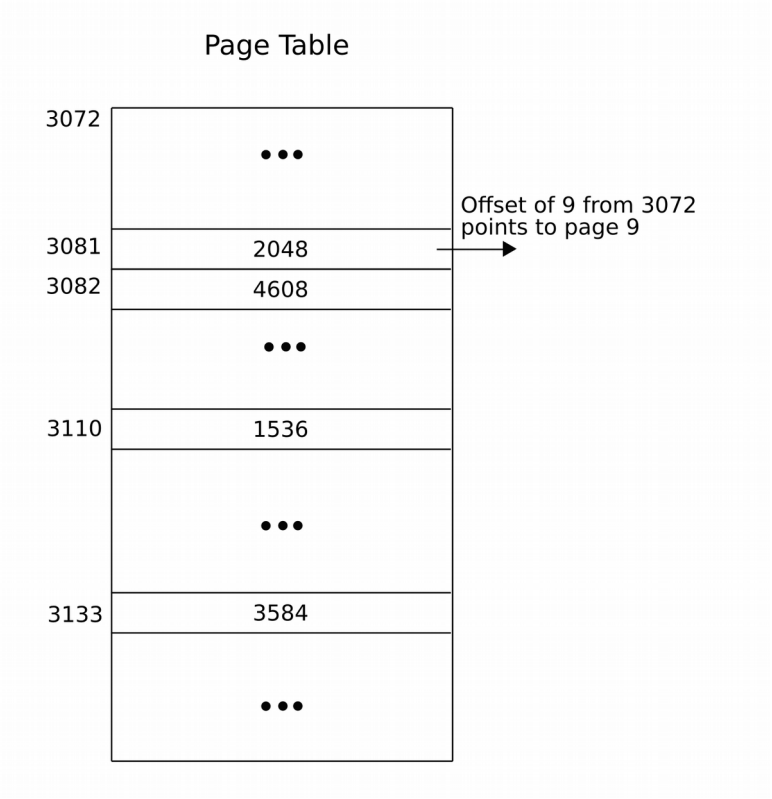
Assignment 4 – Written Answers

1. A minimum allocation size is required because the free list is stored directly in the memory that is allocated. This means that there must be at least room for a size pointer and a next pointer in an allocated spot should it ever be freed and a free list node needs to be put in its place.

2. a)
i) The page table will look like how it does in Figure 1 below.



(Figure 1: Page Table – Daniel Lovegrove)

ii) Since 31000 / 512 = 60.55, this means the process is trying to access page 60. Since page 60 is not in RAM, a **page fault** will occur. The page will need to be loaded from disk so that the program can access what it needs to.

iii) Table 1 shows the conversion from virtual addresses to physical addresses.

Virtual Address	Frame Number	Offset
4608	9	0
5119	9	512
5120	10	0
31240	61	8

(Table 1: Virtual to Physical Address Conversions – Daniel Lovegrove)

2. b) Table 2 illustrates the LRU algorithm for the page reference sequence specified in the assignment. Sucessive columns to the right indicate the flow of time. The “Most Recent” row after the initial state column indicates the page that was referenced.

Initial State

Least Recent	8	8	8	11	1	7	7	15	10	10	10
...	11	11	11	1	7	15	15	10	14	14	11
...	1	1	1	7	15	10	10	14	11	11	17
...	7	15	7	15	10	11	14	11	9	17	9
Most Recent	15	7	15	10	11	14	11	9	17	9	14
Fault?				**FAULT**		**FAULT**		**FAULT**	**FAULT**		

(Table 2: LRU Algorithm Visualization – Daniel Lovegrove)

3. a) The maximum file size allowed in the file system is 303,104 bytes. This is because there are 8 blocks addressable by a direct pointer, 8 by a single indirect, 8 * 8 = 64 by a double indirect, and 8 * 8 * 8 = 512 by a triple indirect pointer. Therefore, the total number of blocks multiplied by the block size is (8 + 8 + 64 + 512) * 512 = **303,104**.

3. b)
i) Exactly **one disk block** is needed. **No internal fragmentation**.

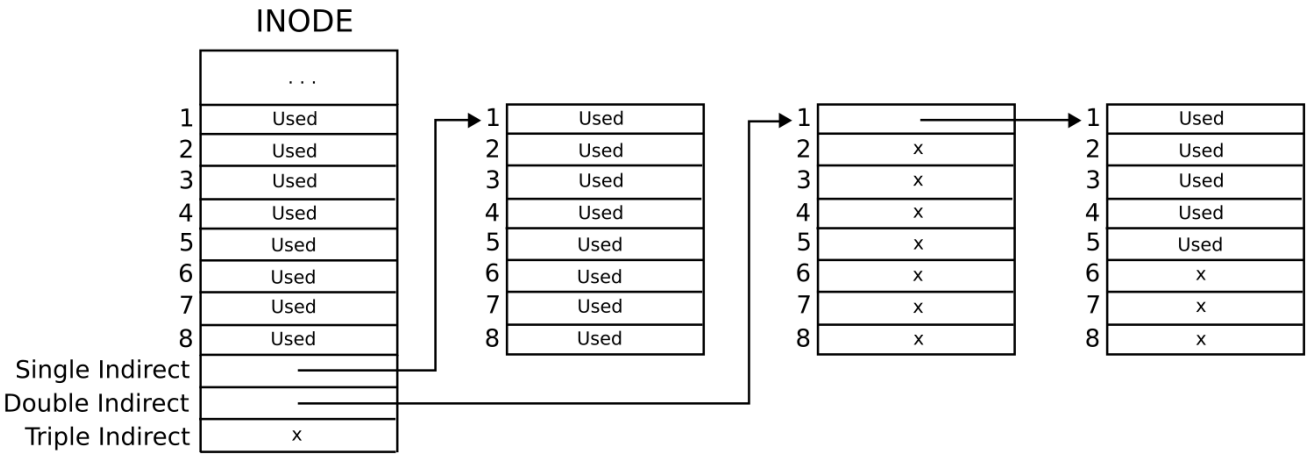
ii) **Two disk blocks** are needed. There are 512 – (516 – 512) = **508 bytes of internal fragmentation**.

iii) Since 512 divides 10752 evenly into 21, we know that there will be 21 disk blocks needed just for the data, each without any internal fragmentation. Figure 2 below shows what the inode structure looks like. All of the x’s indicate unused space. All of the index pointers will need to be stored in a disk block too; there will be 14 pointers in a block that can hold a maximum of 512 pointers. We end up finding that there are (21 + 1) = **22 disk blocks** needed, and (512 – 14) = **498 bytes of internal fragmentation**.

iv) 2134016 is larger than the maximum file size

v) 8401408 is larger than the maximum file size

3. c) 77824 / 512 = 152, so there are 152 blocks allocated to this file. The first 8 blocks take only one disk operation each to read, as they are pointed to by direct pointers. The next 8 require 8 * 2 operations to read, as the pointer has to be read from one disk block to find the correct disk block. The next 64 require 64 * 3 disk operations, adding another level of disk reading. The last 72 require 72 * 4 disk operations to read as they are referenced by a triple indirect pointer. Altogether, there are **432 operations required**. This is assuming that there is no caching done, and that as soon as a disk block is read, it is forgotten.



(Figure 2: Inode Visualization – Daniel Lovegrove)