Homework 4

Nick Palumbo

12.7 In code fragment 12.1 we can make a new function that simply checks if the element that is going to be appended is repeated and if so then the element is not appended.
If the element is not a repeat then we can append and move on to the next element.

12.13 There is a chance that the program will loop till both left and right equal. From there it will go into an infinite loop because the while loop will not check if they are equal and increment. Set $S = [6,6]$ or $S = [1,6,6]$

12.17 The bucket-sort algorithm is not in-place because bucket-sort uses additional containers.
Bucket-Sort creates a new container with the same size as the input sequence and recursively inserts each entry into the bucket.
This algorithm does use the smallest amount of memory need to sort a sequence.

12.19 With the merge_sort algorith, the sequence would sort in $\Theta(n \log n)$ because it is unaware of the list that was an input
With the quick_sort algorithm the sequence would sort in $O(n^2)$. It would be sorted after the first partition but quick_sort is

written to keep partitioning until the whole sequence has had each entry partioned.

12.39 To sort an array in the range $[0, n^2 - 1]$ the sorting algorith to use is radix. Radix is a sorting algorith that starts with the first part of each element and sorts based on that. Then it iterates through the elements sorting based on each element part.

13.8 multiply chain of matrices

$10 \times 5,\ 5 \times 2,\ 2 \times 20,\ 20 \times 12,\ 12 \times 4,\ 4 \times 60$

$M[1,2] = \min_{1 \le k < 2} \{ M[1,1] + M[1+1, 2] + p_0 p_1 p_2 \}$

A_1	A_2	A_3	A_4	A_5	A_6
10×5	5×2	2×20	20×12	12×4	4×60
$p_0\ p_1$	$p_1\ p_2$	$p_2\ p_3$	$p_3\ p_4$	$p_4\ p_5$	$p_5\ p_6$

$A_1 \times A_2 \times A_3 \times A_4 \times A_5 \times A_6$

i\j	1	2	3	4	5	6
1	0	100	500	820	1942	3542
2	x	0	200	600	1602	2842
3	x	x	0	480	576	2792
4	x	x	x	0	960	5,760
5	x	x	x	x	0	2880
6	x	x	x	x	x	0

$M[1,2] = \min_{1 \le k < 2} \{ 0 + 0 + 10 \times 5 \times 2 \}$

$M[2,3] = \min_{2 \le k < 3} \{ 0 + 0 + 5 \times 2 \times 20 \}$

$M[3,4] = \min_{3 \le k < 4} \{ 0 + 0 + 2 \times 20 \times 12 \}$

$M[4,5] = \min_{4 \le k < 5} \{ 0 + 0 + 20 \times 12 \times 4 \}$

$M[1,3] = \min_{1 \le k < 3} \{ 0 + 200 + \underset{p_0\ p_1\ p_3}{10 \times 5 \times 20} \} = 1120$

