COURSEWORK SUBMISSION FORM

TO THE STUDENT: Please complete the boxes and submit the form together with your essay/coursework.

Name (s)	**ID Number (s)**	**Date Handed In**
Changyu Li	20513997	15/12/2022
Module Title Introduction to Algorithms		**Module Convenor** Chenfei Zhang
Coursework Title Coursework Assignment, Semester 1, 2022-2023		**Module Code** CELEN086

Compulsory

I/We have read the section relating to plagiarism in the University's regulations and confirm that the attached submission is my/our own work. I/We understand that 5 marks per working day will be deducted from the final mark for lateness, unless an extension form has been authorised and is attached, e.g. a mark of 42 minus 5 marks changes to 37.
The deadline for handing in coursework is 4pm.

Signature (s)

……Changyu Li……

Optional

I/We give permission for the attached piece of work and any electronic version of this paper that is also submitted to be used for future research and training purposes.

Signature (s)

……Changyu Li……

OFFICE USE ONLY

Copies received 1 or 2

Date & Time received **Late: YES / NO**	**Penalty**	**Extenuating Circumstances** **YES / NO**	**Evidence Attached** **YES / NO**

Question 1

(a)

Arg 1 = n

Arg 2 = n-1

Arg 3 = n

Arg 4 = m-1

(b)

Algorithm: sqrt(n)

Requires: One positive integer n;

Returns: One integer = $\lfloor\sqrt{n}\rfloor$

```
1.  return sqrtHelper(n, n-1)
```

Algorithm: sqrtHelper(n,m)

Requires: Two positive integers n, m;

Returns: One integer = $\lfloor\sqrt{n}\rfloor$

```
1. if m*m > n
2.      return sqrtHelper(n, m-1)
3. else
4.      return m
5. endif
```

I think the two algorithms have the same efficiency. Since the Line 1 is reversed, the if-else is also reversed. So it could not bring any change to the efficiency.

For example: sqrt(5)

For algorithm in (a):

```
return sqrtHelper(5, 4)
        4*4 <= 5     False
        return sqrtHelper(5, 3)
                3*3 <= 5     False
                return sqrtHelper(5, 2)
                        2*2 <= 5     True
```

return 2

return 2

For algorithm in (b):

return sqrtHelper(5, 4)

4*4 > 5 True

return sqrtHelper(5, 3)

3*3 > 5 True

return sqrtHelper(5, 2)

2*2 > 5 False

return 2

return 2

(c)

Algorithm: isPrime(n)

Requires: One positive integer n;

Returns: True if n is a prime number

1. return primeHelper(n,sqrt(n))

Algorithm: primeHelper(a, b)

Requires: Two positive integers a, b;

Returns: True if a is a prime number

1. if b == 1
2. return True;
3. else if a % b == 0
4. return False
5. else
6. return primeHelper(a, b-1)
7. endif

isPrime(41):

return primeHelper(41,6)

b == 1 False

a % b == 0 False

return primeHelper(41,5)

```
                b == 1        False
                a % b == 0          False
                        return primeHelper(41,4)
                        b == 1        False
                        a % b == 0          False
                                return primeHelper(41,3)
                                b == 1        False
                                a % b == 0          False
                                        return primeHelper(41,2)
                                        b == 1        False
                                        a % b == 0          False
                                                return primeHelper(41,1)
                                                b == 1 True
                                                        return True
        return True
```

Question 2

(a)

Algorithm: delete(x, L)

Requires: One existing element x, list L

Returns: List L without x

1. return deleteHelper(x, [], L)

Algorithm: deleteHelper(x, L1,L2)

Requires: One existing element x, list L1, L2

Returns: List L without x

1. if x == head(L2)
2. return cons(L1, tail(L2))
3. else
4. return deleteHelper(x, cons(L1, head(L2)), tail(L2))
5. endif

(b)

Algorithm: selectionSort(L)

Requires: list L

Returns: Sorted list L

1. return sortHelper(L, [])

