Ex 5-1. ✓✓ 7/7

50/100

Ex 5-2.

✗ 0/6

Ex 5-3. ✓✓ 7/7

Ex 5-4.

What I would do is, I search for interval, and when algorithm succesfully finds one, I check lowest number and when it is not we search for we move, if we find we return it. Code could be, we need to modify pseudo code from book,

```
Interval Search(T, i)
  roo = T.root
  while(x != T.nil and i doesn't overlap x.int and min != T.nil){
      if (x.left != T.nil and x.left.max ≥ i.low)
            x = x.left
      else x = x.right
  if (min = null)
       return T.nil
  else{return x}
```

6/6

6

Ex 5-5. ✓✓ 7/7

Ex 5-6. ✓

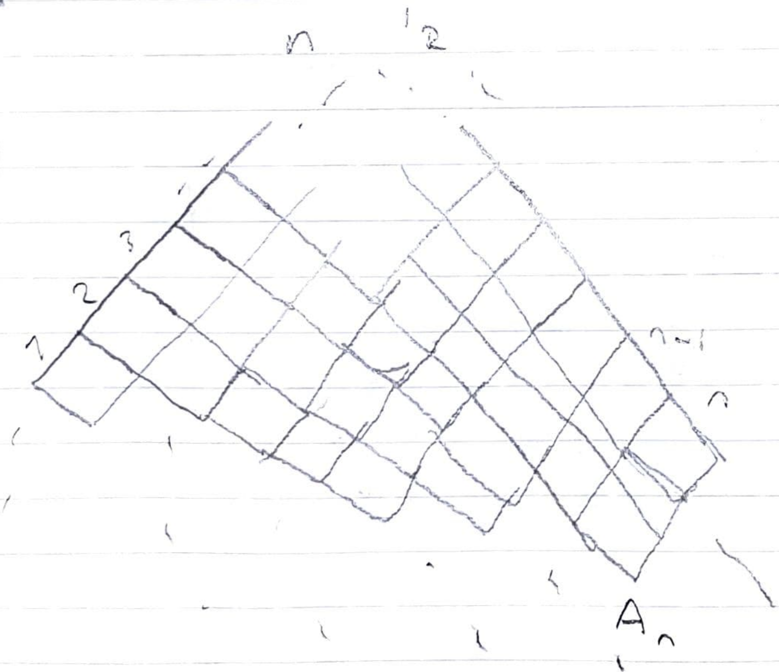
We have the function memoized cut rod from the book, So solution would be

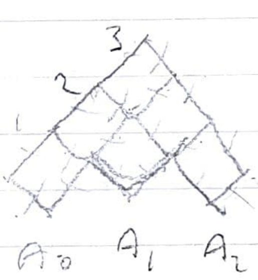
```
func(p, n, r, ar)        //ar is array passed from same func.
    if r[n] ≥ 0
        retun ar

    if n = 0
        q = 0
    else q = - ∞
        for i=1 to n
            q = max(q, sum(p[i]) + func(p, n-1, r, ar)
        r[n] = q, ar.add(q)
    retun ar
```

2/6

Ex 5-7.

1,5/6





18

Edge = 4n - 1

4/6

Ex 5-8. define set

```
func(arr, length)
    for i=0 to length
        if(arr[i] not in set)
            set.add(arr[i])
            k = set.get(arr[i])
            k++
            if(k != set.end()). set.remove(k)
    return set;
```

PS. 1

7/12

a)

```
finger_search( x, keyx)
  while( x.hasUp and x.next.key > keyx )
        x = x.up
  while( x.next.key < keyx)
       x = x.next
       if(keyx = x.key) return x
  while(x.key != keyx)
      if (x.key > keyx)
         x = x.next
      x = x.down
  return x.
```

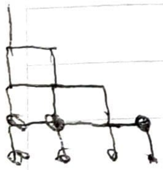
Since there is likely lg n steps or floors since with fair coin toss, this is the number we can go deep.

3/6

b) downrange, this data stores x.down.next value for node x, this way we can decrease one third of operations, since we don't need to go down, get next and turn back etc. each time which will save cost for search ops.
Search → We can look range more easily
Insert → Whenever we insert, we go y.prev.up and change downrange to y.key and check downrange for y and init.
Delete → go to prev.up change downrange to y.next.

c) First, init all ranks,

```
 func init()
     if(x.atBottom)
         if(x.left!=NULL)
             rank(x)= rank(x.prev)+1
         else(rank(x)=1)
     else
         rank(x)= rank(x.under)
```



```
func rank_search(x, r)
 while(x.up!=NULL)
     x=x.up
 while(rank(x.next)<r)
        x=x.next
   x=x.down
   goto
   return x
```

while

in it is O(n)
but search is O(lgn)
naturally.

$\frac{3,5}{7}$

PS-2.

```
      count=0
a) for i=0 to i=n
       if(P[i]<val)
           arr.add(P[i])
           count++
     if(count==n)
         return arr
```

this would take
O(n) time as
each iteration ops
are const.

$\frac{2}{4}$