$M[1,2] + M[2+1,3] + p_0 p_2 p_3$

$100 + 0 + 10 \times 2 \times 20 = 500$

13.9

0 1 2 3 4 5 6 7 8 9
X = G T T C C T A A T A

0 1 2 3 4 5 6 7 8 9 10 11
Y = C G A T A A T T G A G A

GTAATA

13.10

X = skullandbones
Y = lullabybabies

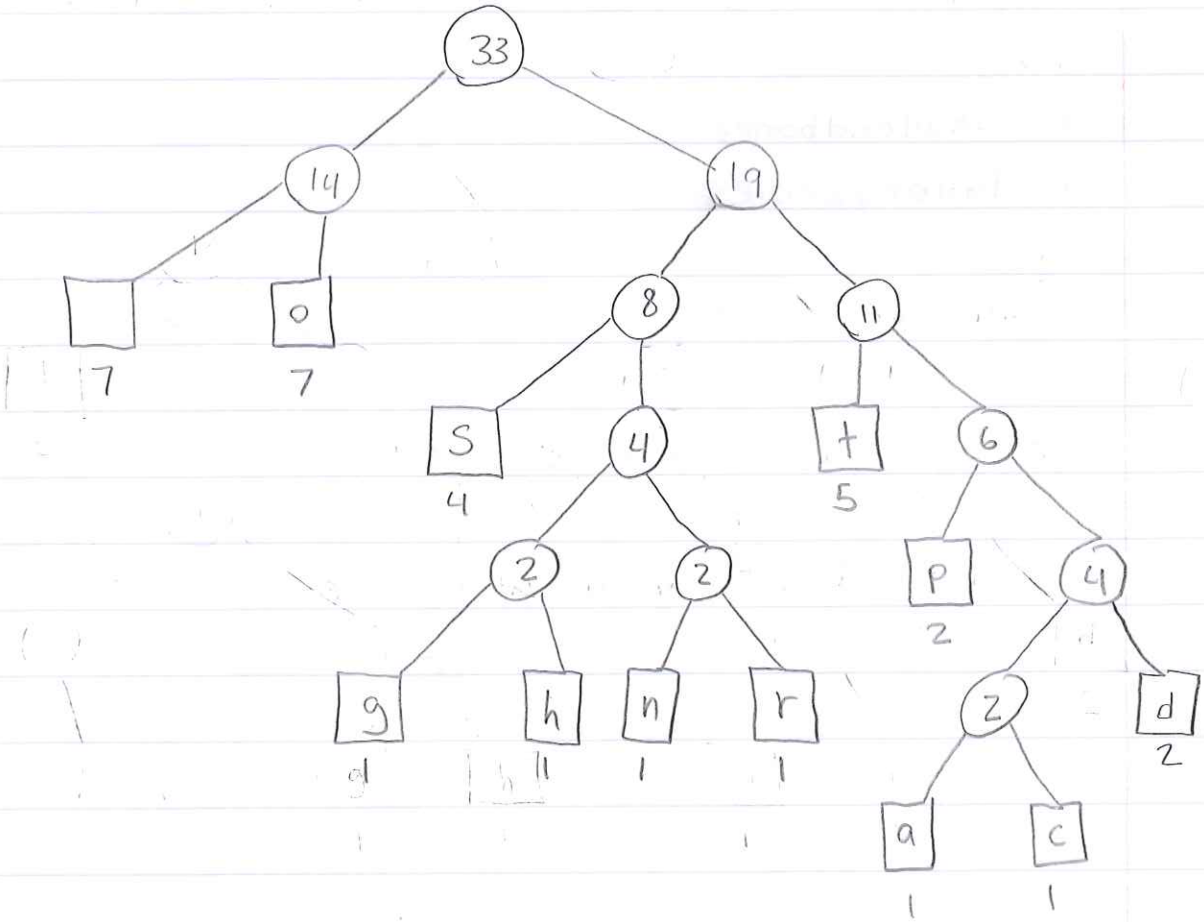
"ullabes"

s k u l l a n d b o n e s

l u l l a b y b a b i e s

13.11 "dogs do not spot hot pots or cats"

character		a	c	d	g	h	n	o	p	r	s	t
frequency:	7	1	1	2	1	1	1	7	2	1	4	5



13.29 An efficiend greed algorithm for making change in a minimum number of coins would start by trying to make change in quarters. After quarters were used, if any in the change, the remainder of the change would try to be made in dimes. After dimes were used, if any, the remainder of the change would try to be made in nickles and then pennies if necessary. This greedy algorithm is correct because in order to use the least amount of coins, the algorithm must start on the most valuable coin so that it gives the most change in the least amount of coins. Then the algorithm can move on to the other coins in descending order so that the least coins are used when making change

5.5 Growing from size k to size 2k requires 3k cyber dollars. Each append operation should be Charged 9 cyber dollars. Each append operation is profited 6 cyber dollars when there is no $3(2^i)$ overflow. Each doubling occurs on 2^i and will cost

5.8

	100	1000	10 000	100000	1000000
k=0	0.334	0.231	1.218	12.038	153.857
k = n // 2	0.308	0.268	0.678	5.671	68.872
k = n	0.216	0.164	0.1662	0.164	0.157

9.3 ~~add(5, A)~~

add(4, B)

add(7, F)

~~add(1, D)~~

remove_min() $\longrightarrow$ (1, D)

~~add(3, J)~~

~~add(6, L)~~

remove_min() $\longrightarrow$ (3, J)

remove_min() $\longrightarrow$ (4, B)

add(8, G)

remove_min() $\longrightarrow$ (5, A)

