

Markscheme

Mock examination — Set 2

Computer science

Higher level

Paper 2

This markscheme is **confidential** and for the exclusive use of the teachers marking this mock examination.

It must **not** be shared with candidates before the mock examination has been sat.

General marking instructions

1. Follow the markscheme provided, award only whole marks and mark only in **RED**.
2. Where a mark is awarded, a tick/check (✓) **must** be placed in the text at the **precise point** where it becomes clear that the candidate deserves the mark. **One tick to be shown for each mark awarded.**
3. Sometimes, careful consideration is required to decide whether or not to award a mark. In these cases write a brief comment in the margin to support your decision, especially for moderation purposes. It should be remembered that the script may be returned to the candidate.
4. Personal codes/notations are unacceptable.
5. Where an answer to a part question is worth no marks but the candidate has attempted the part question, write a zero next to the answer. Where the candidate has not attempted the part question, write “NR”.
6. Ensure that you have viewed **every** page including any additional sheets.
7. A mark should not be awarded where there is contradiction within an answer. Write “CON” next to the contradiction.
8. **If a candidate has attempted more than one Option** within a paper, mark all the candidate’s work and award the marks for the higher scoring Option.

Subject details: Computer Science HL Paper 2 markscheme (Mock examination — Set 2)

Mark allocation

Candidates are required to answer **all** questions in **one** Option. Total 80 marks.

General

A markscheme often has more specific points worthy of a mark than the total allows. This is intentional. Do not award more than the maximum marks allowed for that part of a question.

When deciding upon alternative answers by candidates to those given in the markscheme, consider the following points:

- Each statement worth one point has a separate line and the end is signified by means of a semi-colon (;).
- An alternative answer or wording is indicated in the markscheme by a “/”; either wording can be accepted.
- Words in (...) in the markscheme are not necessary to gain the mark.
- If the candidate’s answer has the same meaning or can be clearly interpreted as being the same as that in the markscheme then award the mark.
- Mark positively. Give candidates credit for what they have achieved and for what they have got correct, rather than penalizing them for what they have not achieved or what they have got wrong.
- Remember that many candidates are writing in a second language; be forgiving of minor linguistic slips. In this subject effective communication is more important than grammatical accuracy.
- Occasionally, a part of a question may require a calculation whose answer is required for subsequent parts. If an error is made in the first part then it should be penalized. However, if the incorrect answer is used correctly in subsequent parts then **follow through** marks should be awarded. Indicate this with “FT”.

General guidance

Issue	Guidance
Answering more than the quantity of responses prescribed in the questions	• In the case of an “identify” question, read all answers and mark positively up to the maximum marks. Disregard incorrect answers. • In the case of a “describe” question, which asks for a certain number of facts eg “describe two kinds”, mark the first two correct answers. This could include two descriptions, one description and one identification, or two identifications. • In the case of an “explain” question, which asks for a specified number of explanations eg “explain two reasons ...”, mark the first two correct answers. This could include two full explanations, one explanation, one partial explanation etc.

Option A — Java

1. (a) Award [3] as follows:

correct values of i and correct TRUE/FALSE values in the $i \leq N$? column;
 correct TRUE/FALSE values in both remainder columns;
 correct values of total and correct output;

i	$i \leq N$?	$i \% 3 = 0$?	$i \% 5 = 0$?	total	output
1	TRUE	FALSE	FALSE	0	
2	TRUE	FALSE	FALSE		
3	TRUE	TRUE		3	
4	TRUE	FALSE	FALSE		
5	TRUE	FALSE	TRUE	8	
6	TRUE	TRUE		14	
7	TRUE	FALSE	FALSE		
8	FALSE				14

Note: When i is 3 or 6, the $i \% 5 = 0$? decision is not reached; do not penalize a value written in that column for these rows. Accept any layout in which the values appear in the correct order. Award FT for later values that follow correctly from an earlier error.

(b) Award [2 max]

$O(N)$ / linear time;
 ...because the loop is repeated N times / once for each value of i from 1 to N ;
 ...and each pass carries out a fixed (maximum) number of steps / at most three decisions and two assignments;
 ...so doubling N (about) doubles the time taken;

Note: Award [1] for $O(N)$ and [1] for a valid description linked to the number of passes through the loop. Do not accept $O(N^2)$, $O(\log N)$ or $O(1)$ for the first mark; a correct description of the loop can still gain [1].

(c) Award [2] as follows: [1] for each step.

Step 1: abstraction;
 Step 2: pattern recognition;

Note: Accept generalization for Step 2. Do not accept decomposition or algorithmic design for either step.

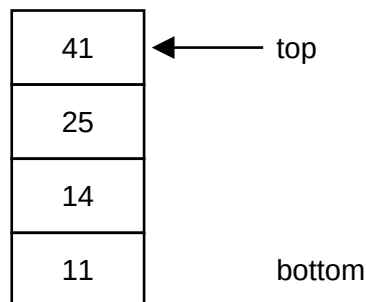
(d) Award [3] as follows:

pass 1: the smallest value, 8, is found (by comparing all the values) and swapped with the first value, 29, giving 8, 14, 37, 29, 21;
 pass 2: the smallest of the values at index 1 to 4 is 14, which is already at index 1, so it is not moved / the list is unchanged;
 pass 3: the smallest of the values at index 2 to 4 is 21, which is swapped with 37, giving 8, 14, 21, 29, 37, and pass 4 leaves 29 at index 3, so the list is sorted;

Note: Award [1 max] for a correct general description of a selection sort that is not applied to this list. Accept the lists after each pass, provided that it is clear that the smallest remaining value is selected on each pass. Award FT for later passes that follow correctly from an earlier error. Accept a selection sort that selects the largest value on each pass: 37 is swapped with 21, giving 29, 14, 21, 8, 37 [1]; 29 is swapped with 8, giving 8, 14, 21, 29, 37 [1]; 21 and 14 are already in place, so the list is sorted [1]. A description of a bubble sort (comparing and swapping adjacent values) gains no marks.

(e) Award [2].

correct trays in the correct order, from bottom to top 11, 14, 25, 41 [1];
correct indication of the top (41) [1];



Note: Do not award the first mark if 18 or 30 is shown in the stack. Accept the stack drawn horizontally, provided that the top is clearly labelled.

(f) Award [3] as follows: [1] for each correct line (or equivalent).

repeat until isEmpty() is true / while isEmpty() is false / size() times: pop a tray number and push it onto the second stack;

compare each popped number with the largest number found so far, and keep the larger (the first number popped / peek() gives the starting value); output the largest once every number has been checked;

then repeat until the second stack is empty: pop a number from the second stack and push it back onto trays;

Note: Answer must demonstrate an algorithmic approach. Pushing the numbers back from the second stack restores their original order. Storing the numbers in an array, list or queue rather than in the second stack does not gain the first or the third mark, because only the operations in the table may be used.

