Assignment 6

R-4.14

Bubble sort is stable.

Heap sort is not a stable sort because of upHeaping.

Insertion sort is stable because the value swaps one by one and it doesn't cross once it finds the equal value

Merge sort is not always stable but it can be stable if the first part of the values are always merged to left part and remaining to right part and if it continues the process.

Quicksort is not stable at all because the pivot is chosen randomly and value swaps from both sides.

R-4.16

The bucket sort uses O(n+N) space. Bucket sort moves items to different buckets to get them sorted. As a result, it's not in-place.

C-4.13

Algorithm IsIdentical(A, B, x)	
Input: A and B are sequences of n integers	
Output: true if A and B have same elements, false otherwise	
HeapSort(A)	O(n log n)
HeapSort(B)	O(n log n)
For i $\leftarrow$0 to i = A.size()-1 do	O(n)
If A.elemAtRank(i) $\neg$= B.elemAtRank(i) then	O(n)
Return false	O(1)
Return true	O(1)

Total running time is O(n log n)

R-5.4

a) a=2, b=2, f(n) = log n

$\log_b a = 1$

case 1: $\log n \leq n^{1 - \varepsilon}$

True for $\varepsilon = 0.5$

T(n) is Θ(n)

b) $a=8$, $b=2$, $f(n) = n^2$

$\log_b a = 3$

case 1: $n^2 \leq n^{3 - \varepsilon}$

True for $\varepsilon = 1$

$T(n)$ is $\Theta(n^3)$

c) $a=16$, $b=2$, $f(n)= (n \log n)^4$

$\log_b a = 4$

Case 1: $(n \log n)^4 \leq n^{4 - \varepsilon}$

Not true for $\varepsilon > 0$

Case 2: $(n \log n)^4 = n^4 \log^k n$

True for $k = 4$

$T(n)$ is $\Theta(n^4 \log^5 n)$

d) $a=7$, $b=3$, $f(n) = n$

$\log_b a = 1.7712$

Case 1: $n \leq n^{1.7712 - \varepsilon}$

True for $\varepsilon = 0.7712$

$T(n)$ is $\Theta(n^{1.7712})$

e) $a=9$, $b=3$, $f(n)= n^3 \log n$

$\log_b a = 2$

Case 1: $n^3 \log n \leq n^{2 - \varepsilon}$

Not true for $\varepsilon > 0$

Case 2: $n^3 \log n = n^2 \log^k n$

Not true

Case 3: $n^3 \log n \geq n^{2 + \varepsilon}$

True for $\varepsilon = 1$

$9\,(n/3)^3 \log (n/3) \leq \delta\, n^3 \log n$

$1/3\; n^3 (\log n - \log 3) \leq \delta\, n^3 \log n$

$\delta = 1/3$, $T(n)$ is $\Theta(n^3 \log n)$

Assignment 7

R-2.19

h(12)= 7　　h(44)= 5　　h(13)=9　　h(88)= 5　　h(23)= 7　　h(94)= 6　　h(11)=6

h(39)= 6　　h(20)=1　　h(16)= 4　　h(5)=4

