Homework 7

7.1 3. b c d c b a a b

	a	b	c	d
frequencies:	2	3	2	1
distribution:	2	5	7	8

A[7] = b	2	5	7	8
A[6] = a	2	4	7	8
A[5] = a	1	4	7	8
A[4] = b	0	4	7	8
A[3] = c	0	3	7	8
A[2] = d	0	3	6	8
A[1] = c	0	3	6	7
A[0] = b	0	3	5	8

sorted array

S[0] S[1] S[2] S[3] S[4] S[5] S[6] S[7]

b

a

a

b

c

d

c

b

a	a	b	b	b	c	c	d

6 Algorithm Ancestor(T, u, v)

// Input: Binary tree T, vertex u, vertex v

// Output: Returns true if u is ancestor of v, else returns false

create an empty stack S

pre ← 0

push root of T into S

while stack S is not empty do

pop node x from S

preorder(x) ← pre

pre ← pre + 1

if $T_{left} \neq \emptyset$

push T_{left} into S

if $T_{right} \neq \emptyset$

push T_{right} into S

empty stack S
post ← 0
push root of T into S
while stack S is not empty
    pop node x from S
    if $T_{left} \neq \phi$
        push T_{left} into S
    if $T_{right} \neq \phi$
        push T_{right} into S
    postorder(x) ← post
    post ← post + 1
if preorder(u) ≤ preorder(v) and postorder(u) ≥ postorder(v)
    return 1
else
    return 0

$0 = k \quad i = 3$

7a.

A

0	1	2	3	4	5	6	7
	y		x				z

B

0	1	2	3	4	5	6	7
3	7	1					

C

0	1	2	3	4	5	6	7
	2		0				1

b. To check whether A[i] has been initialized we have to check that B[C[i]] = i. If this is true than A[i] has been initialized, otherwise than it is false and A[i] has not be initialized.

To check the value that A[i] has been initialized, we must make sure that $0 \leq C[i] \leq counter - 1$ and B[C[i]] = i. If this is true than A[i] has the value that is initialized with it.

7.3 1a. 30, 20, 56, 75, 31, 19 $h(K) = K \bmod 11$

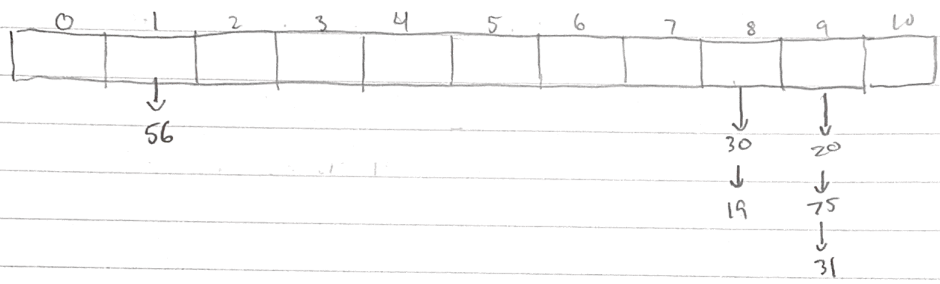
$h(30) = 30 \bmod 11 = 8$ $h(75) = 75 \bmod 11 = 9$

$h(20) = 20 \bmod 11 = 9$ $h(31) = 31 \bmod 11 = 9$

$h(56) = 56 \bmod 11 = 1$ $h(19) = 19 \bmod 11 = 8$

keys	30	20	56	75	31	19
hash addresses	8	9	1	9	9	8

hash table



b. The largest number of key comparisons in a successful search in this table is 3 since the largest number of keys one call holds in this table is 3.