2. (a) Award **[2 max]**. Award **[1]** for a reason and **[1]** for an expansion.

total is only needed inside showTotals() / no other method uses total;
...so other methods cannot change it (by mistake) / the value cannot be affected by code elsewhere in the program;

a local variable only exists while the method is running;
...so its memory is released when the method ends / it is created again, with a new value, each time the method is called;

the same name can be used for variables in other methods (without a conflict);
...so the method is self-contained / easier to reuse/test/debug;

*Note: Award **[1 max]** for a definition of local and global variables that does not give a reason.*

- (b) Award **[2]** as follows: **[1]** for each correct line.

glass: 140;
paper: 350;

Note: paper: 210 (the intended total) does not gain the second mark, because total is not reset. Ignore minor differences in spacing or punctuation.

- (c) Award **[3 max]**

the statement outputs the value of total at the start of the calculation for each material / before the masses of each material are added;
for m = 0 it outputs total = 0, but for m = 1 it outputs total = 140 (the total for glass) instead of 0;

so the programmer can see that total is not reset to 0 for each material / the totals of the earlier materials are carried over;
move the statement int total = 0; inside the outer loop, before the inner loop / add total = 0; as the first statement in the outer loop;

*Note: Award **[2 max]** if no correct correction is given. Accept answers that refer to the values output for m = 2 (350) or m = 3 (525).*

- (d) Award **[3]** as follows:

Award [1] for opening week.txt in append mode (eg new FileWriter("week.txt", true));
Award [1] for writing the day number and the four masses for that day (mass[0][d] to mass[3][d]), separated by commas and followed by a new line;
Award [1] for closing the file;

Example answer

```
public static void saveDay(int d) throws IOException
{
    FileWriter writer = new FileWriter("week.txt", true);
    String line = "" + d;
    for (int m = 0; m < 4; m++)
    {
        line = line + "," + mass[m][d];
    }
    writer.write(line + "\n");
    writer.close();
}
```

Alternative answer

```
public static void saveDay(int d) throws IOException
{
    PrintWriter out = new PrintWriter(new FileWriter("week.txt", true));
    out.println(d + "," + mass[0][d] + "," + mass[1][d] + ","
               + mass[2][d] + "," + mass[3][d]);
    out.close();
}
```

Note: Opening the file without append mode (eg new FileWriter("week.txt")) does not gain the first mark, because it deletes the lines already in the file. The masses may be written using a loop or one at a time. mass[d][m] does not gain the second mark, and nor does a trailing comma after the last mass (eg 5,20,45,35,15,), because the format is given. Penalize write(int) (which writes a character code, not the digits) or joining with char literals such as ',' (which adds character codes) once only. Accept a BufferedWriter with newLine(), or Files.writeString() with StandardOpenOption.APPEND. Accept try-with-resources (try (FileWriter writer = ...) { ... }) for the closing mark. Exception handling (throws IOException or try/catch) is not required for any mark. Ignore minor syntax errors; penalize a repeated error once only.

(e) Award **[5]** as follows:

Award [1] for an outer loop through the 7 days;
Award [1] for an inner loop through the 4 materials that accesses mass[m][d] for that day;
Award [1] for initializing best (eg best = 0) for each day, inside the outer loop and before the inner loop, together with the greatest mass if it is stored (eg greatest = mass[0][d]);
Award [1] for comparing mass[m][d] with the greatest mass so far (eg greatest or mass[best][d]) and updating best (and the greatest mass, if it is stored separately);
Award [1] for outputting the day number and materials[best] once for each day, after the inner loop;

Example answer

```
public static void dailyBest()
{
    for (int d = 0; d < 7; d++)
    {
        int greatest = -1;      // or mass[0][d] with m from 1
        int best = 0;
        for (int m = 0; m < 4; m++)
        {
            if (mass[m][d] > greatest)
            {
                greatest = mass[m][d];
                best = m;
            }
        }
        System.out.println ("Day " + d + ": " + materials[best]);
    }
}
```

*Note: Accept an answer that loops through the materials in the outer loop, stores the greatest mass and its material for each day in two arrays of size 7, and then outputs them; award the equivalent marks. Accept tracking only the index of the best material: best = 0, then if (mass[m][d] > mass[best][d]) best = m; A loop that starts at m = 1 is correct only if, for each day, best is initialized to 0 (and the greatest mass, if stored, to mass[0][d]). best may be initialized once, before the outer loop, only if the inner loop starts at m = 0 and either only the index is tracked or the greatest mass starts at -1 for each day. Initializing greatest = mass[0][d] each day but best only once does not gain the third mark. Ignore the format of the output. An answer that uses a library method that is not permitted (including Math.max()) gains **[3 max]**. Ignore minor syntax errors; penalize a repeated error once only.*

3. (a) Award **[2]** as follows:

Award [1] for stating that the new class (eg AppNotifier) implements Notifier, so it must provide its own update() method;

Award [1] for explaining that Parcel refers to its notifiers only through the Notifier type and only calls update(), so an AppNotifier object can be passed to addNotifier() and is informed by setStatus() like any other notifier / Parcel does not need to know which class a notifier belongs to;

(b) Award **[4]** as follows:

Award [1] for a class header showing that SmsNotifier implements Notifier;

Award [1] for a private attribute for the phone number that is assigned by the constructor;

Award [1] for a public method update(String code, String status) with the same signature as in Notifier (void, no return value);

Award [1] for outputting the message in the required format only when status is "Out for delivery" or "Delivered";

*Accept equals(), compareTo(...) == 0 or ==; penalize == for strings once only in the paper.
Accept @Override before update() and minor differences in the spacing of the output.*

Example answer

```
public class SmsNotifier implements Notifier{

    private String phone;

    public SmsNotifier(String phone)
    {
        this.phone = phone;
    }

    public void update(String code, String status)
    {
        if (status.equals("Out for delivery")
            || status.equals("Delivered"))
            System.out.println("SMS to " + phone + ": " + code
                               + " " + status);
    }
}
```

Without implements Notifier, an SmsNotifier object cannot be passed to addNotifier() or stored in notifiers (the code does not compile).