0	1	2	3	4	5	6	7	8	9	10
Ø		Ø	Ø					Ø		Ø
	20			16	44	49	12		13	
				5	88	39	23			
					11					

R-2.20

0	1	2	3	4	5	6	7	8	9	10
11	39	20	5	16	44	88	12	23	13	94

R-2.21

0	1	2	3	4	5	6	7	8	9	10
	20	16	11	39	44	88	12	23	13	94

R-2.22

$h'(12)= 2$　　$h'(44)= 2$　　$h'(13)= 2$　　$h'(88)= 2$　　$h'(23)= 2$　　$h'(94)= 2$

$h'(11)= 3$　　$h'(39)= 3$　　$h'(20)=1$　　$h'(16)=5$　　$h'(5)= 2$

0	1	2	3	4	5	6	7	8	9	10
11	23	20	16	39	44	94	12	88	13	5

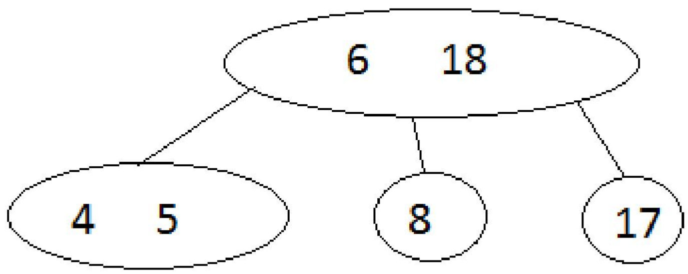
Assignment 8

R-3.8

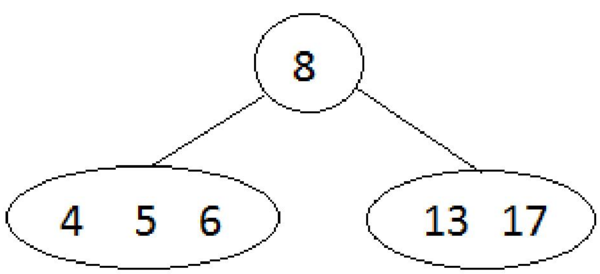
No, the tree in the figure is not a (2,4) tree, because all external nodes don’t have the same depth

R-3.10

1) 5, 8 , 13, 17, 4, 6



2) 13, 4, 8, 5, 6, 17



In conclusion, the (2,4) tree structure changes with the order in which the items are inserted.

C-4.11

Algorithm getWinner(S, C)

```
    Input sequence S containing all the votes
    Output the winner Id
    H ← create new hashtable                          1
    Foreach vote Є C do                               k
        H.insertItem(vote,0)                          k
    Foreach vote Є S do                               n
        count ← H.removeElement(vote)                 n
        count ← count +1                              n
        H.insertItem(vote,count)                      n
    winnerId ← null                                   1
    maxVotes ← 0                                      1
    foreach item(c, count) Є H                        k
        if count > maxVotes then                      k
            maxVotes ← count                          k
            winnerId ← c                              k
    return winnerId                                   1
```

Total running time is O(n)

C-4-22

Algorithm findPair(A, B, k)

Input sequence A containing integers, sequence B containing integers, integer value k

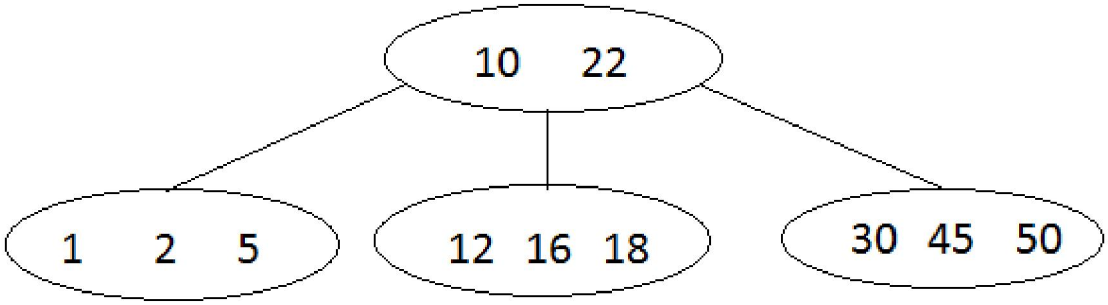
Output Boolean value indicating if there a pair (a,b) which sums to k

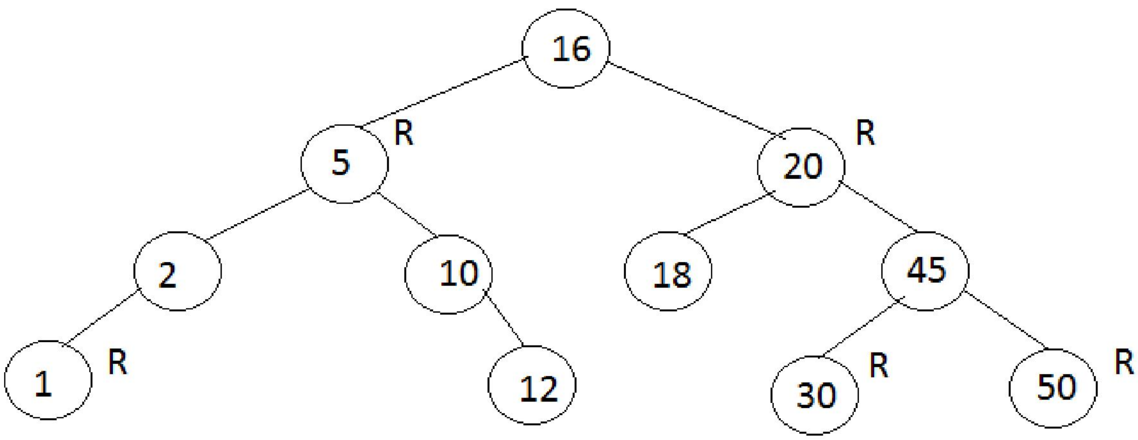
H ← create new hashtable	1
Foreach v Є B do	n
H.insertItem(v, v)	n
Foreach a Є A do	n
b ← H.findElement(v)	n
if b ¬ = No_Such_Key then	n
return true	n
return false	1

