1. Assume 0-based logical block addresses and that B.PTR[n] stores the nth pointer in block B where n is zero-based.

a. FIND-CONTIGUOUS-ADDRESS(L,F)
   
   \[ P = FIRST + L \]
   
   return \( P \)

b. FIND-LINKED-ADDRESS(L,F)
   
   \[ P = FIRST; \]
   \[ I = 0 \]
   
   while \( I \) is less than \( L \)
     \[ B = LOAD-BLOCK(P) \]
     \[ P = P.next \]
   
   return \( P \)

c. FIND-INDEXED-ADDRESS(L,F)
   
   \[ P = ERROR \]
   if \( L \) is less than 10
     \[ P = PTR[L] \]
   else if \( L \) is less than 110
     \[ B = LOAD-BLOCK(PTR[10]) \]
     \[ P = PTR[L - 10] \]
   else if \( L \) is less than 10 + 100 + 100*100
     \[ B = LOAD-BLOCK(PTR[11]) \]
     \[ I = \text{floor of } \left( \frac{L - 110}{100} \right) \]
     \[ B = LOAD-BLOCK(B.PTR[I]) \]
     \[ I = \text{remainder of } \left( \frac{L - 110}{100} \right) \]
     \[ P = B.PTR[I] \]
   
   return \( P \)
2. Note that the following answers assume that a file owns n blocks, that all directory accesses (read or write) can be done in a single operation, and that all files allocated using the linked scheme are doubly linked.

   a. Reversal In Place

      i. Contiguous Allocation Scheme

         This answer depends on if the file has an even or odd number of blocks.

         If even, $2n + 1$ disk operations:
         1 read for the directory
         1 read for each block in the file
         1 write for each block in the file

         If odd, $2(n-1) + 1$ disk operations:
         1 read for the directory
         1 read and one write for each block excluding the middle block

         The algorithm for reversing in place reads the first and last blocks of the file and
         writes them to the swapped locations. Each block must be read once and
         written once with the possible exception of the middle block in an odd block
         count scenario which would just stay in place. There is a single additional read
         of the directory to locate the file.

      ii. Linked Allocation Scheme

         $2n + 2$ disk operations:
         1 read for the directory
         1 write to the directory to swap the head and tail pointers
         1 read and 1 write per block to swap the head and tail pointers

         Here we walk the list from head to tail swapping the previous and next pointers.
         Each block must be read and written one time and we must read and update
         the directory.

      iii. Indexed Allocation Scheme

         3 disk operations:
         1 read of the directory to locate the Inode
         1 read of the Inode
         1 write of the Inode to reverse the order of the block pointers

         Here we simply read the directory to locate the Inode, read the Inode to get the
         list of blocks it owns, and write the Inode with the list of blocks in reverse order.
b. Split Into Two

i. Contiguous Allocation Scheme

2 disk operations:
- 1 read of the directory to obtain the pointer and length
- 1 write of the directory with the 2 pointers and 2 lengths

ii. Linked Allocation Scheme

4 + n disk operations:
- 1 directory read to find the head and tail
- 1 read for each block from head and tail until we meet in the middle
- 1 write each for the new head and tail
- 1 directory write to update it with the new head, tail and filename

Here we read the directory to find the head and tail of the file, then read each block following the next and previous pointers to the middle. Once we meet at the middle, we write the new head and tail blocks to update their pointers and write the directory with the updated tail for the old file and the new file with head and tail.

iii. Indexed Allocation Scheme

7 disk operations:
- 1 directory read to find the inode
- 1 read of the inode to find the blocks
- 1 free list read to find a free block for the new inode
- 1 free list write to remove a block from the free list
- 2 inode writes (one per inode, original and new)
- 1 directory write to add the new inode

c. Merge

Assume file 1 has n blocks, file 2 has m blocks, and the files are not adjacent.

i. Contiguous Allocation Scheme

2(n+m)+2 disk operations:
- 1 directory read to find the file start and length for the two files and obtain free block bitmap
- 1 read and 1 write for each block in the two files to read them out of their original location and write to the newly allocated space (2(n+m))
- 1 write to update the directory and free bitmap
This algorithm locates the files, locates a new free space which is large enough to hold the unified file, and then copies both files there in sequence. Finally it writes the updated directory with the new filename, start and length, as well as an updated free bitmap.

ii. Linked Allocation Scheme

6 disk operations:
1. read of the directory to locate the files
2. read and 1 write for the tail block of one and the head block of the other to update the pointers to link to each other \(4=(1+1)^2\)
3. 1 write of the updated directory

Read the directory to find the tail of File1 and the head of file2. Then read each of those blocks. Write the tail of File1 with an updated next pointer which indicates File 2's head block and the head of File 2 with an updated previous pointer which indicates the tail block of File 1. Then write the updated directory.

iii. Indexed Allocation Scheme

7 disk operations:
1. read of the directory to locate the inodes
2. reads (1 of each inode) to locate the blocks
3. 1 write of the updated inode used for the merged file
4. 1 read and one write to place the freed inode in the free list
5. 1 write of the updated directory