(c) Award **[3 max]**

Parcel is the subject/publisher and the notifiers (EmailNotifier and SmsNotifier objects) are its observers/subscribers;

a notifier subscribes through addNotifier(), which stores it in notifiers;

when the status changes, setStatus() notifies every registered observer by calling update();

...each observer decides how to respond, eg SmsNotifier texts only for two statuses;

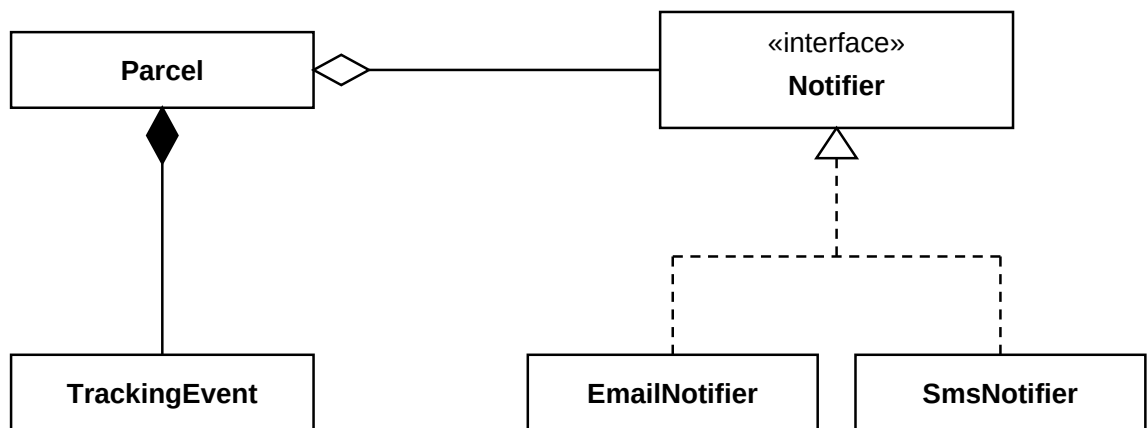
Award [1 max] if the roles (subject/publisher and observer/subscriber) are only named, without reference to how the classes carry them out.

(d) Award **[3]** as follows:

Award [1] for composition between Parcel and TrackingEvent, shown by a line with a filled (solid) diamond at the Parcel end;
Award [1] for aggregation between Parcel and Notifier, shown by a line with a hollow (unfilled) diamond at the Parcel end;
Award [1] for EmailNotifier and SmsNotifier each joined to Notifier by a line with a hollow triangle at the Notifier end (realization);

Accept a solid line instead of a dashed line for the realization, and «interface» may be omitted. Accept aggregation lines from Parcel to both EmailNotifier and SmsNotifier instead of to Notifier. Multiplicities are not required. A diamond at the wrong end does not earn the mark.

Example answer



(e) Award **[3 max]**

the **TrackingEvent** objects are created inside the **Parcel** object (by `setStatus()`) and only the parcel refers to them (through `history`);
...so they are deleted with the **Parcel** object / cannot exist without it (composition);
the **Notifier** objects are created outside the **Parcel** object (by the driver class) and are passed in through `addNotifier()`;
...so they still exist after the **Parcel** object is deleted and go on informing the customer about their other parcels (aggregation);

Award **[2 max]** if only one of the two kinds of object is addressed. Award **[1 max]** if the relationships are only named (composition, aggregation) without describing what happens to the objects. Accept "garbage-collected" for deleted.

(f) Award [5] as follows:

Award [1] for handling an empty history (`numEvents == 0`), so that the first status is always recorded;

Award [1] for comparing status with `history[numEvents - 1].getStatus()`, and doing nothing (no event added, no notifier updated) if they are the same;

Award [1] for creating a new `TrackingEvent` object from status and time;

Award [1] for storing the new event in `history[numEvents]` and adding 1 to `numEvents`;

Award [1] for a loop over the first `numNotifiers` elements of `notifiers` that calls `update(code, status)` on each one;

Award the first mark only if `history[numEvents - 1]` is never accessed when `numEvents` is 0 (this causes an `ArrayIndexOutOfBoundsException`). Award the fifth mark only if the loop stops at `numNotifiers`: a for-each loop over the whole `notifiers` array reaches null elements and causes a `NullPointerException`. Accept an early return for a repeated status. Accept `equals()`, `compareTo(...) == 0` or `==`; penalize `==` for strings once only in the paper.

Example answer

```
public void setStatus(String status, String time)
{
    if (numEvents == 0
        || !history[numEvents - 1].getStatus().equals(status))
    {
        history[numEvents] = new TrackingEvent(status, time);
        numEvents = numEvents + 1;
        for (int i = 0; i < numNotifiers; i++)
            notifiers[i].update(code, status);
    }
}
```

Alternative answer

```
public void setStatus(String status, String time)
{
    boolean repeat = false;
    if (numEvents > 0)
        if (history[numEvents - 1].getStatus().equals(status))
            repeat = true;
    if (repeat == false)
    {
        TrackingEvent e = new TrackingEvent(status, time);
        history[numEvents] = e;
        numEvents = numEvents + 1;
        for (int i = 0; i < numNotifiers; i++)
            notifiers[i].update(code, status);
    }
}
```

Calling `update()` through a `Notifier` reference is dynamic polymorphism: the version in `EmailNotifier` or in `SmsNotifier` is selected at run time.

4. (a) **Award [2 max]**

an ADT is defined by the operations that can be carried out on it / by its behaviour (eg add(), remove() and contains() in the table);
it describes what each operation does, not how it is done: the table does not say how the set is stored (eg in a hash table or an array) / its implementation is hidden;
it specifies the kind of data/values it holds (eg a collection of different medicine names, in no particular order);
it can be implemented in different ways (eg HashSet or TreeSet) without changing the code that uses its operations;
it gives a high-level description of a data structure and its operations, so programmers can use it without knowing its implementation;

Award [1] for each property outlined. Award [1 max] if properties are only named (eg “abstraction”, “encapsulation”) without an outline.

(b) **Award [2] as follows:**

Award [1] for 731 at index 7;
Award [1] for 384 at index 0;

731 % 11 = 5 and slots 5 and 6 are occupied, so 731 is stored in slot 7. 384 % 11 = 10 and slot 10 is occupied, so the search continues from slot 0, which is empty.

Index	0	1	2	3	4	5	6	7	8	9	10
Code	384		189			357	412	731		603	846

(c) **Award [3] as follows:**

Award [1] for a loop that visits each medicine in prescription;
Award [1] for testing whether each medicine is in stock using contains();
Award [1] for returning false as soon as a medicine is not in stock, and true only after every medicine has been checked;

Using an operation that is not in the table (eg containsAll(), removeAll() or retainAll()) gains [0]. Do not award the third mark if either set is changed. Award [1 max] if the method tests whether stock is a subset of prescription instead (eg it loops over stock). Accept a Boolean flag that is set to false when a medicine is not in stock and returned after the loop.