Algorithm: sortHelper(L1, L2)

Requires: list L1, L2

Returns: Sorted list L

1. if tail(L1) == []
2. return cons (head(L1), L2)
3. else
4. L2 = cons (maxlist(L1), L2)
5. return sortHelper(delete(maxlist(L1)), L2)
6. endif

(c)

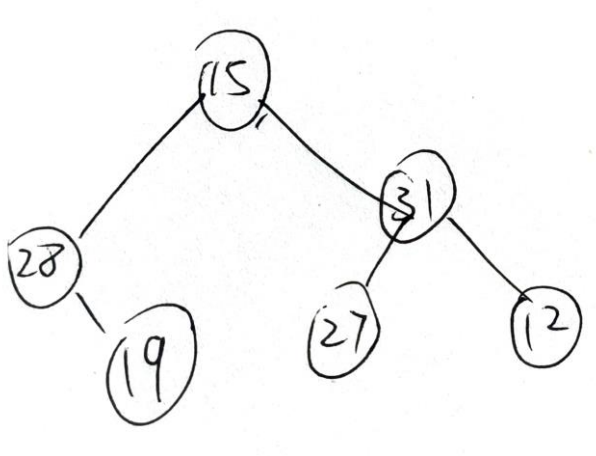
L = [4, 7, 6, 3, 5, 1]

```
return sortHelper(L, [])
        tail(L1) == []        False
        L2 = cons (7, [])
        return sortHelper([4, 6, 3, 5, 1], [7])
                tail(L1) == []        False
                L2 = cons (6, [7])
                return sortHelper([4, 3, 5, 1], [6, 7])
                        tail(L1) == []        False
                        L2 = cons (5, [6, 7])
                        return sortHelper([4, 3, 1], [5, 6, 7])
                                tail(L1) == []        False
                                L2 = cons (4, [5, 6, 7])
                                return sortHelper([3, 1], [4, 5, 6, 7])
                                        tail(L1) == []        False
                                        L2 = cons (3, [4, 5, 6, 7])
                                        return sortHelper([1], [3, 4, 5, 6, 7])
                                                tail(L1) == []        True
                                                return cons (1, [3, 4, 5, 6, 7])
```

return [1, 3, 4, 5, 6, 7]

Question 3

(a)



[19, 28, 27, 12, 31, 15]

(b)

Algorithm: average (T)

Requires: a binary tree (T)

Returns: the average of all values stored in the tree

1. return (averageList(1, traversal(T)))

Algorithm: traversal(T)

Requires: a binary tree (T), an empty list L

Returns: a list L

1. if isLeaf(T)
2. return (cons(root(T), L)
3. else
4. cons(traversal(left(T)), L)
5. cons(traversal(right(T)), L)
6. cons(root(T)), L)
7. return L
8. endif

Algorithm: averageList (m, L)

Requires: a list L, length of list m

Returns: average of list

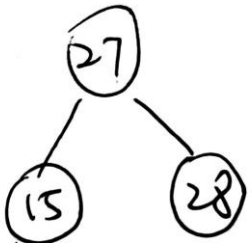
1. if tail(L) == []
2. return (head(L) / m)
3. else
4. return (averageList (m+1, cons(head(L) + head(tail(L)), tail(tail(L)))))
5. endif

(c)

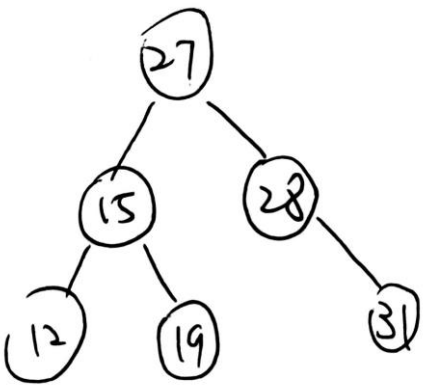
1. sort the list: [12, 15, 19, 27, 28, 31]

2. Find the middle element, and store it in root node. (27)

3. For left/right sub-lists, find the middle elements and store them in the left/right child nodes on next level.



6. repeat this process until all elements are stored in the tree.



