Answer for SET4

(1)

Algorithm: find(x,L)

Requires: a list and an element

Return: the position of that element in the list

```
1. if linsearch(x,L) == false then
2.     return 0
3. elseif x == head(list) then
4.     return 1
5. else
6.     return 1 + find(x,tail(L))
7. 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
```

(2)

Algorithm: mergeSort(list)

Requires: a list

Returns: a sorted list

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

trace for [10,11,12,0,3,4,2,1,5]

```
mergesort([10,11,12,0,3,4,2,1,5])
    isEmpty(list)||isEmpty(tail(list))??            NO!
    L1 = [3,4,2,1,5]
    L2 = [0,12,11,10]
    let S1 = mergesort([3,4,2,1,5])
    let S2 = mergesort([0,12,11,10])
    return [0,1,2,3,4,5,10,11,12]
mergesort([3,4,2,1,5])
    isEmpty(list)||isEmpty(tail(list))??            NO!
    L1 = [2,1,5]
    L2 = [4,3]
    let S1 = mergesort([2,1,5])
    let S2 = mergesort([4,3])
    return [1,2,3,4,5]
mergesort([2,1,5])
```

```
    isEmpty(list)||isEmpty(tail(list))??            NO!
    L1 = [1,5]
    L2 = [2]
    let S1 = mergesort([1,5])
    let S2 = mergesort([2]) = [2]
    return [1,2,5]
mergesort([1,5])
    isEmpty(list)||isEmpty(tail(list))??            NO!
    L1 = [5]
    L2 = [1]
    let S1 = mergesort([5]) = [5]
    let S2 = mergesort([1]) = [1]
    return [1,5]
mergesort([4,3])
    isEmpty(list)||isEmpty(tail(list))??            NO!
    L1 = [3]
    L2 = [4]
    let S1 = mergesort([3]) = [3]
    let S2 = mergesort([4]) = [4]
    return [3,4]
mergesort([0,12,11,10])
    isEmpty(list)||isEmpty(tail(list))??            NO!
    L1 = [11,10]
    L2 = [12,0]
    let S1 = mergesort([11,10])
    let S2 = mergesort([12,0])
    return [0,10,11,12]
mergesort([11,10])
    isEmpty(list)||isEmpty(tail(list))??            NO!
    L1 = [10]
```

```
    L2 = [11]
    let S1 = mergesort([10]) = [10]
    let S2 = mergesort([11]) = [11]
    return [10,11]
mergesort([12,0])
    isEmpty(list)||isEmpty(tail(list))??            NO!
    L1 = [0]
    L2 = [12]
    let S1 = mergesort([0]) = [0]
    let S2 = mergesort([12]) = [12]
    return [0,12]
```

(3)

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
```

(4)

插入排序 O(n^2)

Algorithm: insertionSort(list)

Requires: a list of numbers

Return: a ***sorted*** list of numbers in ascending order

```
1. return sortHelper(list,[])
```

Algorithm: sortHelper(leftList,rightList)

Requires: two lists

Returns: rightList, sorted version of leftList

```
1. if isEmpty(leftlist) then
2.     return rightlist
3. else
4.     return
   sortHelper(tail(leftlist),insert(head(leftlist),rightlist))
5. endif
```

(5)

Algorithm: plus(x,binT)

Requires: a BT and a value x

Return: a new BT with updated node values

```
1. if isLeaf(binT) then
2.     return binT
3. elseif isLeaf(left(binT)) && isLeaf(right(binT)) then
4.     return node(leaf,root(binT)+x,leaf)
5. else
6.     return
   node(plus(x,left(binT)),root(binT)+x,plus(x,right(binT)))
7. endif
```

(6)

Algorithm: sumNodes(binT)

Requires: a BT

Return: the sum of all the node values

```
1. if isLeaf(binT) then
2.     return 0
3. elseif isLeaf(left(binT)) && isLeaf(right(binT)) then
4.     return root(binT)
5. else
6.     return     root(binT)     +     sumNodes(left(binT))     +
   sumNodes(right(binT))
7. endif
```