Example answer

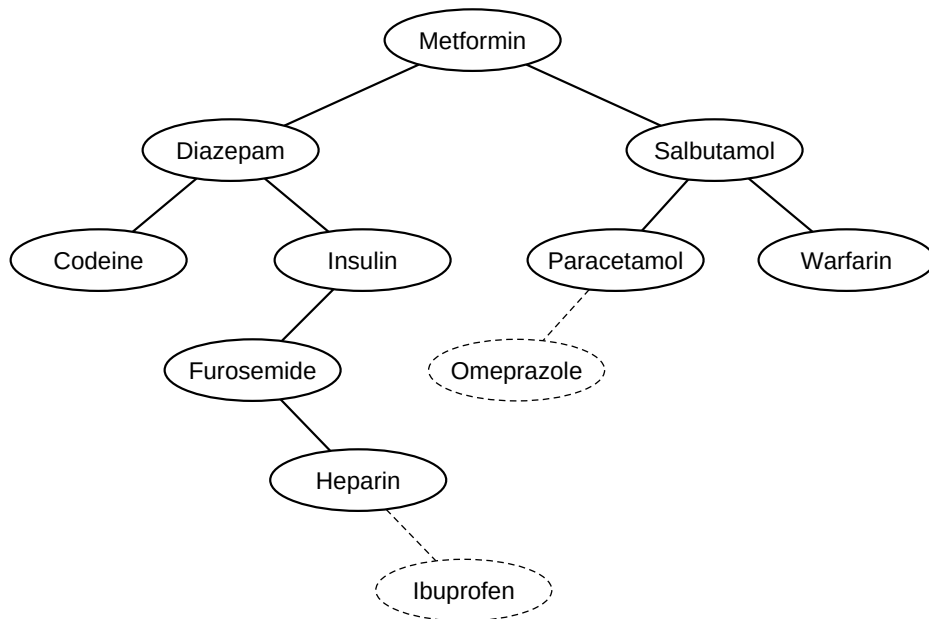
```
public boolean canDispense(HashSet<String> prescription,
                           HashSet<String> stock)
{
    for (String m : prescription)
    {
        if (!stock.contains(m))        // allow == false
            return false;
    }
    return true;
}
```

Alternative answer

```
public boolean canDispense(HashSet<String> prescription,  
                           HashSet<String> stock)  
{  
    int found = 0;  
    for (String m : prescription)  
        if (stock.contains(m))  
            found = found + 1;  
    return found == prescription.size();  
}
```

5. (a) Award **[2]** as follows:

Award [1] for Omeprazole as the left child of Paracetamol;
Award [1] for Ibuprofen as the right child of Heparin;



The new nodes are shown with a thinner, dashed outline. The rest of the tree must be unchanged.

(b) Award **[2 max]**

Metformin, Diazepam, Codeine, Insulin, Furosemide, Heparin, Salbutamol, Paracetamol, Warfarin;

Award **[1]** for Metformin, Diazepam, Codeine, Insulin, Furosemide, Heparin in this order (the root, then its left subtree) and **[1]** for Salbutamol, Paracetamol, Warfarin in this order after them. An in-order (alphabetical) list scores **[0]**.

(c) Award **[3]** as follows:

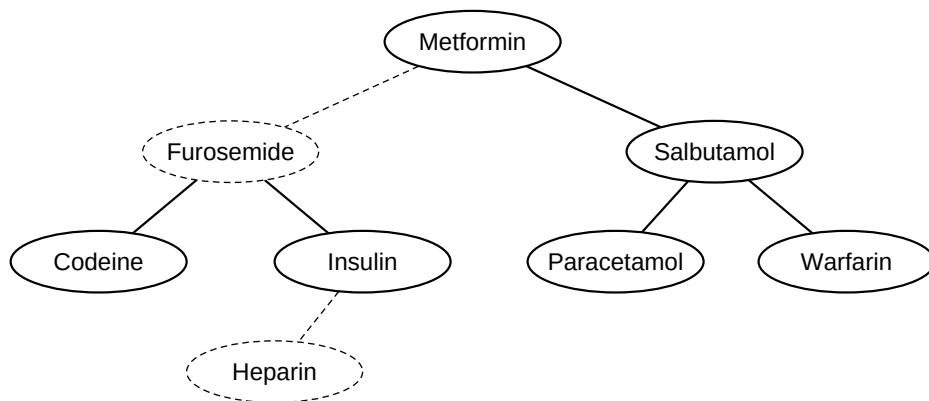
Award [1] for searching from the root: Diazepam is compared with Metformin and, as it comes before Metformin, the left child is visited, where Diazepam is found;

Award [1] for replacing Diazepam, which has two children, with its in-order successor Furosemide (the leftmost node in its right subtree) / with its in-order predecessor Codeine (the rightmost node in its left subtree);

Award [1] for removing the successor from its old position, so that its right child Heparin becomes the left child of Insulin / for removing the predecessor Codeine, which is a leaf, from its old position;

Accept any other method that leaves a valid BST containing all eight of the other names, eg Codeine replaces Diazepam and the subtree rooted at Insulin becomes the right subtree of Codeine; award the second and third marks for it. Accept a correct general description that is applied to this tree (eg “the smallest name in the right subtree of Diazepam”). Award **[1 max]** if the resulting tree is not a binary search tree or loses any name other than Diazepam. Allow the use of diagrams to illustrate any of the above points.

Example answer (in-order successor)



The two nodes that have moved (Furosemide and Heparin) are shown with a thinner, dashed outline.

(d) Award **[2]** as follows:

Award [1] for the four calls in this order: remaining(400, 3), remaining(200, 2), remaining(100, 1), remaining(50, 0);

Award [1] for stating that every call returns 50 / 50.0 (the base case returns 50.0 and each call passes it back unchanged);

Accept 200.0, 100.0 and 50.0 as the first arguments. Do not award the second mark if the values returned are given as 400, 200, 100 and 50: these are the arguments, not the values returned. The answer may be set out as a table, as shown.

Call	Value returned
remaining(400, 3)	50.0
remaining(200.0, 2)	50.0
remaining(100.0, 1)	50.0
remaining(50.0, 0)	50.0

(e) Award **[4]** as follows:

Award [1] for a base case that returns tablets[0] when n is 1;

Award [1] for a recursive call busiest(tablets, n - 1), which leads to the base case;

Award [1] for comparing tablets[n - 1] with the value returned by the recursive call;

Award [1] for returning the larger of the two values;

Award **[1 max]** if the method is not recursive (eg it uses a loop). Award **[2 max]** if Math.max() is used. Accept a base case $n == 0$ that returns 0, because the numbers of tablets are never negative. Accept a solution that makes the recursive call twice (eg once in the condition and once in the else branch): it returns the correct value but repeats work.

Example answer

```
public int busiest(int[] tablets, int n)
{
    if (n == 1)
        return tablets[0];
    int rest = busiest(tablets, n - 1);
    if (tablets[n - 1] > rest)
        return tablets[n - 1];
    else
        return rest;
}
```

6. (a) Award **[5]** as follows:

The algorithm will:

- set the prev pointer of newTrack to current;
- set the next pointer of newTrack to the next pointer of current / to the node after current;
- if current is the tail / the next pointer of current is null, set tail to newTrack;
- otherwise, set the prev pointer of the node after current to newTrack;
- set the next pointer of current to newTrack, after the node after current has been linked to newTrack / saved;

*Award **[3 max]** if the next pointer of current is changed before the node after current has been saved or linked to newTrack (the rest of the playlist is lost). Award **[4 max]** if the tail case is not handled. Allow the use of diagrams to illustrate any of the above points.*

Accept the steps in any order that does not lose the node after current, eg set the next pointer of newTrack first, then the next pointer of current, then use the next pointer of newTrack to reach the following node.

