1) In FFS, a new block is appended to a file. Name 3 structures that must be updated.

2) File Tail Waste.

a) Block size is 1024. File is 512 bytes. What percent of used space is wasted?
b) Block size is 2048. File is 512 bytes. What percent of used space is wasted?
c) Block size is 2048. File is 1024 bytes. What percent of used space is wasted?
d) All files are between 1 and 8KB. Each size is equally common.
   i) how much space does the average file waste?
   ii) how large is the average file?
   iii) how much space does the average file consume?
   iv) what percent of space is wasted in used data blocks in the file system?

3) Give 5 examples of redundancy in FFS.

4) We're appending. What if we crash after writing only 1 or 2 of the 3 blocks?

a) bitmap
b) data
c) inode
d) bitmap and data
e) bitmap and inode
f) data and inode

5) Disk banking.

Suppose you're writing a banking application that uses disk directly (there is no file system). Sadly, you only have two customers. Also (again sadly), your disk has only five blocks. The account balance of each customer is stored in a block (so two blocks are used for this). Your application can use the extra three blocks for bookkeeping however you see fit.

You need a method to transfer money between customers, updating the two blocks. If the system crashes, you need to make sure the customers don't lose money, or worse (for you), get free money.

The disk only does one write at a time. It cannot reorder writes. Furthermore, each individual write to a single block is atomic. So if block 3 contains A, and you crash while writing B to block 3, you will see either A or B in block 3 when you recover. You would never get Z, for instance.

a) First, think about a concrete example. Alice has $12 and Bob has $3. Alice is paying Bob $2. If we crash in the middle of the transaction, either (a) Alice should still have $12 and Bob should still have $3, or (b) Alice should have $10 and Bob should have $5. Any other combination would be wrong. In each row below, update one block from the previous row. Do your updates such that for each time, you have enough information on disk to recover to one of the desired states.

Time	**Block 0: Alice**	**Block 1: Bob**	**Block 2: extra**	**Block 3: extra**	**Block 4: extra**
1	12	3	0	0	0
2					
3					
4					
5					
6					
7					
8					

b) Write pseudo code for arbitrary updates (update_accounts), and the logic to perform after a crash. This should update blocks 0 and 1 safely.

```
void update_accounts(int cash1, int cash2) {







}

void recovery() {






}
```