Algorithm: count (x, T)

Requires: a binary tree T

Returns: number of elements greater than x

1. L = traversal(T)
2. return countHelper(0, L)

Algorithm: countHelper(x, L)

Requires: a list L, x

Returns: number of elements greater than x

```
1. if header(L) > x
2.     x = x+1
3. Endif
4. if tail(L) ==[]
5.     return x
6. else
7.     return countHelper(x, tail(L))
8. endif
```

Tracing:

```
return countHelper(0, [19, 28, 27, 12, 31, 15])
        19 > 18       true
                X = 1
        tail(L) ==[]  false
                return countHelper(1, [28, 27, 12, 31, 15])
                28 > 18       true
                        X = 2
                tail(L) ==[]  false
                        return countHelper(2, [27, 12, 31, 15])
                        27 > 18       true
                                X = 3
                        tail(L) ==[]  false
                                return countHelper(3, [12, 31, 15])
                                12 > 18       false
                                        X = 3
                                tail(L) ==[]  false
                                        return countHelper(3, [31, 15])
                                        12 > 18       true
                                                X = 4
                                        tail(L) ==[]  false
                                                return countHelper(4, [15])
                                                15 > 18 false
```

X = 4
tail(L) ==[] true
return 4
return 4

Question 4

(a)

Algorithm: bin2dec(L)
Requires: a list L
Returns: value in decimal number system

1. return decHelper(length(L) - 1, L, 0)

Algorithm: decHelper(x, L, n)
Requires:x, n, a list L
Returns: value n in decimal number system

1. if x == 0
2. return n + head(L)
3. else
4. return decHelper(x-1, tail(L), n + head(L)*(2^n))
5. endif

(b)

return decHelper(5, [1, 1, 0, 1, 0, 1], 0)
X == 0 false
return decHelper(4, [1, 0, 1, 0, 1], 32)
X == 0 false
return decHelper(3, [0, 1, 0, 1], 48)
X == 0 false
return decHelper(2, [1, 0, 1], 48)
X == 0 false
return decHelper(1, [0, 1], 52)
X == 0 false
return decHelper(0, [1], 52)

X == 0 true

return 53

return 53

(c)

Algorithm: dec2bin(n)

Requires: a number n

Returns: a list L

1. let a = largestPower(n, 1)
2. return reverse(binHelper(a, [], n))

Algorithm: binHelper(a, L, n)

Requires: a number n, a, L

Returns: a list L

1. if a == 0 && n == 1
2. return cons(1, L)
3. else if a == 0 && n == 0
4. return cons(0, L)
5. else if 2 ^ a <= n
6. return binHelper(a-1, cons(1, L), n – 2^(a–1))
7. else
8. return binHelper(a-1, cons(0, L), n)
9. endif

Algorithm: largestPower(n, a)

Requires: numbers n, a

Returns: largest Power

1. if 2 ^ a < n
2. return largestPower(n, a+1)
3. else
4. return a – 1
5. endif

Tracing:

```
a = 4
return reverse(binHelper(4, [], 19))
        a == 0 && n == 1          false
        a == 0 && n == 0          false
        2 ^ 4 <= 19         true
                return binHelper(3, [1], 3)
                a == 0 && n == 1          false
                a == 0 && n == 0          false
                2 ^ 3 <= 3          false
                        return binHelper(2, [0, 1], 3)
                        a == 0 && n == 1          false
                        a == 0 && n == 0          false
                        2 ^ 2 <= 3          false
                                return binHelper(1, [0, 0, 1], 3)
                                a == 0 && n == 1          false
                                a == 0 && n == 0          false
                                2 ^ 1 <= 3          true
                                        return binHelper(0, [1, 0, 0, 1], 1)
                                        a == 0 && n == 1          true
                                                return [1, 1, 0, 0, 1]
return [1, 0, 0, 1, 1]
```