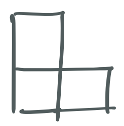
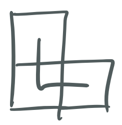
	T1	T2	T3
Average mark	27.6	36.4	39.2
per cent	75%	65%	70%

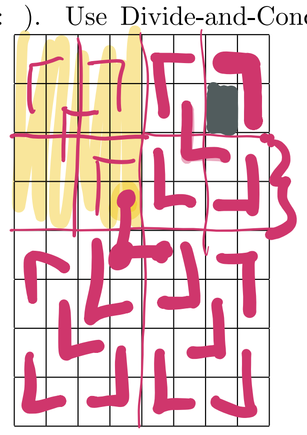
Computer Science 260 Midterm 3: Hashing, Divide and Conquer, Dynamic Programming, AVL trees

Out of 56 possible marks NAME:___________________________________

1. (6 marks) Circle either **T** (True) or **F** (False), whichever is a more accurate assessment of the following statements:

T [✓ **F**] A hash function is always computed "mod m", where m is the total number of keys to be stored.
[✓ **T**] **F** To rebalance and AVL tree after an insertion, rotations are performed at the location of the lowest node that is out of balance.
T [✓ **F**] The minimum key value in a hash table can be found in O(log$n$) time.

2. (4 marks) Here is an 8x8 grid, with a square missing. Herringbone tile the rest of the grid. (Herringbone tiles look like this: ). Use Divide-and-Conquer to find the tiling. Mark a tile spanning three squares like this: 



3. (8 marks) Dance Contest! The annual Dance Contest has published the details, and you know exactly how many points you will gain for each of these dances, if you dance it. Some dances tire you out, and you have to "skip" a few of the following dances. Below is presented, for each dance in the order they will be executed, the points you would gain if you danced it, and the "offset" to the next dance you can dance without being exhausted. In other words, if the Offset is 1 for dance number 8, then the next dance you can dance is 8+1=9; so that dance does not tire you out at all, you are ready for the next one! Offsets must be at least 1. Give the optimal set of dances, and the total number of points.

Dance	1	2	3	4	5	6	7
Points	5	10	14	3	6	7	11
Offset	2	2	4	4	2	3	1
Dance It	5+25 30	10+17 27	14+11 25	3+0 3	6+11 17	7	11
Don't Dance	27	25	17	17	11	11	0

30 points
dancing 1, 3, 7.

