Malloc Challenge

Naho Yokoyama

Implementing best-fit

Instead of choosing first slot (that has its size greater than or equals to the size we are looking for), I modified the program so that it chooses the slot that has the size closest to the size we are looking for (of course, its value should be greater, not less). The result are as follows, and it is clear that the utilization increased but since we have to go through all slots, the time increased a lot.

```
================================================
Challenge #1    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|               7 =>             1328
Utilization [%] |              70 =>               70
================================================
Challenge #2    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|              14 =>             1050
Utilization [%] |              39 =>               39
================================================
Challenge #3    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|             155 =>             1179
Utilization [%] |               9 =>               52
================================================
Challenge #4    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|           58944 =>            23217
Utilization [%] |              16 =>               72
================================================
Challenge #5    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|           52088 =>            16054
Utilization [%] |              15 =>               72
```

Implementing free list bin

Since the time of the best-fit version was long, I implemented free list bin to reduce the required for the program to find best-fit slot. First, I implemented a simple free list having a length 4 and grouping slots based on whether they are less than 1000, 2000, 3000, or more. The example on the slide uses “dummy” slot, but I thought that this would work without dummy so I deleted the dummy.

```
================================================
Challenge #1    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|               7 =>             1407
Utilization [%] |              70 =>               70
================================================
Challenge #2    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|              15 =>             1061
Utilization [%] |              39 =>               39
================================================
Challenge #3    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|             154 =>             1282
Utilization [%] |               9 =>               52
================================================
Challenge #4    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|           60699 =>            15294
Utilization [%] |              16 =>               72
================================================
Challenge #5    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|           52081 =>            12350
Utilization [%] |              15 =>               72
```

The reason why the time increased for challenge 1 to 3 is that the slots used in these challenges may all have the same size, so adding an index just requires more work. On the other hand, for challenge 4 and 5, since there are more varieties of sizes for slot, indexing worked better.

Improving free list bin#1

I noticed that when searching best-fit slot, the program searches all of the slots in the bin even if it already find the best fit. I thought that there are cases which I can stop searching in the middle, which is when I find the slots that has the difference of zero. Therefore, I implemented this, and the result shows that the time has been reduced a lot.

```
==================================================
Challenge #1    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|               7 =>              21
Utilization [%] |              70 =>              70
==================================================
Challenge #2    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|              16 =>              14
Utilization [%] |              39 =>              39
==================================================
Challenge #3    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|             159 =>              43
Utilization [%] |               9 =>              52
==================================================
Challenge #4    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|           58982 =>            2856
Utilization [%] |              16 =>              72
==================================================
Challenge #5    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|           54378 =>            4432
Utilization [%] |              15 =>              72
```

Improving free list bin#2

To improve the effectiveness of the free list bin, I modified the size of the bin and how it group different slots. I experimented so that what is the best free list bin size. Here, I evenly divide the slots to the bin.

```
==================================================
Challenge #1    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|               7 =>               9
Utilization [%] |              70 =>              70
==================================================
Challenge #2    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|              13 =>              14
Utilization [%] |              39 =>              39
==================================================
Challenge #3    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|             128 =>              28
Utilization [%] |               9 =>              52
==================================================
Challenge #4    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|           35964 =>             357
Utilization [%] |              16 =>              72
==================================================
Challenge #5    |   simple_malloc =>       my_malloc
--------------- + --------------- => ---------------
       Time [ms]|           30112 =>             535
Utilization [%] |              15 =>              72
```