The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

# COURSEWORK SUBMISSION FORM

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

| **Name** | **ID Number** | **Date Submitted** |
|---|---|---|
| Guosheng SU | 20411905 | **2/12/2021** |
| **Module Title** | | **Module Convenor** |
| Introduction to Algorithms | | Aidin JALILZADEH |
| **Coursework Title** | | **Module Code** |
| CW_20411905_N086 | | 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.**

**Signature (s)**
**Guosheng SU**

**.....**

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)
**Guosheng SU**

.....

---

## OFFICE USE ONLY

**Copies received 1 or 2**

| **Date & Time Received** | **Penalty** | **Extenuating Circumstances** | **Evidence Attached** |
|---|---|---|---|
| **Late: YES / NO** | | **YES / NO** | **YES / NO** |

**Question 1**

(a)

**Algorithm:** addN(x,n)

**Requires:** a number **x** and a positive integer **n**

**Return:** n*x

```
1. if n == 1 then
2.    return x
3. else
4.    return x + addN(x,n-1)
5. endif
```

(b)

**Algorithm:** power(x,n)

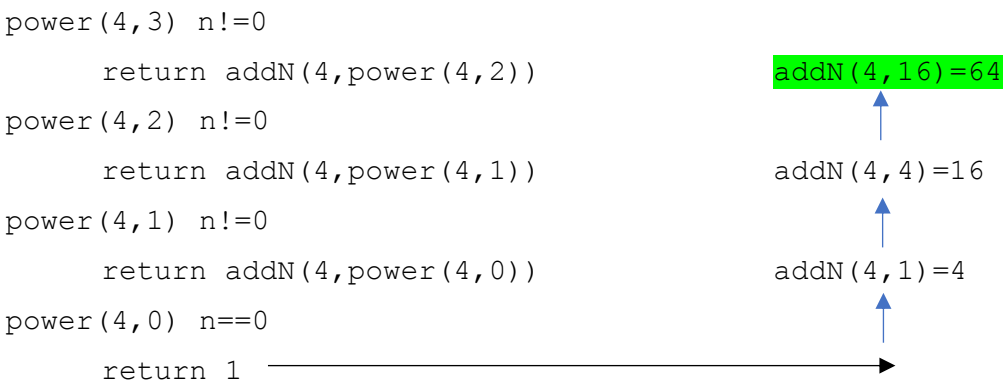
**Requires:** a positive integer **x** and a positive integer **n**

**Return:** the value of x^n

```
1. if n == 0 then
2.    return 1
3. else
4.    return addN(x,power(x,n-1))
5. endif
```

(c)

*trace* ***power(x,n)*** *for* ***power(4,3)***



**Question 2**

(a)

**Algorithm:** concat(L1,L2)
**Requires:** 2 lists L1 L2
**Return:** a new list with L1 attached to the head of L2

```
1. if isEmpty(L1) then
2.    return L2
3. else
4.    return cons(head(L1),concat(tail(L1),L2))
5. endif
```

(b)

**Algorithm:** subset(L1,L2)
**Requires:** 2 lists L1 L2
**Return:** TRUE or FALSE

```
1. if length(L1) > length(L2) then
2.    return False
3. elseif isEmpty(L1) then
4.    return True
5. elseif linSearch(head(L1),L2) == false then
6.    return False
7. else
8.    return subset(tail(L1),L2)
9. endif
```

**Algorithm:** linSearch(x,list)
**Requires:** a positive integer x and a list
**Return:** TRUE or FALSE

```
1. if x == head(list) then
2.    return True
3. elseif isEmpty(list) then
4.    return False
5. else
6.    return linSearch(x,tail(list))
7. endif
```

(c)

*trace* ***subset(L1,L2)*** *for* ***subset([1,5,2],[4,5,1,0,2,9])***

```
subset([1,5,2],[4,5,1,0,2,9])
      length(L1) > length(L2)??        NO!
      isEmpty(L1)??                    NO!
      linSearch(1,[4,5,1,0,2,9])??     TRUE!
      return subset([5,2],[4,5,1,0,2,9])
subset([5,2],[4,5,1,0,2,9])
      length(L1) > length(L2)??        NO!
      isEmpty(L1)??                    NO!
      linSearch(5,[4,5,1,0,2,9])??     TRUE!
      return subset([2],[4,5,1,0,2,9])
subset([2],[4,5,1,0,2,9])
      length(L1) > length(L2)??        NO!
      isEmpty(L1)??                    NO!
      linSearch(2,[4,5,1,0,2,9])??     TRUE!
      return subset([],[4,5,1,0,2,9])
subset([],[4,5,1,0,2,9])
      length(L1) > length(L2)??        NO!
      isEmpty(L1)??                    YES!
      return TRUE
```