Algorithm is:

1. Set the prev pointer of newTrack to current.
2. Set the next pointer of newTrack to the next pointer of current.
3. If the next pointer of current is null (current is the tail), set tail to newTrack.
4. Otherwise, set the prev pointer of the node after current to newTrack.
5. Set the next pointer of current to newTrack.

(b) Award **[5]** as follows:

Award [1] for traversing the list from head with a temporary node, comparing each title with title, and stopping at the first match or at the end of the list;

Award [1] for removing the first node: setting head to the second node and the prev pointer of the new head to null;

Award [1] for removing the last node: setting tail to the previous node and the next pointer of the new tail to null;

Award [1] for removing a node in the middle: setting the next pointer of the previous node to the following node **and** the prev pointer of the following node to the previous node;

Award [1] for outputting a suitable message only when no node matches, without changing the list;

Accept equals(), compareTo(...) == 0 or ==; penalize == for strings once only in the paper. Accept the combined form shown in the alternative answer, which earns the second, third and fourth marks. Removing every matching node does not earn the first mark.

Example answer

```
public void removeTrack(String title)
{
    Node temp = head;
    while (temp != null && !temp.title.equals(title))
        temp = temp.next;
    if (temp == null)
        System.out.println(title + " is not in the playlist");
    else if (temp == head)
    {
        head = temp.next;
        head.prev = null;
    }
    else if (temp == tail)
    {
        tail = temp.prev;
        tail.next = null;
    }
    else
    {
        temp.prev.next = temp.next;
        temp.next.prev = temp.prev;
    }
}
```

Alternative answer

```
public void removeTrack(String title)
{
    Node temp = head;
    boolean found = false;
    while (temp != null && found == false)
    {
        if (temp.title.equals(title))
            found = true;
        else
            temp = temp.next;
    }
    if (found == false)
        System.out.println(title + " is not in the playlist");
    else
    {
        if (temp.prev != null)
            temp.prev.next = temp.next;
        else
            head = temp.next;
        if (temp.next != null)
            temp.next.prev = temp.prev;
        else
            tail = temp.prev;
    }
}
```

Option B — Python

7. (a) Award [3] as follows:

correct values of i and correct TRUE/FALSE values in the $i \leq N$? column;
 correct TRUE/FALSE values in both remainder columns;
 correct values of total and correct output;

i	$i \leq N$?	$i \% 3 = 0$?	$i \% 5 = 0$?	total	output
1	TRUE	FALSE	FALSE	0	
2	TRUE	FALSE	FALSE		
3	TRUE	TRUE		3	
4	TRUE	FALSE	FALSE		
5	TRUE	FALSE	TRUE	8	
6	TRUE	TRUE		14	
7	TRUE	FALSE	FALSE		
8	FALSE				14

Note: When i is 3 or 6, the $i \% 5 = 0$? decision is not reached; do not penalize a value written in that column for these rows. Accept any layout in which the values appear in the correct order. Award FT for later values that follow correctly from an earlier error.

(b) Award [2 max]

$O(N)$ / linear time;
 ...because the loop is repeated N times / once for each value of i from 1 to N ;
 ...and each pass carries out a fixed (maximum) number of steps / at most three decisions and two assignments;
 ...so doubling N (about) doubles the time taken;

Note: Award [1] for $O(N)$ and [1] for a valid description linked to the number of passes through the loop. Do not accept $O(N^2)$, $O(\log N)$ or $O(1)$ for the first mark; a correct description of the loop can still gain [1].

(c) Award [2] as follows: [1] for each step.

Step 1: abstraction;
 Step 2: pattern recognition;

Note: Accept generalization for Step 2. Do not accept decomposition or algorithmic design for either step.

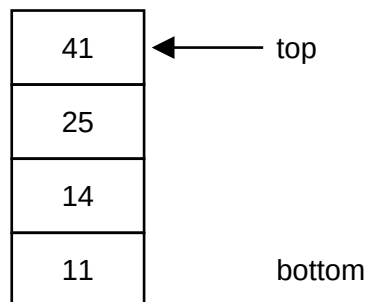
(d) Award [3] as follows:

pass 1: the smallest value, 8, is found (by comparing all the values) and swapped with the first value, 29, giving 8, 14, 37, 29, 21;
 pass 2: the smallest of the values at index 1 to 4 is 14, which is already at index 1, so it is not moved / the list is unchanged;
 pass 3: the smallest of the values at index 2 to 4 is 21, which is swapped with 37, giving 8, 14, 21, 29, 37, and pass 4 leaves 29 at index 3, so the list is sorted;

Note: Award [1 max] for a correct general description of a selection sort that is not applied to this list. Accept the lists after each pass, provided that it is clear that the smallest remaining value is selected on each pass. Award FT for later passes that follow correctly from an earlier error. Accept a selection sort that selects the largest value on each pass: 37 is swapped with 21, giving 29, 14, 21, 8, 37 [1]; 29 is swapped with 8, giving 8, 14, 21, 29, 37 [1]; 21 and 14 are already in place, so the list is sorted [1]. A description of a bubble sort (comparing and swapping adjacent values) gains no marks.

(e) Award [2].

correct trays in the correct order, from bottom to top 11, 14, 25, 41 [1];
correct indication of the top (41) [1];



Note: Do not award the first mark if 18 or 30 is shown in the stack. Accept the stack drawn horizontally, provided that the top is clearly labelled.

(f) Award [3] as follows: [1] for each correct line (or equivalent).

repeat until isEmpty() is true / while isEmpty() is false / size() times: pop a tray number and push it onto the second stack;

compare each popped number with the largest number found so far, and keep the larger (the first number popped / peek() gives the starting value); output the largest once every number has been checked;

then repeat until the second stack is empty: pop a number from the second stack and push it back onto trays;

Note: Answer must demonstrate an algorithmic approach. Pushing the numbers back from the second stack restores their original order. Storing the numbers in an array, list or queue rather than in the second stack does not gain the first or the third mark, because only the operations in the table may be used.

8. (a) Award **[2 max]**. Award **[1]** for a reason and **[1]** for an expansion.

total is only needed inside show_totals() / no other function uses total;
...so other functions cannot change it (by mistake) / the value cannot be affected by code elsewhere in the program;

a local variable only exists while the function is running;
...so its memory is released when the function ends / it is created again, with a new value, each time the function is called;

the same name can be used for variables in other functions (without a conflict);
...so the function is self-contained / easier to reuse/test/debug;

*Note: Award **[1 max]** for a definition of local and global variables that does not give a reason.*

- (b) Award **[2]** as follows: **[1]** for each correct line.

glass: 140;
paper: 350;

Note: paper: 210 (the intended total) does not gain the second mark, because total is not reset. Ignore minor differences in spacing or punctuation.

