2-1. ✓ read $\frac{8.3}{8.3}$

2-2.) $7x^3 - x^2 + x - 10$ and $8x^3 - 6x + 3$

$\frac{0}{8.3}$

2-3.) from the book recursive-FFT(a)

n = a.length

if (n == 1) return a

$w_n = e^{2\pi i/n}$

$w = 1$

$a^{[0]}$ = even indexes $(a_0, a_2, a_4 \ldots a_{n-2})$

$a^{[1]}$ = odd " $(a_1, a_3, a_5 \ldots a_{n-1})$

$y^{[0]}$ = recursive-fft $(a^{[0]})$

$y^{[1]}$ = recursive-fft $(a^{[1]})$

for $k=0 \rightarrow k = n/2 - 1$

$y_k = y_k^{[0]} + w y_k^{[1]}$

$y_{k+(n/2)} = y_k^{[0]} - w y_k^{[1]}$

$w = w \cdot w_n$

return y;

$\frac{8.3}{8.3}$

$T(n) = O(n \log n)$

2-4 ✓ Redd 8.3/8.3

18.2-5

S/8.3

Say a leaf can take two more keys. Now max key count is 7.
When inserting to leaf with size <7 regularly add key.
Take the middle key one up and divide the leaf into two leafs divided from the middle. After that, add key to appropriate place. Now (old) middle now parent keys left points to new leaf from left and other side points to right leaf.
Parent node can now take up to normal count while child leaves can still take some more.

* I believe solution on course website is false, in that solution from one side of a key there can go two pointers to different leafs but this is illegal.

3/8,3

18.3-2

delete(k)

remove(find(k))
if (isleaf(k) and number of keys in leaf <t)
    remove items on leaf
    insert(items)

if (isparent(k))
    remove(childs)
    insert(childs)

* not same as solution provided but I can't see why this wouldn't work. A bit more time consuming though.

Problem 2-1

a) Here most trivial solution would be traversing the entire S string and at each iteration look if current next characters match P. Code would look like this

4/4

```
for (i → n-3)
    for (j=0 → m)
        if ( S[i → i+3] == P( j  j+3)
            list.add(i)
```

Since two lopps are m and n respectively time complexity is O(m.n).

b) My solution is to give all characters in S 1 or -1 each. Say a → 1, b → -1 then give each character of P opposite value so that a=-1, b=1. Multiply those two if we find $-1 * m$ as solution where m is number of chars in P except * character.
For example let S = ababbab and P = ab *
S becomes 1,-1,1,-1,-1,1,-1 and P = -1, 1, 0
In first multiplication, $-1 \times 1 + -1 \times 1 + 0 = -2$ now we see that, -2 is $= -1 \times m$ and $m = 2$ so we found an answer at 0.
Adding 0 to array and continuing, multiply S with 0,-1,1,0,0,0,0
Result is 2 and this doesn't fit our criteria. Move on, we have P = 0, 0, -1, 1, 0, 0, 0 mult with S we have,
$0 - 0 - 1 - 1 + 0 + 0 + 0 = -2$ which again fits and add 2 to array.
Time complexity: Since we have to iterate over n-m element which is basically n and at each cycle multiply m elements (we can ignore 0s with O(1)) we have O(m.n) complexity.

11/12

* My solution is a bit similar to official but I assure you, this is my own solution. Did not even look internet.

c)

0/3

Here think P as a matrix but to construct this, we merge P vectors constructed as in b) and at each row shifting them for example if P = -1, 1, -1 then

matrix

$$\begin{array}{cccccc} -1 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & 0 & \ldots & & \end{array}$$

as this. With the matrix x vector multiplication at each multiplication skip 0s and directly do multiply operation since we multipy m vectors with FFT this woud result in $m \cdot \log(m)$ but we do n operations so we have as result $T(n) = n \cdot m \cdot \log(m)$

d) a

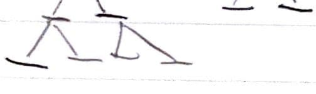
0/6

P 2-2.

3/5

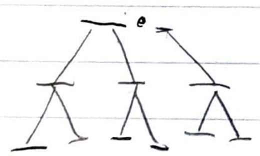
a) Here solution is simple let left pointer of k point to T_1 and right points to T_2 and since $h_1 = h_2$ Btree property is satisfied. Since keys $T_1 < k$
keys $T_2 > k$ this would work in $O(1)$ because we don't have to manage anything.

b)



Here, I would take all keys at top of H_1 next to k to satisfy balance.

2/5



this would work if keys of left +1 is within range.

Else; Again combine left with k but take the middle at top to one up. These are both $O(1)$ algorithms since we make const num of ops ex: 1, 3 at most.

c) 0/5

d) 0/10

60/100