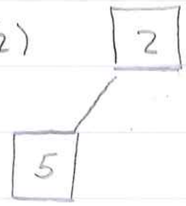
add(2, H)

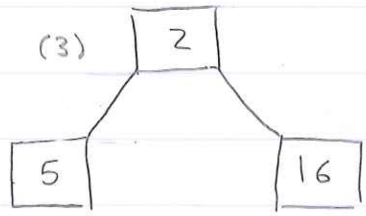
remove_min() $\longrightarrow$ (2, H)

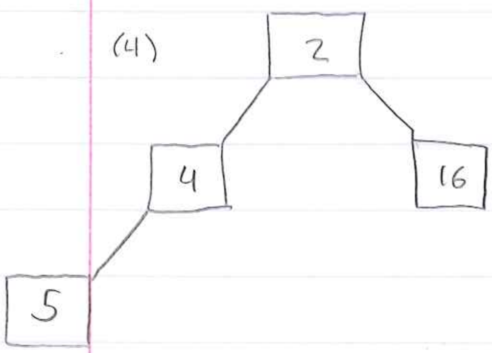
remove_min() $\longrightarrow$ (6, L)

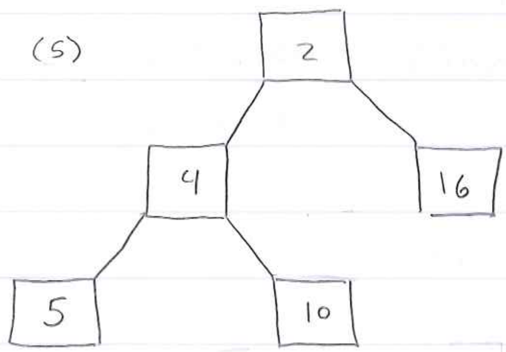
9.13 Illustrate in-place heap-sort algorithm
(2, 5, 16, 4, 10, 23, 39, 18, 26, 15)

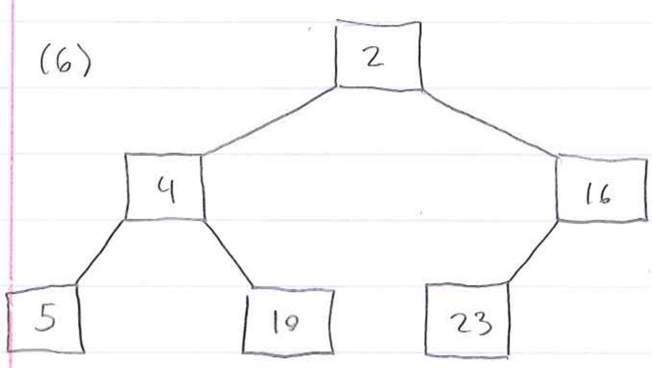
(1)

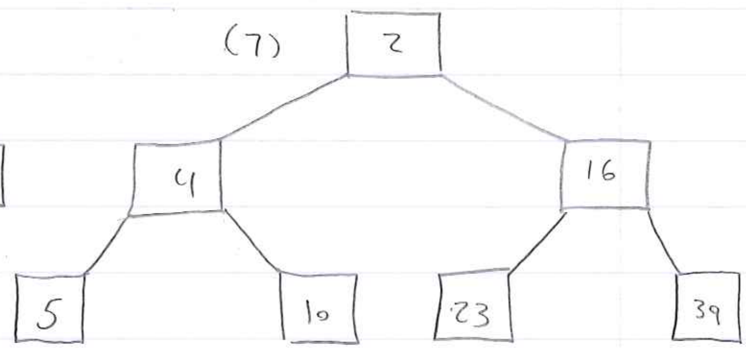

(2)