Key	30	20	56	75	31	19
No. Comp	1	1	1	2	3	2

c. $1 + 1 + 1 + 2 + 3 + 2 = 10$ 6 keys $10/6 \approx 1.67$

The average number of key comparisons in a successful search in this table is 1.67

2a.

keys	30	20	56	75	31	19
hash addresses	8	9	1	10	0	2

$h(30) = 30 \bmod 11 = 8$

$h(20) = 20 \bmod 11 = 9$

$h(56) = 56 \bmod 11 = 1$

$h(75) = 75 \bmod 11 = 9 + 1 = 10$

+1 due to collision

$h(31) = 31 \bmod 11 = 8$ but collision so wrapped to next open so 0

$h(19) = 19 \bmod 11 = 8$ but collision wrapped to next open so = 2

0	1	2	3	4	5	6	7	8	9	10
								30		
								30	20	
	56							30	20	
	56							30	20	75
31	56							30	20	75
31	56	19						30	20	75

b. The largest number of key comparisons in a successful search in this table is 6, which is when we search for the key 19.

c. $1+1+1+2+3+6 = 14 \quad 14/6 \approx 2.33$

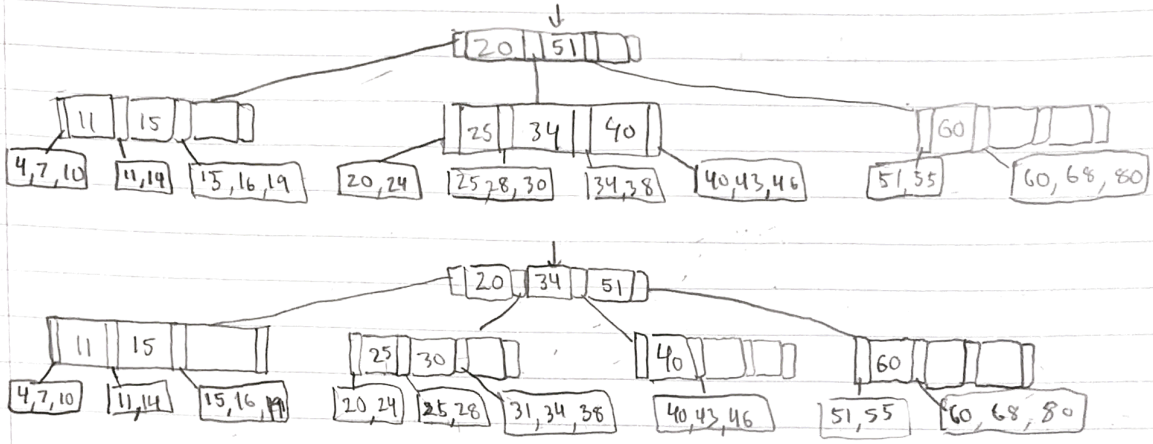
The average number of key comparisons in a successful search is 2.33

7. To check whether all elements of a list are distinct using hashing, one must compute the hash value for all elements and then each elements element will compare against the hash values. If there is one match for all elements, they are distinct. The time efficiency is $O(n)$ since the total time complexity is $O(n) + O(n)$ for computing hash values for all elements and comparing for all elements. This is much faster than brute force version, $O(n^2)$, and presort algorithm, $O(n \log n)$.