- (c) Award **[3 max]**

the statement outputs the value of total at the start of the calculation for each material / before the masses of each material are added;
for m = 0 it outputs total = 0, but for m = 1 it outputs total = 140 (the total for glass) instead of 0;
so the programmer can see that total is not reset to 0 for each material / the totals of the earlier materials are carried over;
move the statement total = 0 inside the outer loop, before the inner loop / add total = 0 as the first statement in the outer loop;

*Note: Award **[2 max]** if no correct correction is given. Accept answers that refer to the values output for m = 2 (350) or m = 3 (525).*

- (d) Award **[3]** as follows:

Award [1] for opening week.txt in append mode (open("week.txt", "a"));
Award [1] for writing the day number and the four masses for that day (mass[0][d] to mass[3][d]), separated by commas and followed by a new line;
Award [1] for closing the file;

Example answer

```
def save_day(d):  
    file = open("week.txt", "a")  
    line = str(d)  
    for m in range(0, 4):  
        line = line + "," + str(mass[m][d])  
    file.write(line + "\n")  
    file.close()
```

Alternative answer

```
def save_day(d):
    with open("week.txt", "a") as file:
        file.write(str(d) + "," + str(mass[0][d]) + ","
                  + str(mass[1][d]) + "," + str(mass[2][d])
                  + "," + str(mass[3][d]) + "\n")
```

Note: Opening the file in write mode ("w") does not gain the first mark, because it deletes the lines already in the file. Accept with open("week.txt", "a") as file: for both the opening and the closing marks. The masses may be written using a loop or one at a time; accept print(line, file=file). mass[d][m] does not gain the second mark, and nor does a trailing comma after the last mass (eg 5,20,45,35,15,), because the format is given. Penalize a missing str() (which raises a TypeError) once only. Exception handling is not required for any mark. Ignore minor syntax errors; penalize a repeated error once only.

(e) Award **[5]** as follows:

Award [1] for an outer loop through the 7 days;
Award [1] for an inner loop through the 4 materials that accesses mass[m][d] for that day;
Award [1] for initializing best (eg best = 0) for each day, inside the outer loop and before the inner loop, together with the greatest mass if it is stored (eg greatest = mass[0][d]);
Award [1] for comparing mass[m][d] with the greatest mass so far (eg greatest or mass[best][d]) and updating best (and the greatest mass, if it is stored separately);
Award [1] for outputting the day number and materials[best] once for each day, after the inner loop;

Example answer

```
def daily_best():
    for d in range (0, 7):
        greatest = -1          # or mass[0][d] with m from 1
        best = 0
        for m in range (0, 4):
            if mass[m][d] > greatest:
                greatest = mass[m][d]
                best = m
        print ("Day " + str(d) + ": " + materials[best])
```

*Note: Accept an answer that loops through the materials in the outer loop, stores the greatest mass and its material for each day in two lists of size 7, and then outputs them; award the equivalent marks. Accept tracking only the index of the best material: best = 0, then if mass[m][d] > mass[best][d]: best = m. A loop that starts at m = 1 is correct only if, for each day, best is initialized to 0 (and the greatest mass, if stored, to mass[0][d]). best may be initialized once, before the outer loop, only if the inner loop starts at m = 0 and either only the index is tracked or the greatest mass starts at -1 for each day. Initializing greatest = mass[0][d] each day but best only once does not gain the third mark. Ignore the format of the output. An answer that uses a function or method that is not permitted gains **[3 max]**. Ignore minor syntax errors; penalize a repeated error once only.*

9. (a) Award [2] as follows:

Award [1] for stating that the new class (eg AppNotifier) inherits from Notifier, so it must override update() / the abstract class forces each subclass to provide update();
Award [1] for explaining that Parcel only calls update() on the objects in self._notifiers and never checks their class (duck typing), so an AppNotifier object can be passed to add_notifier() and is informed by set_status() like any other notifier;

(b) Award [4] as follows:

Award [1] for a class header showing that SmsNotifier inherits from Notifier;
Award [1] for __init__() assigning the phone number to a private attribute (eg self._phone);
Award [1] for a method update(self, code, status) with the same parameters as in Notifier;
Award [1] for outputting the message in the required format only when status is "Out for delivery" or "Delivered";

Accept minor differences in the spacing of the output, eg print ("SMS to", self._phone + ":", code, status).

Example answer

```
class SmsNotifier(Notifier):  
  
    def __init__(self, phone):  
        self._phone = phone  
  
    def update(self, code, status):  
        if status == "Out for delivery" or status == "Delivered":  
            print ("SMS to " + self._phone + ": " + code + " " + status)
```

A class that inherits from Notifier but does not define update() cannot be instantiated (a TypeError is raised), because update() is abstract.

(c) Award [3 max]

Parcel is the subject/publisher and the notifiers (EmailNotifier and SmsNotifier objects) are its observers/subscribers;
a notifier subscribes through add_notifier(), which stores it in self._notifiers;
when the status changes, set_status() notifies every registered observer by calling update();
...each observer decides how to respond, eg SmsNotifier texts only for two statuses;

Award [1 max] if the roles (subject/publisher and observer/subscriber) are only named, without reference to how the classes carry them out.

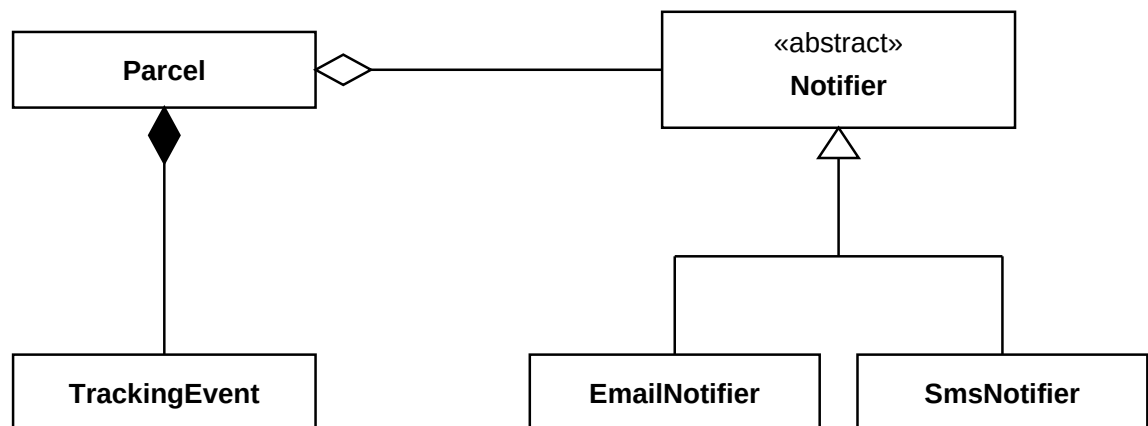
(d) Award [3] as follows:

Award [1] for composition between Parcel and TrackingEvent, shown by a line with a filled (solid) diamond at the Parcel end;
Award [1] for aggregation between Parcel and Notifier, shown by a line with a hollow (unfilled) diamond at the Parcel end;
Award [1] for EmailNotifier and SmsNotifier each joined to Notifier by a line with a hollow triangle at the Notifier end (inheritance);

Accept a dashed line for the inheritance, and «abstract»/{abstract} may be omitted. Accept aggregation lines from Parcel to both EmailNotifier and SmsNotifier instead of to Notifier.