**Question 3**

(a)

**Algorithm:** level(x,binT)

**Requires:** a binary tree and a node value x

**Return:** the generation level to which the node x belongs

```
1. if x == root(binT) then
2.    return 0
3. elseif searchBT(x,left(binT)) then
4.    return 1 + level(x,left(binT))
5. else
6.    return 1 + level(x,right(binT))
7. endif
```

**Algorithm:** searchBT(x,binT)

**Requires:** a binary tree and a node value x

**Return:** true or false

```
1. if isLeaf(binT) then
2.    return false
3. elseif x == root(binT) then
4.    return true
5. elseif isLeaf(left(binT)) && isLeaf(Right(binT)) then
6.    return false
7. else
8.    return searchBT(x,left(binT)) || searchBT(x,right(binT))
9. endif
```

(b)

*trace* ***level(x,binT)*** *for* ***level(45,T)***

```
level(45,T)                                                    level = 2
      45 == 50??                                NO!
      searchBT(45,left(T))??                    TRUE!
      return 1 + level(45,left(T))
level (45,left(T))                                             level = 1
      45 == 40??                                NO!
      searchBT(45,left(left(T)))?               FALSE!
      searchBT(45,right(left(T)))?              TRUE!
      return 1 + level(45,right(left(T)))
level(45,right(left(T)))                                       level = 0
      45 == 45??                                YES!
      return 0 ------------------------------------------------------>
```

(c)

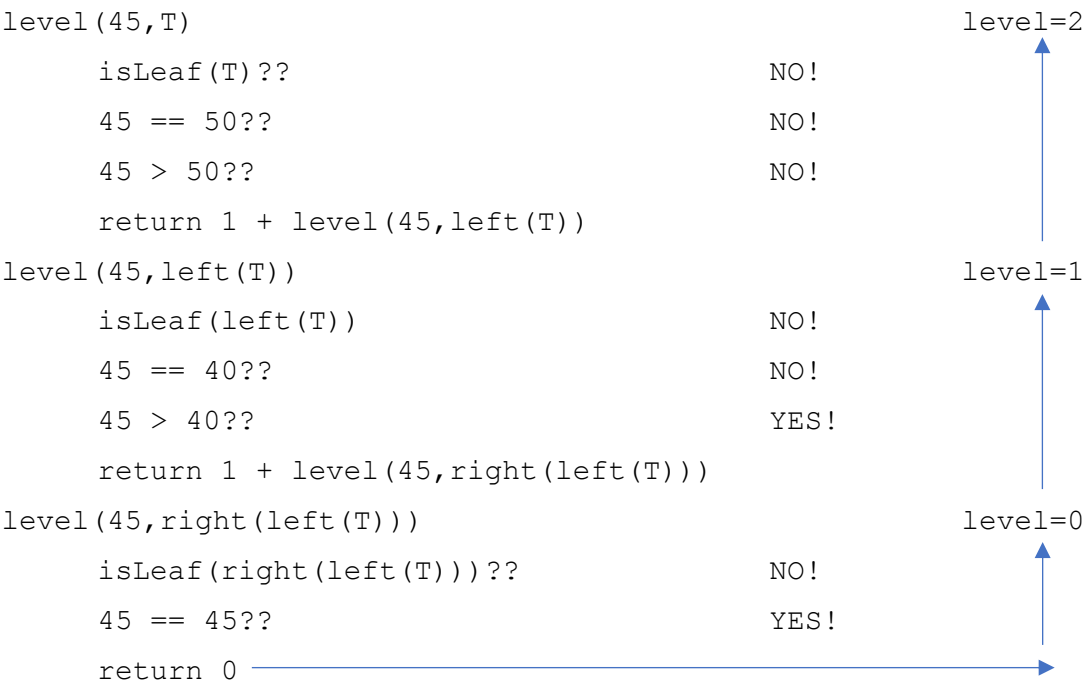
**Algorithm:** level(x,BST)