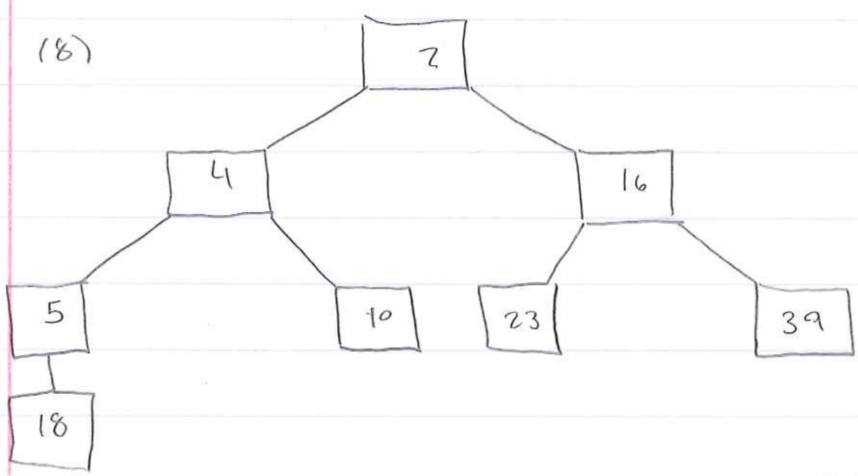
(3)


(4)


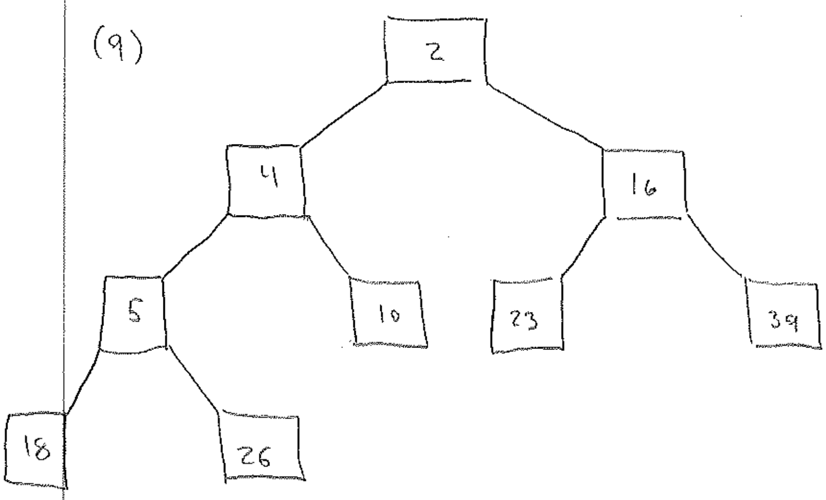
(5)


(6)


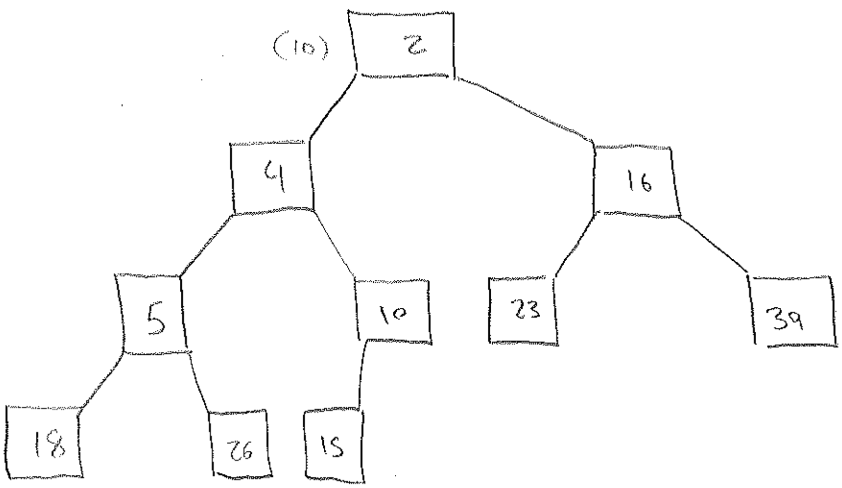
(7)


(8)


(9)



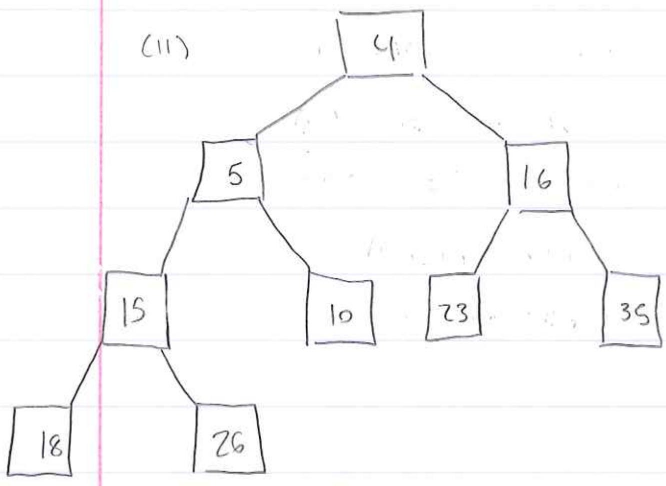
(10)



