Week 7　Malloc Challenge

Honoka Kobayashi

gist : https://gist.github.com/hono-mame/f7a5c218bff7f5a4dc5b8ceda2175c7e
100マス計算： https://drive.google.com/file/d/1Cma-OUOMjofYZZIYy_Hz4pnVyUiPUNJH/view?usp=sharing

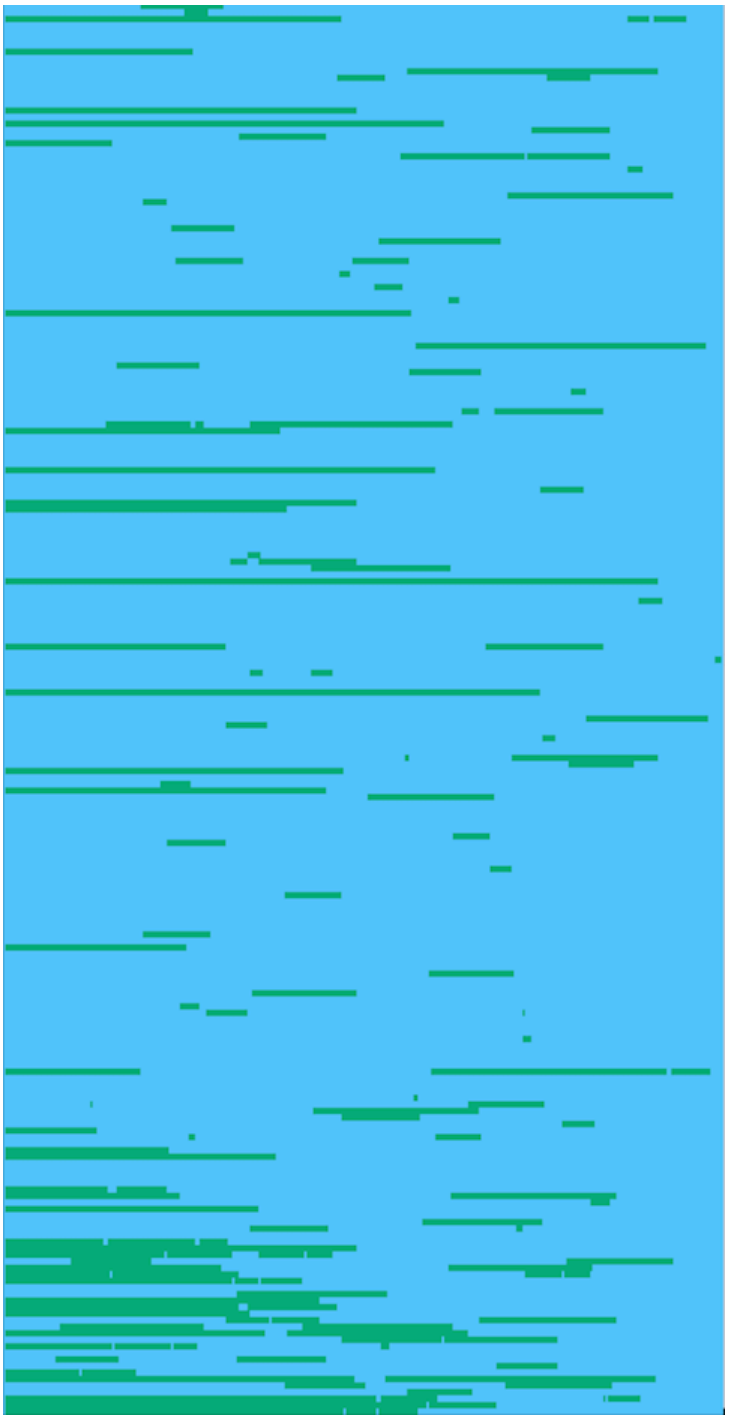
Performance improvements

First Fit

Result

```
========================================
Challenge #1    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|             9 =>             8
Utilization [%] |           70 =>            70
========================================
Challenge #2    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|             6 =>             6
Utilization [%] |           40 =>            40
========================================
Challenge #3    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|            82 =>            94
Utilization [%] |           9 =>             7
========================================
Challenge #4    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|         14799 =>         16573
Utilization [%] |           15 =>            15
========================================
Challenge #5    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|         14145 =>         14502
Utilization [%] |           15 =>            15

8,70,6,40,94,7,16573,15,14502,15,
```