Multiplicities are not required. A diamond at the wrong end does not earn the mark.

Example answer



(e) **Award [3 max]**

the **TrackingEvent** objects are created inside the **Parcel** object (by `set_status()`) and only the parcel refers to them (through `self._history`);
...so they are deleted with the **Parcel** object / cannot exist without it (composition);
the **Notifier** objects are created outside the **Parcel** object (by the driver code) and are passed in through `add_notifier()`;
...so they still exist after the **Parcel** object is deleted and go on informing the customer about their other parcels (aggregation);

Award [2 max] if only one of the two kinds of object is addressed. Award [1 max] if the relationships are only named (composition, aggregation) without describing what happens to the objects. Accept "garbage-collected" for deleted.

(f) **Award [5] as follows:**

Award [1] for handling an empty `self._history` (`len(self._history) == 0`), so that the first status is always recorded;
Award [1] for comparing status with the status of the last event, `self._history[-1].get_status()`, and doing nothing (no event added, no notifier updated) if they are the same;
Award [1] for creating a new **TrackingEvent** object from status and time;
Award [1] for appending the new event to `self._history`;
Award [1] for a loop over `self._notifiers` that calls `update(self._code, status)` on each one;

Award the first mark only if `self._history[-1]` is never accessed when `self._history` is empty (this raises an `IndexError`). Accept `self._history[len(self._history) - 1]` and an early return for a repeated status.

Example answer

```
def set_status(self, status, time):
    if self._history == [] or self._history[-1].get_status() != status:
        self._history.append(TrackingEvent(status, time))
        for n in self._notifiers:
            n.update(self._code, status)
```

Alternative answer

```
def set_status(self, status, time):
    if len(self._history) > 0:
        last = self._history[len(self._history) - 1]
        if last.get_status() == status:
            return # a repeated scan: nothing is done
    event = TrackingEvent(status, time)
    self._history.append(event)
    for i in range(len(self._notifiers)):
        self._notifiers[i].update(self._code, status)
```

Calling update() on each object in self._notifiers without checking its class is dynamic polymorphism: the version in EmailNotifier or in SmsNotifier is selected at run time.

10. (a) **Award [2 max]**

an ADT is defined by the operations that can be carried out on it / by its behaviour (eg add(), discard() and in in the table);
it describes what each operation does, not how it is done: the table does not say how the set is stored (eg in a hash table or a list) / its implementation is hidden;
it specifies the kind of data/values it holds (eg a collection of different medicine names, in no particular order);
it can be implemented in different ways (eg a hash table or a sorted list) without changing the code that uses its operations;
it gives a high-level description of a data structure and its operations, so programmers can use it without knowing its implementation;

Award [1] for each property outlined. Award [1 max] if properties are only named (eg “abstraction”, “encapsulation”) without an outline.

(b) **Award [2] as follows:**

Award [1] for 731 at index 7;
Award [1] for 384 at index 0;

731 % 11 = 5 and slots 5 and 6 are occupied, so 731 is stored in slot 7. 384 % 11 = 10 and slot 10 is occupied, so the search continues from slot 0, which is empty.

Index	0	1	2	3	4	5	6	7	8	9	10
Code	384		189			357	412	731		603	846

(c) **Award [3] as follows:**

Award [1] for a loop that visits each medicine in prescription;
Award [1] for testing whether each medicine is in stock using in / not in;
Award [1] for returning False as soon as a medicine is not in stock, and True only after every medicine has been checked;

Using an operation that is not in the table (eg issubset(), <=, - or &) gains [0]. Do not award the third mark if either set is changed. Award [1 max] if the function tests whether stock is a subset of prescription instead (eg it loops over stock). Accept a Boolean flag that is set to False when a medicine is not in stock and returned after the loop.

Example answer

```
def can_dispense(prescription, stock):  
    for m in prescription:  
        if m not in stock:           # allow (m in stock) == False  
            return False  
    return True
```

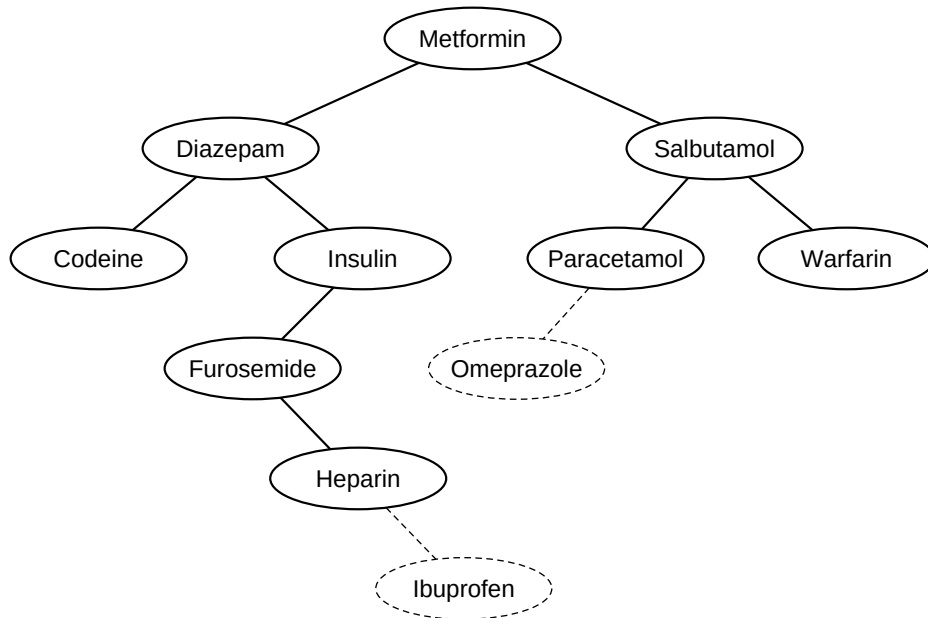
Alternative answer

```
def can_dispense(prescription, stock):  
    found = 0  
    for m in prescription:  
        if m in stock:  
            found = found + 1  
    return found == len(prescription)
```

11. (a) Award **[2]** as follows:

Award **[1]** for Omeprazole as the left child of Paracetamol;

Award **[1]** for Ibuprofen as the right child of Heparin;



The new nodes are shown with a thinner, dashed outline. The rest of the tree must be unchanged.

