**Task 1**

Size of Main Memory (user space): 4MB

Size of a Page: 4KB

Organisation of memory blocks:

* 16 blocks of 2 pages = 128KB
* 24 blocks of 4 pages = 384KB
* 48 blocks of 8 pages = 1536KB
* 32 blocks of 16 pages = 2048KB

120 Blocks in total = 4096KB, which encompasses the entire user space

**Algorithms**

**Free Memory Tracking**

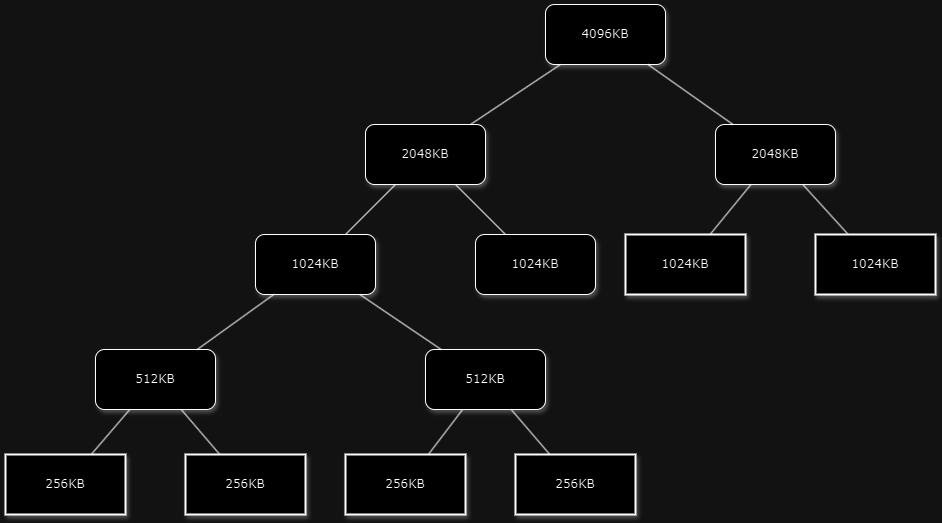
* Due to the fact that the memory is allocated in variable-sized blocks, the data structure that stores the address to the free blocks would need to be highly efficient.
* An algorithm with a fast lookup time (O(1) or O(log n)) would likely be needed to efficiently search the data structure for free blocks of memory.

**Linked Lists**

* Would contain objects that store:
  + First memory address
  + Size
  + Next object
* A linked list wouldn’t be efficient due to its O(n) search time.
* When a memory request is made, the linked list will be searched for a block of closest fit.
* Due to its lookup and search time, I won’t use a linked list, however it should be kept in mind as it could be used in addition to other data structures.

**Binary Trees**

* Binary trees would be far more efficient than Linked lists in terms of lookup and search times, providing O(log n) for worst-case lookup times.
* Each node in the binary tree would contain:
  + Left child
  + Right child
  + Parent
  + Start memory address
  + Size of block (based on the level)
* Each node would represent a block.
* Only the leaf nodes are blocks that are being used.
* The lowest possible level of the binary tree would be a level where the block size would be the size of a page, in this case 4KB.
* This data structure would work well with the buddy system in memory allocation, as it makes it very easy to split the blocks.



* In this diagram, there are 4 blocks of 256KB and 3 blocks of 1024KB, which adds up to 4096KB.
* When a request is made for 512KB, it will be given a block of 1024KB.

**Hash Maps**