Total running time is O(n)

Assignment 9

R-3.11





R-3.14

- **a)** False, because the root node can't be red
- **b)** True
- **c)** True, there is only one unique (2,4) associated with a red-black tree
- **d)** False, a single (2,4) tree could have different red-black tree representations

C-3.10

```
Algorithm findAllInRange(k1,k2)
        S ← new sequence
        V ← T.root()
        findAllInRange(k1, k2, v, T, S)
        return S.elements()
Algorithm findAllInRange(k1,k2, v, T, S)
        If T.IsExternal(v) then
                Return
        Else
          If key(v) > k1 then
                findAllInRange(k1, k2, T.leftChild(v), T, S)
          if key(v) ≥ k1 ∧ key(v) ≤ k2 then
                S.insertLast(key(v))
          If key(v) < k2 then
                findAllInRange(k1, k2, T.rightChild(v), T, S)
```

Assignment 10

R-3.19

Algorithm removeElement(e)

Input e element to remove

Output out element deleted or No_such_element

P ← get the least node in the highest list

y ← after(p)

While e ¬= y.element ∧ y ¬= null do

    While e> y do

        y ← after (y)

    if e ¬ = y then

        y ← down(left(y))

if e=y then

    tmp ← null

    while down(y) ¬ = null do

        tmp ← down(y)

        removeNode(y)

    return tmp

else

    return No_Such_Element

C-4.16

Algorithm containsDuplicate(S)

Input sequence S contains a list of integers

Output Boolean indicating whether there is a duplicate integer in S

H ← create new hashtable

Foreach v ∈ S do

existingElement ← H.removeElement(v)

if existingElement ¬ = No_Such_Key then

return true

else

H.insertItem(v,v)

Return false

C-4.18

Algorithm inPlacePartition(S, lo, hi)

Input Sequence S and ranks lo and hi, lo, hi

Output the pivot is now stored at its sorted rank

p ← a random integer between lo and hi

S.swapElements(S.atRank(lo), S.atRank(p))

pivot ← S.elemAtRank(lo)

for i← 1 to S.size() do

if pivot ← s.elementAtRank(i) then

lo ← lo +1

s.swapElements(S.atRank(lo), S.atRank(i))

j ← lo + 1

k ← hi

while j < k do

while k > j ∧ S.elemAtRank(k) > pivot do

k ← k – 1

while j < k ∧ S.elemAtRank(j) < pivot do

```
                j ← j + 1
            if j < k then
            S.swapElements(S.atRank( j ), S.atRank( k ))
        S.swapElements(S.atRank( lo ), S.atRank( k )) {move pivot to sorted rank}
        return k
```

C-4-19

```
Algorithm countInversions(S, C, count)
    Input sequence S with n elements, comparator C
    Output number of inversions in S
    if S.size() > 1 then
        (S1, S2) ← partition(S, n/2)
        countInversions (S1, C, count)
        countInversions (S2, C, count)
        count ← merge(S1, S2, C, count)
        return count
    return 0
Algorithm merge(A, B, C, count)
    Input sequences A and B with n/2 elements each, comparator C
    Output number of inversions in S
    S ← empty sequence
    while ¬A.isEmpty() ∧ ¬B.isEmpty() do
        if C.isLessThan( B.first().element(),A.first().element() ) then
            S.insertLast(B.remove(B.first()))
            Count ← count + 1
        else
            S.insertLast(A.remove(A.first()))
    while ¬A.isEmpty() do
        S.insertLast(A.remove(A.first()))
```

```
while ¬B.isEmpty() do
        S.insertLast(B.remove(B.first()))
return S
```

C-4.25

Algorithm matchBolts(A, B)

Input sequence A of n nuts and a sequence B of n bolts	
Output sequence contains items of matched nuts and bolts	
S ← create new sequence	1
While ¬A.isEmpty() ∧ ¬B.isEmpty() do	n
a ← A.removeFirst()	n
b ← B.first()	n
while ¬a.match(b) do	n^2
b ← B.next(b)	n^2
{A match has been found)	
b ← B.removeElement(b)	n
S.insertItem((a,b))	n
Return S	1

Total running time is $O(n^2)$