**Requires:** a BST and a node value x

**Return:** the generation level to which the node x belongs

```
1. if isLeaf(BST) then
2.    return 0
3. elseif x == root(BST) then
4.    return 0
5. elseif x > root(BST) then
6.    return 1 + level(x,right(BST))
7. else
8.    return 1 + level(x,left(BST))
9. endif
```

*trace* ***level(x,BST)*** *for* ***level(45,T)***




**level(x,BST)** is faster than **level(x,T)**

**level(x,BST)** is **O(lgn)**

**level(x,T)** is **O(n)**

**Question 4**

(a)

**Algorithm:** partition(L)
**Requires:** a list
**Return:** LP, PP and RP

```
1. if isEmpty(L) then
2.    return [], [], []
3. else
4.    return pHelper(head(L),tail(L),[],[])
5. endif
```

**Algorithm:** pHelper(x,list,L1,L2)
**Requires:** a positive integer x and 3 lists
**Return:** 3 new lists

```
1. if isEmpty(list) then
2.    return L1, cons(x,nil), L2
3. elseif x < head(list) then
4.    return pHelper(x,tail(list),L1,cons(head(list),L2))
5. else
6.    return pHelper(x,tail(list),cons(head(list),L1),L2)
7. endif
```

(b)

**Algorithm:** quicksort(list)
**Requires:** an unsorted list
**Return:** a sorted list of numbers

```
1. if isEmpty(list) || isEmpty(tail(list)) then
2.    return list
3. else
4.    let (L1,L2,L3) = partition(list)
5.    let S1 = quicksort(L1)
6.    let S2 = quicksort(L3)
7.    return concat(S1,concat(L2,S2))
8. endif
```

**Algorithm:** concat(L1,L2)
**Requires:** 2 lists L1 L2
**Return:** a new list

```
1. if isEmpty(L1) then
2.    return L2
3. else
4.    return cons(head(L1),concat(tail(L1),L2))
5. endif
```

(c)

*Trace* ***partition(L)*** *for* ***[10,20,40,50,45]***

```
partition([10,20,40,50,45])
      isEmpty([10,20,40,50,45])??                    NO!
      return pHelper(10,[20,40,50,45],[],[])

pHelper(10,[20,40,50,45],[],[])
      isEmpty([20,40,50,45])??                       NO!
      10 < 20??                                      YES!
      return pHelper(10,[40,50,45],[],[20])
pHelper(10,[40,50,45],[],[20])
      isEmpty([40,50,45])??                          NO!
      10 < 40??                                      YES!
      return pHelper(10,[50,45],[],[40,20])
pHelper(10,[50,45],[],[40,20])
      isEmpty([50,45])??                             NO!
      10 < 50??                                      YES!
      return pHelper(10,[45],[],[50,40,20])
pHelper(10,[45],[],[50,40,20])
      isEmpty([45])??                                NO!
      10 < 45??                                      YES!
      return pHelper(10,[],[],[45,50,40,20])
pHelper(10,[],[],[45,50,40,20])
      isEmpty([])??                                  YES!
      return [], [10], [45,50,40,20]
```

