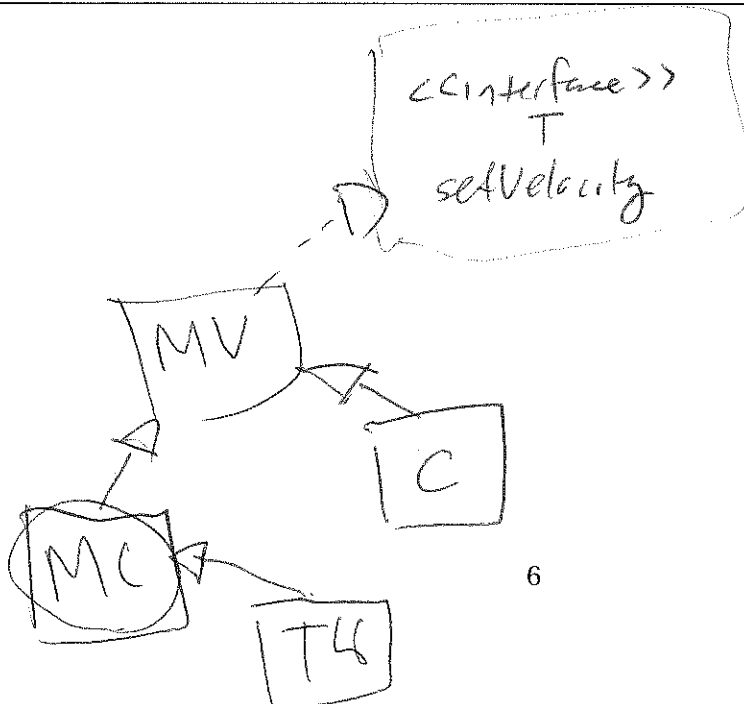
```
interface T {
    public void setVelocity(double speed, int turnDelta); //sets speed & direction
}

abstract class MV implements T {
    double speed;
    int curDirection; //the index in the directions array, e.g., 2 is NE
    public static final String[] directions =
        new String[]{"NW","N","NE","E","SE","S","SW","W"}

    public void setVelocity(double speed, int turnDelta) {
        this.speed = speed;
        if(turnDelta != 0)
            this.turn(turnDelta); //>0 turns right, <0 turns left
        System.out.print("speed:"+this.speed +
            " direction:"+this.directions[this.curDirection]);
    }

    public void turn(int turnDelta) {
        this.curDirection = (this.curDirection + turnDelta);
        if(turnDelta > 0) //turning right
            this.curDirection = this.curDirection % directions.length(); //wrap
        else
            if(this.curDirection < 0) //turning left
                this.curDirection = directions.length() + this.curDirection;
    }
}

public abstract void accelerate(int speedDelta);
}
```



```
class MC extends MV {
    int currentGear; //constructor elided
    public void accelerate(int speedDelta) {
        this.speed += speedDelta;
        System.out.print("Speed:" + this.speed);
    }
}
```

```
class TK extends MC {
    String wheelType;
    public void turn(int turnDelta) {
        super.turn(turnDelta);
        System.out.print(" TKTurn");
    }
}
```

```
class C extends MV {
    //constructor elided
    public void setVelocity(double speed, int turnDelta) {
        this.speed = speed + 9; //always cheat and go faster
        System.out.print("speed" + this.speed +
            " direction:" + this.directions[this.curDirection]);
    }
    public void accelerate(int speedDelta) {
        this.speed += speedDelta;
        System.out.print("Faster: " + this.speed);
    }
}
```

Consider each of the following code snippets independently. (That is, errors in one question will NOT affect the other questions.) For each, write the output, or if it is an error write the type (compile-time or run-time error).

a. `T t1 = new TK();`
`t1.setVelocity(60, 1);`

TKTurn speed:60.0 directionN

b. `MV m1 = new C();`
`((C)m1).setVelocity(90,-2);`

speed99.0 direction:NW

c. `T t2 = new MC();`
`t2.accelerate();`

compile time error

d. `TK t3 = new MV();`
`t3.turn(-2);`

compile time error

e. `MC m2 = new TK();`
`m2.turn(4);`

TKTurn

f. `MC m3 = new MC();`
`((TK)m3).accelerate(20);`

run-time error