Malloc Visualizer (challenge #5)



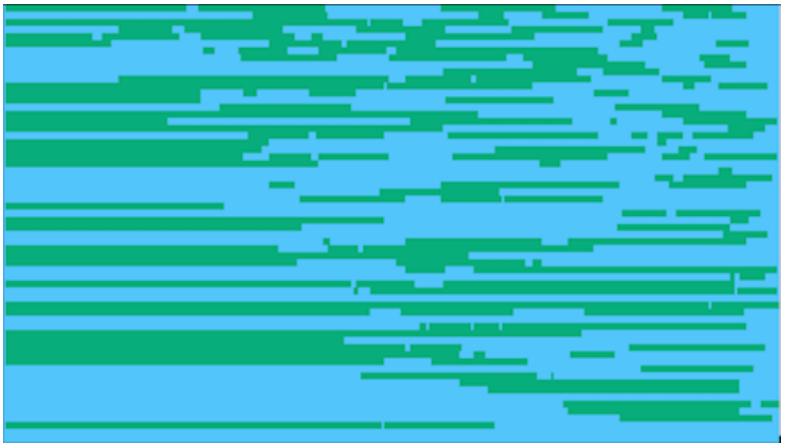
Best Fit

Result

```
========================================
Challenge #1    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|            12 =>           1015
Utilization [%] |            70 =>             70
========================================
Challenge #2    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|             4 =>            656
Utilization [%] |            40 =>             40
========================================
Challenge #3    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|            77 =>            766
Utilization [%] |             9 =>             50
========================================
Challenge #4    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|         15148 =>           7079
Utilization [%] |            15 =>             71
========================================
Challenge #5    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|          9857 =>           4206
Utilization [%] |            15 =>             74
```

1015,70,656,40,766,50,7079,71,4206,74,

Malloc Visualizer (challenge #5)



Consideration

- **Takes shorter time compared to first-fit (for challenge #4 and #5)**

If there is no free slot available, we need to request a new memory region from the system by calling mmap_from_system(). The result of malloc visualizer is much shorter compared to first-fit, which means we did not request a new memory region from the system.