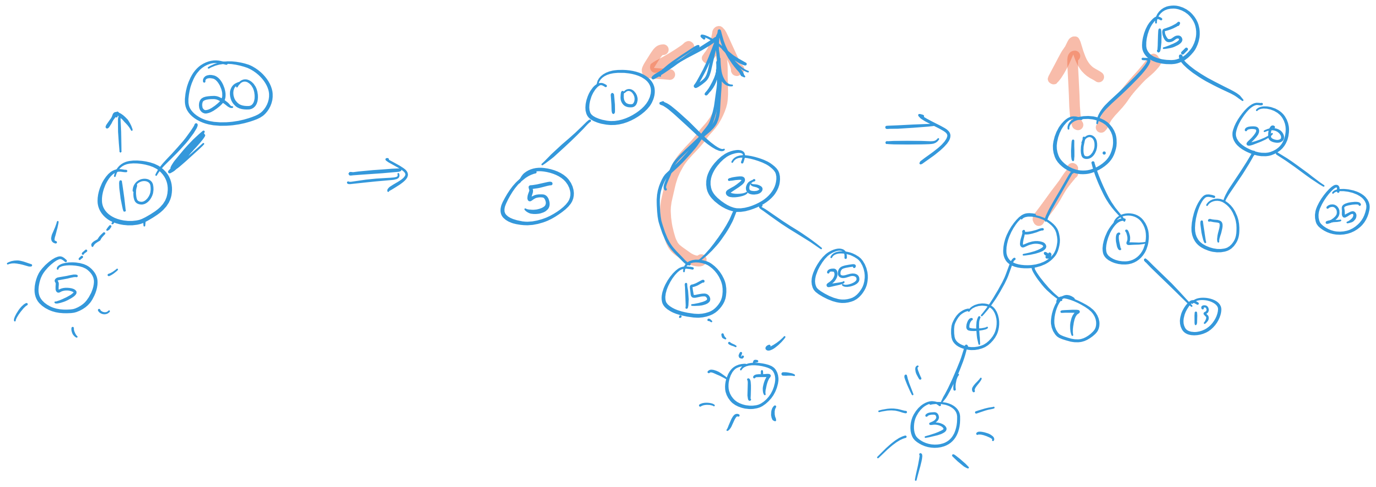
ε λ

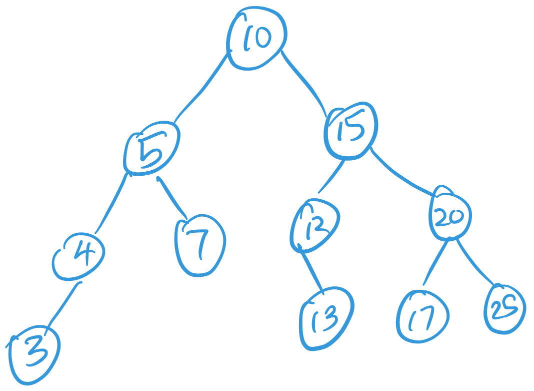
4. (8 marks) Use Dynamic Programming to find the longest common subsequence (not necessarily contiguous) in the following two strings, by filling the table below.

	λ	A	C	C	B	B	A	C	D
λ	0	0	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1	1	1
A	0	1	1	1	1	1	2	2	2
C	0	1	2	2	2	2	2	3	3
B	0	1	2	2	3	3	3	3	3
B	0	1	2	2	3	4	4	4	4
C	0	1	2	3	3	4	4	5	5
D	0	1	2	3	3	4	4	5	6
A	0	1	2	3	3	4	5	5	6

6 letter sequence.

5. (8 marks) Perform the AVL inserts of 20, 10, 5, 15, 25, 17, 12, 7, 13, 4, 3 into an originally empty AVL Tree. Do them in that order; show the tree that results. You do not need to redraw the tree as it grows, UNLESS it needs rebalancing – you must redraw it after each rebalancing.





6. Consider the hash function $h(c_1c_2\dots c_n) = (\Sigma_{i=1}^{n}\text{ord}(c_i))\text{mod } m$. Note that $\text{ord}(c_i)$ is the ordinal number of the letter c_i in the alphabet. The ordinal values are provided here for your convenience:

										0	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4
1	2	3	4	5	6	7	8	9	10	~~11~~	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	1											1	9												

(a) (6 marks) Insert the following keys into the hash table below, using open addressing (all elements are stored in the table itself) with double hashing to handle collisions. The secondary hash function is "string length + number of vowels" mod m. For example, the secondary hash of EE is 4.

AN	MB	HB	CD	EE	LN	GI	UF
4	4 2+			10, 4	4, 2	5	5, 3

UF			EE	AN	GI	MB	CD	LN		HB
0	1	2	3	4	5	6	7	8	9	10

(b) (8 marks) Write an Insert(string key) algorithm, in pseudocode, for the above table, called T. For simplicity, let T be a table of keys only, and $T[i] == EMPTY$ and $T[i] == DELETED$ are the accepted usage for detecting if a slot i has never been written to, or the element there has been deleted, respectively.

You can assume that the primary hash function `h(string k)` is provided; also provided is a function `h2(string k)` that computes "string length + number of vowels (mod 11)" – i.e., you don't have to write the code for these, you can just call these functions.

```
Insert(K)
    s = h(K)
    original s = s
    offset = $h_2(K)$
    while (T[s] != EMPTY ~~or~~ T[s] != DELETED)
        s = (s + offset) mod m
        if s == original s    return "TABLEFULL"
    T[s] = K
    return s
```

7. (8 marks) Give a simple, efficient, recursive, Dynamic Programming algorithm to perform exponentiation. The input is a Hint: $a^n = a * a^{\frac{n}{2}} * a^{\frac{n}{2}}$, if n is odd, and $a^n = a^{\frac{n}{2}} * a^{\frac{n}{2}}$, if n is even. Integer division, rounding down, is used in the exponents.

Another hint about efficiency: not all the entries in `aTo[]` need to be computed – not all are used for a particular value of `n`. You can use a "just in time" computation strategy – compute it only if it is needed and it is not yet computed (you can test if it has already been computed by checking if it's value is not 0).

```
Assume that a global variable "double aTo[1..n]", an array of floating point numbers,
is declared and initialized to zeroes.

Algorithm Expon( a, n )
input: a is a floating point number, n is a positive integer
output: a^n
```

```
if n==1 return a
if aTo[n] != 0 return aTo[n]
if aTo[n/2] == 0
    aTo[n/2] = Expon(a, n/2)
if n is odd
    return a* aTo[n/2] * aTo[n/2]
else return aTo[n/2] * aTo[n/2].
```