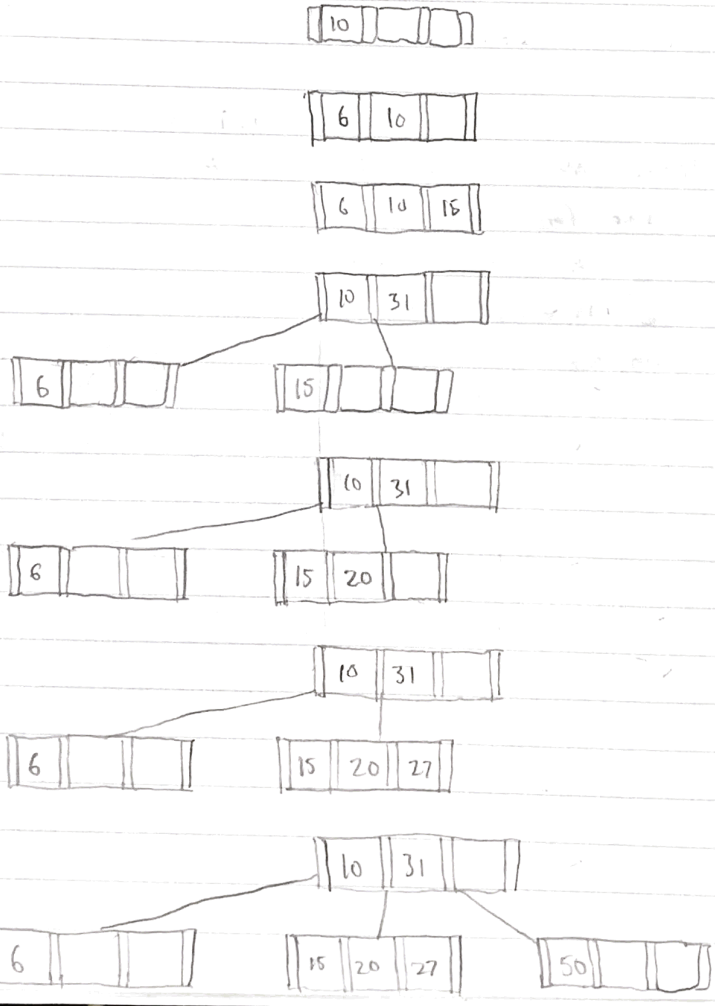
8

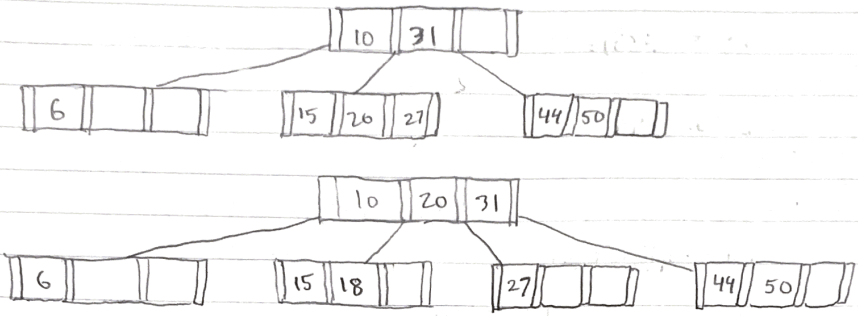
	unordered array	ordered array	binary search tree	balanced search tree	hashing
search	$\Theta(n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
insertion	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
deletion	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$

7.4 4



6a. 10, 6, 15, 31, 20, 27, 50, 44, 18





b. The principal advantage of this insertion procedure is that when inserting a new key, the splitting of a full node will happen if the leaf where the key is being inserted turns out to be full. This split never causes a chain reaction of splits because the parents of the leaf will always have room for an extra key.

The principal disadvantage of this splitting of full nodes on the way down of inserting the new key is that it can lead to a larger tree than required.

8.1 2 5, 1, 2, 10, 6

Index	0	1	2	3	4	5
C	—	5	1	2	10	6
F	0	5	5	7	15	15

$F(0)=0$, $F(1)=c_1$

$F[0]=0$

$F[1]=c_1=5$

$F[2]=\max[1+0,5]=5$

$F[3]=\max[2+5,5]=7$

$F[4]=\max[10+5,7]=15$

$F[5]=\max[6+7,15]=15$

3a. $F(n) = F(n-1) + F(n-2)$ $F(0)=0$ basic operation is $O(1)$

$T(n) = T(n-1) + T(n-2) + O(1)$

$T(n-1) = T(n-2) + T(n-3) + O(1)$ $T(n-3) = T(n-4) + T(n-5) + O(1)$

$T(n-2) = T(n-3) + T(n-4) + O(1)$ $T(n-4) = T(n-5) + T(n-6) + O(1)$

so

$T(n) = T(n-2) + T(n-3) + O(1) + T(n-3) + T(n-4) + O(1) + O(1)$

$= T(n-3) + T(n-4) + O(1) + O(1) + T(n-4) + T(n-5) + O(1) + O(1) + T(n-4) + T(n-5) + O(1) + O(1) + T(n-5) + T(n-6) + O(1) + O(1)$

$= T(n-3) + 2[T(n-4) + T(n-5) + O(1) + O(1)] + T(n-5) + T(n-6) + O(1) + O(1) + O(1) + O(1)$ ...