→ **Calling mmap_from_system() requires so much time !**

Worst Fit

Result

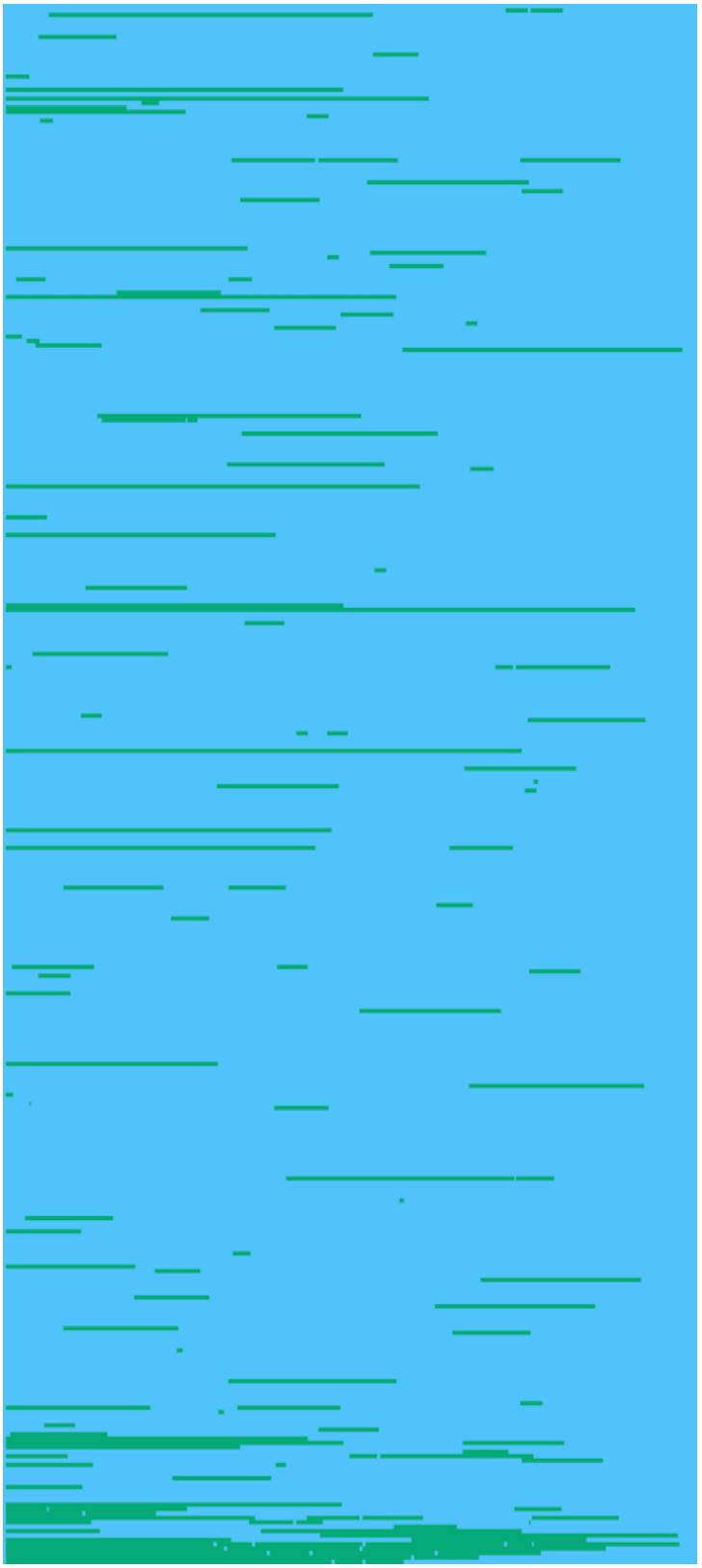
```
=====================================
Challenge #1    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|           12 =>           998
Utilization [%] |           70 =>           70
=====================================
Challenge #2    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|            4 =>           686
Utilization [%] |           40 =>           40
=====================================
Challenge #3    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|           83 =>         47963
Utilization [%] |            9 =>           4
=====================================
Challenge #4    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|        16344 =>         700148
Utilization [%] |           15 =>           7
=====================================
Challenge #5    |  simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|        11804 =>         502417
Utilization [%] |           15 =>           7

998,70,686,40,47963,4,700148,7,502417,7,
```

Consideration

- **Takes so much time compared to first-fit and best-fit**

This is because of the same reason as first-fit, but in this case, mmap_from_system() is called so much more time than first-fit and best-fit.
(Memory is allocated to the worst location, so there is no suitable location at the end of the trial. So, **green areas are concentrated at the end**.)

Malloc Visualizer(challenge #5)



Free list bin

① **Separate values every 1024**

free_list_bin[0] → size 0 ~ 1023
free_list_bin[1] → size 1024 ~ 2047
free_list_bin[2] → size 2048 ~ 3071
free_list_bin[3] → size 3072 ~ 4095

Result

```
==================================================
Challenge #1    |   simple_malloc =>      my_malloc    best_fit
--------------- + --------------- => ---------------
      Time [ms]|            12 =>          1006             1015
Utilization [%] |            70 =>            70               70
==================================================
Challenge #2    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|             4 =>           674              656
Utilization [%] |            40 =>            40               40
==================================================
Challenge #3    |   simple_malloc =>      my_malloc    best_fit
--------------- + --------------- => ---------------
      Time [ms]|            80 =>           818              766
Utilization [%] |             9 =>            51               50
==================================================
Challenge #4    |   simple_malloc =>      my_malloc    best_fit
--------------- + --------------- => ---------------
      Time [ms]|         18746 =>          4807             7079
Utilization [%] |            15 =>            72               71
==================================================
Challenge #5    |   simple_malloc =>      my_malloc    best_fit
--------------- + --------------- => ---------------
      Time [ms]|         11539 =>          3015             4206
Utilization [%] |            15 =>            75               74
```

Consideration

- **Takes less time compared to best-fit with no free_list_bin**

If the size is very large, we do not have to check the small one. By dividing the free_list, we do not have to check too small ones.