(b) Award **[2 max]**

Metformin, Diazepam, Codeine, Insulin, Furosemide, Heparin, Salbutamol, Paracetamol, Warfarin;

Award **[1]** for Metformin, Diazepam, Codeine, Insulin, Furosemide, Heparin in this order (the root, then its left subtree) and **[1]** for Salbutamol, Paracetamol, Warfarin in this order after them. An in-order (alphabetical) list scores **[0]**.

(c) Award **[3]** as follows:

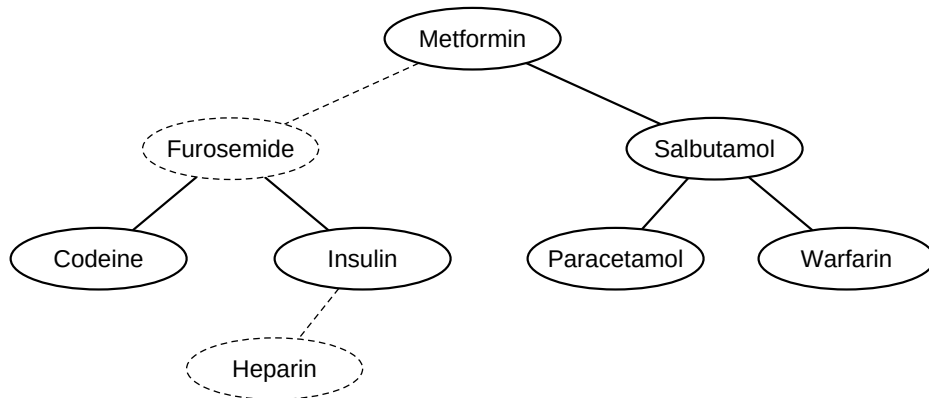
Award **[1]** for searching from the root: Diazepam is compared with Metformin and, as it comes before Metformin, the left child is visited, where Diazepam is found;

Award **[1]** for replacing Diazepam, which has two children, with its in-order successor Furosemide (the leftmost node in its right subtree) / with its in-order predecessor Codeine (the rightmost node in its left subtree);

Award **[1]** for removing the successor from its old position, so that its right child Heparin becomes the left child of Insulin / for removing the predecessor Codeine, which is a leaf, from its old position;

Accept any other method that leaves a valid BST containing all eight of the other names, eg Codeine replaces Diazepam and the subtree rooted at Insulin becomes the right subtree of Codeine; award the second and third marks for it. Accept a correct general description that is applied to this tree (eg “the smallest name in the right subtree of Diazepam”). Award **[1 max]** if the resulting tree is not a binary search tree or loses any name other than Diazepam. Allow the use of diagrams to illustrate any of the above points.

Example answer (in-order successor)



The two nodes that have moved (Furosemide and Heparin) are shown with a thinner, dashed outline.

(d) Award **[2]** as follows:

Award [1] for the four calls in this order: remaining(400, 3), remaining(200, 2), remaining(100, 1), remaining(50, 0);

Award [1] for stating that every call returns 50 / 50.0 (the base case returns 50.0 and each call passes it back unchanged);

Accept 200.0, 100.0 and 50.0 as the first arguments. Do not award the second mark if the values returned are given as 400, 200, 100 and 50: these are the arguments, not the values returned. The answer may be set out as a table, as shown.

Call	Value returned
remaining(400, 3)	50.0
remaining(200.0, 2)	50.0
remaining(100.0, 1)	50.0
remaining(50.0, 0)	50.0

(e) Award **[4]** as follows:

Award [1] for a base case that returns tablets[0] when n is 1;

Award [1] for a recursive call busiest(tablets, n - 1), which leads to the base case;

Award [1] for comparing tablets[n - 1] with the value returned by the recursive call;

Award [1] for returning the larger of the two values;

Award [1 max] if the function is not recursive (eg it uses a loop). Award [2 max] if max() is used. Accept a base case $n == 0$ that returns 0, because the numbers of tablets are never negative. Accept a solution that makes the recursive call twice (eg once in the condition and once in the else branch): it returns the correct value but repeats work. Accept busiest(tablets[1:], n - 1) compared with tablets[0].

Example answer

```
def busiest(tablets, n):  
    if n == 1:  
        return tablets[0]  
    rest = busiest(tablets, n - 1)  
    if tablets[n - 1] > rest:  
        return tablets[n - 1]  
    else:  
        return rest
```

12. (a) Award [5] as follows:

The algorithm will:

- set the prev pointer of new_track to self.current;
- set the next pointer of new_track to the next pointer of self.current / to the node after self.current;
- if self.current is the tail / the next pointer of self.current is None, set self.tail to new_track;
- otherwise, set the prev pointer of the node after self.current to new_track;
- set the next pointer of self.current to new_track, after the node after self.current has been linked to new_track / saved;

Award [3 max] if the next pointer of self.current is changed before the node after self.current has been saved or linked to new_track (the rest of the playlist is lost). Award [4 max] if the tail case is not handled. Allow the use of diagrams to illustrate any of the above points.

Accept the steps in any order that does not lose the node after self.current, eg set the next pointer of new_track first, then the next pointer of self.current, then use the next pointer of new_track to reach the following node.

Algorithm is:

1. Set the prev pointer of new_track to self.current.
2. Set the next pointer of new_track to the next pointer of self.current.
3. If the next pointer of self.current is None (self.current is the tail), set self.tail to new_track.
4. Otherwise, set the prev pointer of the node after self.current to new_track.
5. Set the next pointer of self.current to new_track.

(b) Award [5] as follows:

Award [1] for traversing the list from self.head with a temporary node, comparing each title with title, and stopping at the first match or at the end of the list;

Award [1] for removing the first node: setting self.head to the second node and the prev pointer of the new head to None;

Award [1] for removing the last node: setting self.tail to the previous node and the next pointer of the new tail to None;

Award [1] for removing a node in the middle: setting the next pointer of the previous node to the following node **and** the prev pointer of the following node to the previous node;

Award [1] for outputting a suitable message only when no node matches, without changing the list;

Accept the combined form shown in the alternative answer, which earns the second, third and fourth marks. Removing every matching node does not earn the first mark.

Example answer

```
def remove_track(self, title):
    temp = self.head
    while temp != None and temp.title != title:
        temp = temp.next
    if temp == None:
        print (title, "is not in the playlist")
    elif temp == self.head:
        self.head = temp.next
        self.head.prev = None
    elif temp == self.tail:
        self.tail = temp.prev
        self.tail.next = None
    else:
        temp.prev.next = temp.next
        temp.next.prev = temp.prev
```

Alternative answer

```
def remove_track(self, title):
    temp = self.head
    found = False
    while temp != None and found == False:
        if temp.title == title:
            found = True
        else:
            temp = temp.next
    if found == False:
        print (title, "is not in the playlist")
    else:
        if temp.prev != None:
            temp.prev.next = temp.next
        else:
            self.head = temp.next
        if temp.next != None:
            temp.next.prev = temp.prev
        else:
            self.tail = temp.prev
```