$= T(n) = 2^k T(n-k)$

$T(n) = 2^k T(n-k)$ sub $n=k$ since eventually it will reduce down to

$= 2^k T(k-k)$

$= 2^k T(0)$

$= 2^k O(1)$

$= 2^k$

Thus

$T(n) = 2^n$

Therefore coin-row problem by straight-forward application of recurrence is exponential, because time complexity is $O(2^n)$.

b. The time efficiency of solving the coin-row problem by exhaustive search is at least exponential, because this method tries all possible combination of numbers to get the max amount. Thus the possible number of ways is $C(2n, n)/n+1$. And combinatorial function grow exponentially. Therefore the time efficiency is at least exponentially.

4. denominations 1, 3, 5 amount n=9

n	0	1	2	3	4	5	6	7	8	9
F	0	1	2	1	2	1	2	3	2	3

$F[0] = 0$

$F(1) = \min[F(1-1)] + 1 = 1$

$F(2) = \min[F(2-1)] + 1 = 2$

$F(3) = \min[F(3-1), F(3-3)] + 1 = 1$

$F(4) = \min[F(4-1), F(4-3)] + 1 = 2$

$F(5) = \min[F(5-1), F(5-3), F(5-5)] + 1 = 1$

$F(6) = \min[F(6-1), F(6-3), F(6-5)] + 1 = 2$

$F(7) = \min[F(7-1), F(7-3), F(7-5)] + 1 = 3$

$F(8) = \min[F(8-1), F(8-3), F(8-5)] + 1 = 2$

$F(9) = \min[F(9-1), F(9-3), F(9-5)] + 1 = 3$

7a

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	1	2	3	4	5	6	7	8
3	1	3	6	10	15	21	28	36
4	1	4	10	20	35	56	84	120
5	1	5	15	35	70	126	210	330
6	1	6	21	56	126	252	462	792
7	1	7	28	84	210	462	924	1716
8	1	8	36	120	330	792	1716	3432

$P(1,j) = P(i,1) = 1$

$P(i,j) = P(i-1,j) + P(i,j-1)$

The number of shorest paths from one corner to another corner that is diagonally opposite is 3432.

10a: Algorithm DagLongestPath(G)

```
// Input: A weighted Dag
// Output: Length of longest path
for every vertex v do
    d_v ← 0        // length of longest path to v so far
for every vertex V in topological order do
    for every vertex u adjacent to v do
        if d_v + w(v,u) > d_u
            d_u ← d_v + w(v,u)
d_max ← 0
for every vertex v do
    if d_v > d_max
        d_max ← d_v
return d_max
```

b. We can reduce the coin-row problem to the problem of finding the longest path in a dag by F[i] ← max(C[i] + F[i-2], F[i-1]). where we partition all allowed coin sections into two groups, and compare max.

12a. $P(i,j) = p * P(i-1,j) + q * P(i,j-1)$ for $i,j > 0$ where p is probability of team A winning and q is probability of team A loosing the same.

b. $P(0,0) = 0$ $P(i,0) = 0$ $P(0,j) = 1$

i/j	0	1	2	3	4
0		1	1	1	1
1	0	0.4	0.64	0.78	0.87
2	0	0.16	0.35	0.52	0.66
3	0	0.06	0.18	0.32	0.46
4	0	0.03	0.09	0.18	0.29

Thus P[4,4] ≈ 0.29

c. Algorithm WorldSeries(n, p)

// Input: A number of wins n needed to win the series
and probability p of one team winning a game

// Output: The probability of the team winning the series

```
q ← 1 − p
for j ← 1 to n do
    P[0, j] ← 1
for i ← 1 to n do
    P[i, 0] ← 0
    for j ← 1 to n do
        P[i, j] ← p * P[i−1, j] + q * P[i, j−1]

return P[n, n]
```

The time efficiency and space efficiency of this algorithm are in $O(n^2)$.