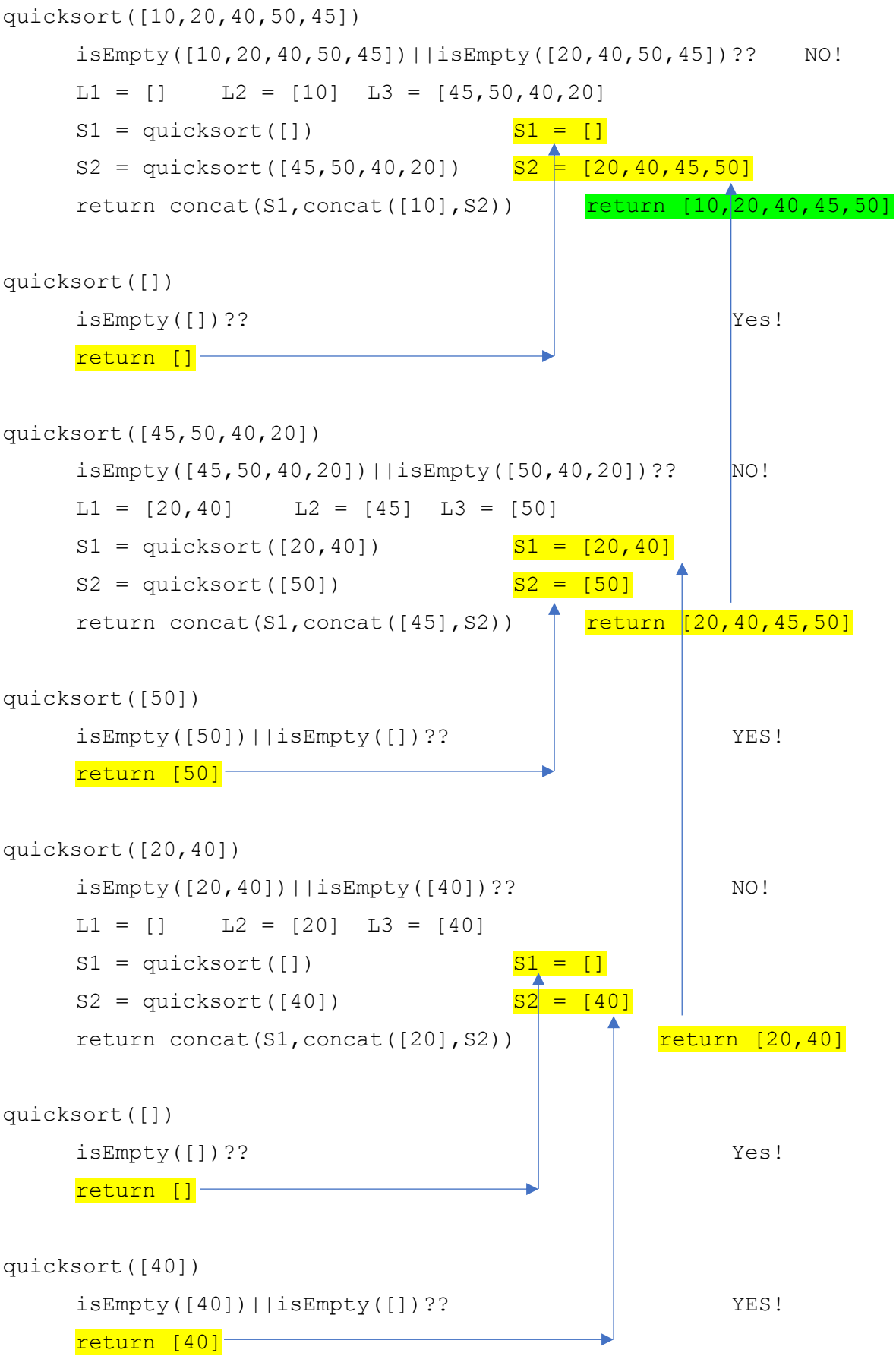
The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