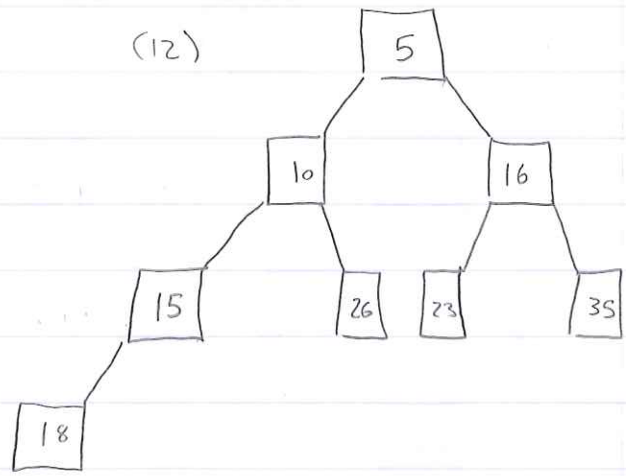
1. (5, 16, 4, 10, 23, 39, 18, 26, 15)
2. (16, 4, 10, 23, 39, 18, 26, 15)
3. (4, 10, 23, 39, 18, 26, 15)
4. (10, 23, 39, 18, 26, 15)
5. (23, 39, 18, 26, 15)
6. (39, 18, 26, 15)
7. (18, 26, 15)
8. (26, 15)
9. (15)

10. ( )

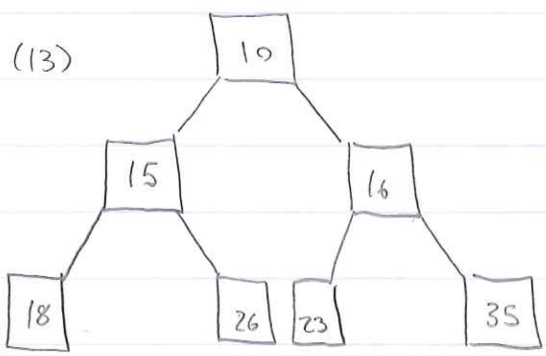
(11)



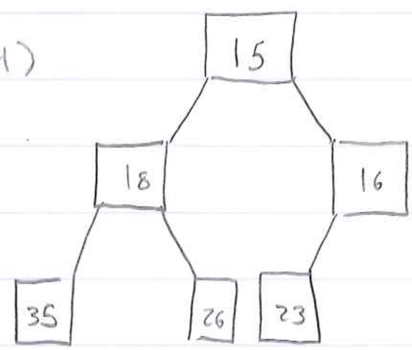
(12)



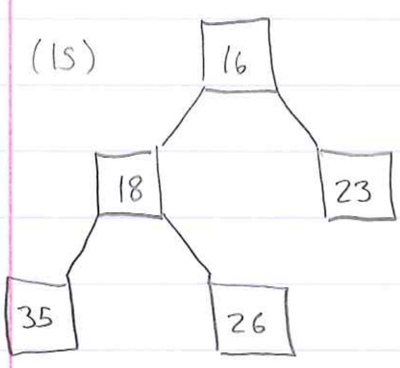
(13)



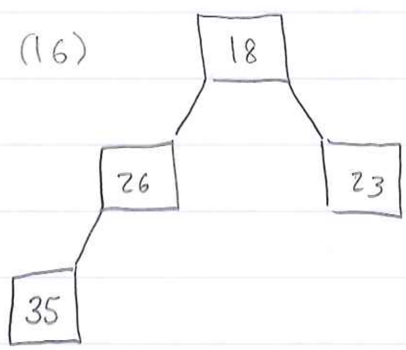
(14)



