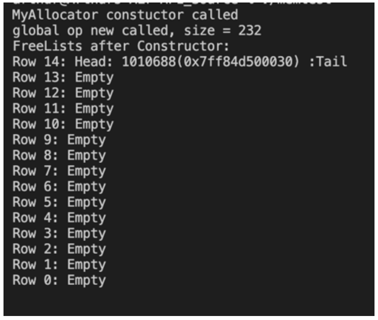
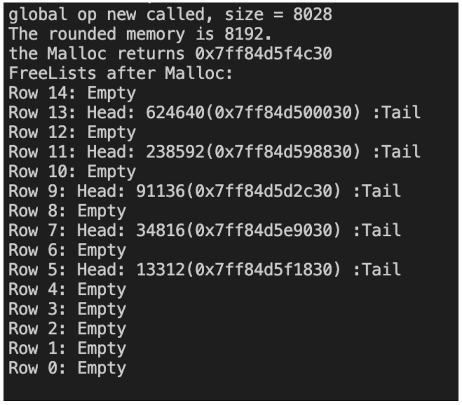
As we can notice in our current algorithm, the memory tends to break into large number of intervals.
For example:



```
MyAllocator constuctor called
global op new called, size = 232
FreeLists after Constructor:
Row 14: Head: 1010688(0x7ff84d500030) :Tail
Row 13: Empty
Row 12: Empty
Row 11: Empty
Row 10: Empty
Row 9: Empty
Row 8: Empty
Row 7: Empty
Row 6: Empty
Row 5: Empty
Row 4: Empty
Row 3: Empty
Row 2: Empty
Row 1: Empty
Row 0: Empty
```

FreeList has one big segment of memory, but with an allocation of size 8192, memory breaks into five parts.



```
global op new called, size = 8028
The rounded memory is 8192.
the Malloc returns 0x7ff84d5f4c30
FreeLists after Malloc:
Row 14: Empty
Row 13: Head: 624640(0x7ff84d500030) :Tail
Row 12: Empty
Row 11: Head: 238592(0x7ff84d598830) :Tail
Row 10: Empty
Row 9: Head: 91136(0x7ff84d5d2c30) :Tail
Row 8: Empty
Row 7: Head: 34816(0x7ff84d5e9030) :Tail
Row 6: Empty
Row 5: Head: 13312(0x7ff84d5f1830) :Tail
Row 4: Empty
Row 3: Empty
Row 2: Empty
Row 1: Empty
Row 0: Empty
```

Our current algorithm suffers from significate time complexity due to large number of fragmentations, because the iterator has to spend more time to iterate through the list.

I would suggest to manage the freed and occupied memory blocks through any sorting algorithms, such as red-black tree, and sort the memory blocks using O(N log N) time.

By sorting the memory blocks, we could possibly increase the utility of the memory. My code is aborted whenever the requested memory cannot be found in the free list.