*Trace* ***partition(L)*** *for* ***[10,9,12,8,15]***

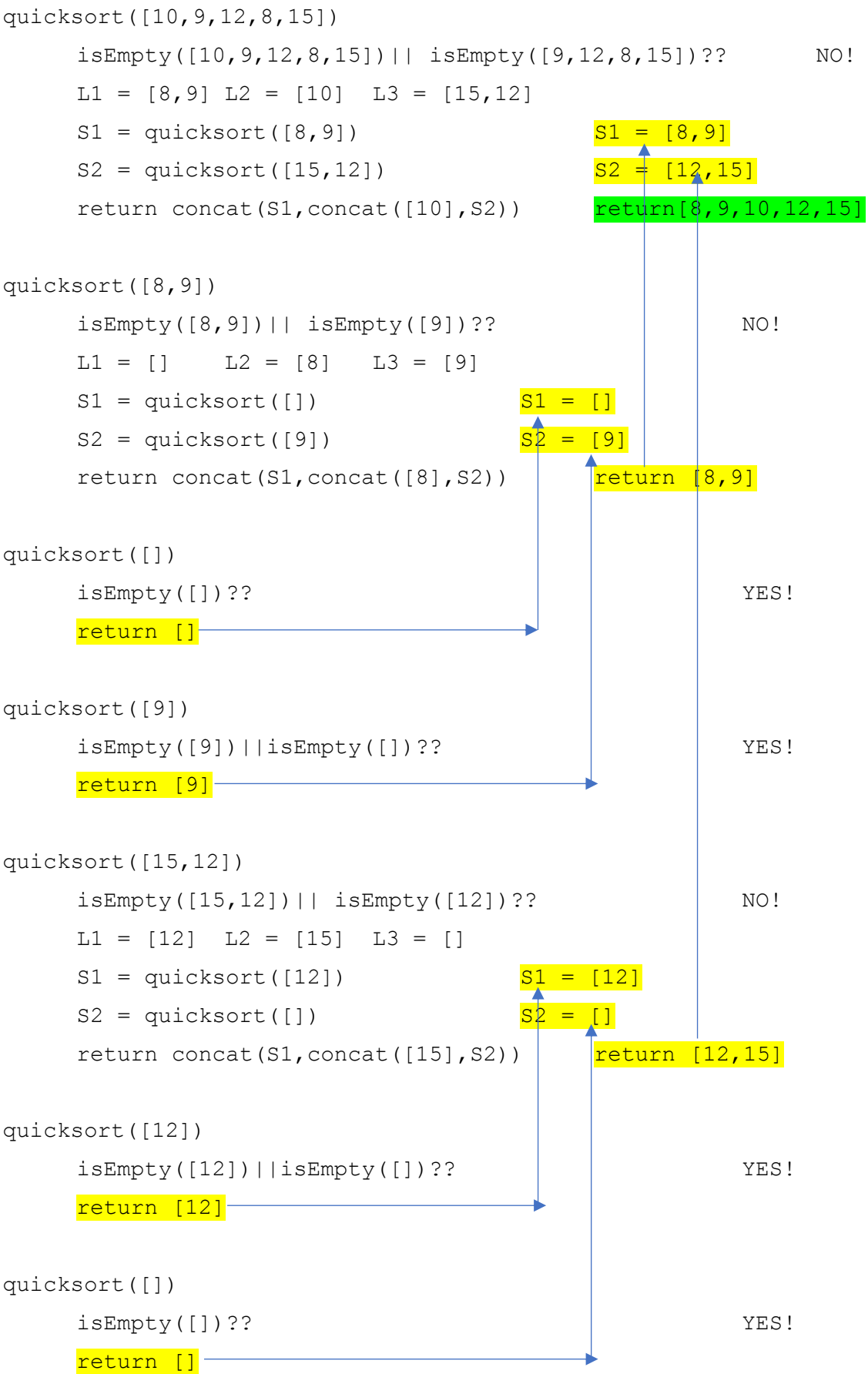
```
partition([10,9,12,8,15])
      isEmpty([10,9,12,8,15])??                    NO!
      return pHelper(10,[9,12,8,15],[],[])

pHelper(10,[9,12,8,15],[],[])
      isEmpty([9,12,8,15])??                       NO!
      10 < 9??                                     NO!
      return pHelper(10,[12,8,15],[9],[])
pHelper(10,[12,8,15],[9],[])
      isEmpty([12,8,15])??                         NO!
      10 < 12??                                    YES!
      return pHelper(10,[8,15],[9],[12])
pHelper(10,[8,15],[9],[12])
      isEmpty([8,15])??                            NO!
      10 < 8??                                     NO!
      return pHelper(10,[15],[8,9],[12])
pHelper(10,[15],[8,9],[12])
      isEmpty([15])??                              NO!
      10 < 15??                                    YES!
      return pHelper(10,[],[8,9],[15,12])
pHelper(10,[],[8,9],[15,12])
      isEmpty([])??                                YES!
      return [8,9], [10], [15,12]
```

*Trace* ***quicksort(list)*** *for* ***[10,20,40,50,45]***



*Trace* ***quicksort(list)*** *for* ***[10,9,12,8,15]***



(d)

As for List1, the head of List1 is the smallest number in List1.
This leads to longer length of S2.
And longer length of S2 leads to more operations for partition, thus leading to more operations for L1.

As for List2, the head of List2 is the middle number in List2.
This leads to the same length of S1 and S2.
The length of S1(S2) is nearly half of the length of List2.
And shorter length of S1 and S2 leads to less operations for partition, thus leading to less operations for L2.