1.

0	1	2	3	4	5	6	7	8	9	10
	3		0	12			9	70		
				1			42			
				98						

Linear Probing

0	1	2	3	4	5	6	7	8	9	10
	3		0	12	1	98	9	42	70	

Quadratic

0	1	2	3	4	5	6	7	8	9	10
	3		0	12		42	9	98		1

70 Could not be inserted at (8*8 +5) % 11 is 3 and (3*3+5) % 11 is still 3. As 0 is already in position 3, we cannot insert 70.

2.

I would pick 101 as its prime and we're likely to have more than 7 pieces of data which is the only other prime number. If we picked 7, we would have to rehash soon anyways so it's better to start with something large and prime.

3.

Load Factor: 53491/106963 = 0.500009

Given linear probing with a load factor of 0.5, it's likely that we'll have to probe a few times every insert and search meaning we should probably rehash. We'll also run into issues with data clumping together which makes inserts and searches worse in those spots

For separate chaining, we shouldn't have as bad of an issue with data clumping together meaning we could probably get away with a load factor of 0.5. However, we a are approaching a point probably around 0.7-0.8 where we might have to consider rehashing.

4.

Function	Big O
Insert	O(1)
Rehash	O(n)
Remove	O(1)
Contains	O(1)

5.

```
Int hashit(int key, int tablesize) {

Int position  = -1;

Position = key * 2654435761 % tablesize;

Return position;

}

Int hashit ( std::string key, int tablesize) {

Int position = -1, stringint = 7;

For ( std::string::size_type I = 0; i<key.size(); i++) {

            Stringint = stringint*17 + key[i];

}

Position = stringint % tablesize;

Return position;

}
```

6.

When resizing, we multiple the original table size by 2 which gives it an additional factor of 2 and severely hinders the spread of the data being inserted or searched for.

7.

When several computations are being performed simultaneously and can be seen as a sort of divide and conquer approach to programs.

8.

You could take the amount of work that needs to be done and divide by the amount of threads available which would assign random work to each thread.

You could also try to weigh different problems by complexity and separate them into somewhat equally complex piles and assign them to different threads hopefully to balance out some randomness.