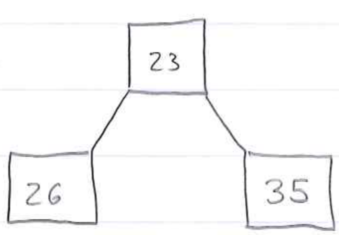
(15)



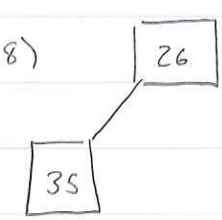
(16)



(17)



(18)



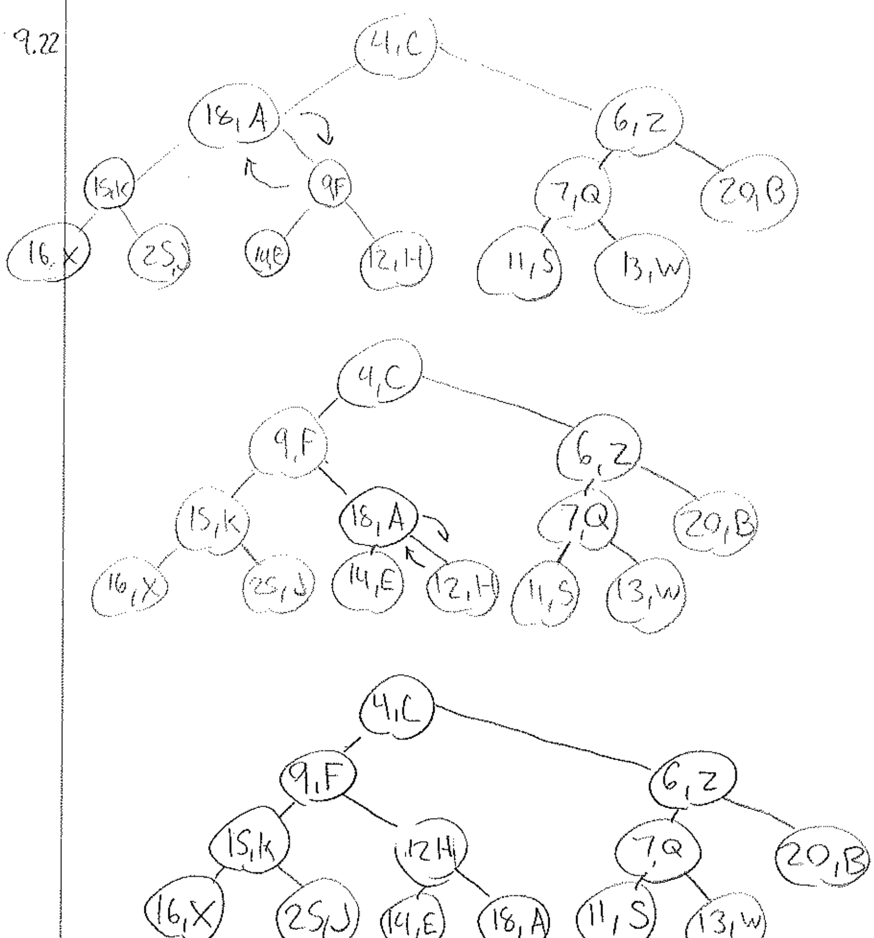
(19)

11. (2)
12. (2,4)
13. (2,4,5)
14. (2,4,5,10)
15. (2,4,5,10,15)
16. (2,4,5,10,15,16)
17. (2,4,5,10,15,16,18)
18. (2,4,5,10,15,16,18,23)
19. (2,4,5,10,15,16,18,23,26)
20. (2,4,5,10,15,16,18,23,26,35)

9.15 The description of down-heap bubbling does not consider the case in which position p has a right child but not a left child because when running a down-heap bubbling algorithm it only needs to consider the minimum value

9.18 $\sum_{i=1}^{n} \log i = \log \left(\sum_{i=1}^{n} i \right) = \log (n!) . \left(\frac{n}{2} \right)^{\frac{n}{2}} \le n! \le n^{\frac{n}{2}}$

9.22



```
9.35  def lessThanKey(H, n):
          if n.key < K
              left.append(n.value, n.key)
              if n.left:
                  lessThanKey(H, n.left)
              if n.right:
                  lessThanKey(H, n.right)
```