② **Separate values every 512**

free_list_bin[0] → size 0 ~ 511
free_list_bin[1] → size 512 ~ 1023
……
free_list_bin[8] → size 3072 ~ 3583
free_list_bin[9] → size 3584 ~ 4095

```
==================================================
Challenge #1    |   simple_malloc =>      my_malloc     best_fit
--------------- + --------------- => ---------------
      Time [ms]|            13 =>           1003
Utilization [%] |            70 =>             70
==================================================
Challenge #2    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|             4 =>            661
Utilization [%] |            40 =>             40
==================================================
Challenge #3    |   simple_malloc =>      my_malloc     best_fit
--------------- + --------------- => ---------------
      Time [ms]|            80 =>            782
Utilization [%] |             9 =>             51
==================================================
Challenge #4    |   simple_malloc =>      my_malloc    best_fit
--------------- + --------------- => ---------------
      Time [ms]|         19275 =>           2407
Utilization [%] |            15 =>             72
==================================================
Challenge #5    |   simple_malloc =>      my_malloc    best_fit
--------------- + --------------- => ---------------
      Time [ms]|         13020 =>           2021
Utilization [%] |            15 =>             75
```

1003,70,661,40,782,51,2407,72,2021,75,

③ **Separate values every 256**

```
==================================================
Challenge #1    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|            13 =>           1014
Utilization [%] |            70 =>             70
==================================================
Challenge #2    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|             4 =>            658
Utilization [%] |            40 =>             40
==================================================
Challenge #3    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|            85 =>            817
Utilization [%] |             9 =>             51
==================================================
Challenge #4    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|         16295 =>            213
```

```
Utilization [%] |          15 =>          72
==================================================
Challenge #5    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|         10110 =>         1114
Utilization [%] |          15 =>          75
```

1003,70,661,40,782,51,2407,72,2021,75,

🌟 It's quicker to break it up into smaller pieces. (?)

④ **Separate values every 1**

```
==================================================
Challenge #1    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|          11 =>          846
Utilization [%] |          70 =>          70
==================================================
Challenge #2    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|           4 =>          656
Utilization [%] |          40 =>          40
==================================================
Challenge #3    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|          75 =>          138
Utilization [%] |           9 =>          51
==================================================
Challenge #4    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|         17902 =>           30
Utilization [%] |          15 =>          72
==================================================
Challenge #5    |   simple_malloc =>      my_malloc
--------------- + --------------- => ---------------
      Time [ms]|         10003 =>           55
Utilization [%] |          15 =>          75
```

846,70,656,40,138,51,30,72,55,75,
(this is so fast, but uses large memory ??)

セグフォをデバッグした時のメモ

```
void *my_malloc(size_t size) {
 printf("%d",1); // for check
 int index = size / 1024;
 my_metadata_t *metadata = my_heap[index].free_head;
 my_metadata_t *prev = NULL;
 my_metadata_t *best_fit = NULL;
 my_metadata_t *prev_of_best_fit = NULL;
```

↑これで実行すると111111111111111,,,,,,で無限ループしてsegmentation fault

```
while (metadata) {
   if (metadata->size >= size && (!best_fit || metadata->size <
best_fit->size)) {
     // Update the best_fit and the prev_best_fit
     best_fit = metadata;
     prev_of_best_fit = prev;
   }
   prev = metadata;
   metadata = metadata->next;
 }
```

もしこのwhile文が実行されないと、次のif文に入ってしまって全く同じ関数が再帰呼び出しされる→だからセグフォになる

```
if (!best_fit) {
   size_t buffer_size = 4096;
   my_metadata_t *metadata = (my_metadata_t *)mmap_from_system(buffer_size);
   metadata->size = buffer_size - sizeof(my_metadata_t);
   metadata->next = NULL;
   // Add the memory region to the free list.
   my_add_to_free_list(metadata);
   // Now, try my_malloc() again. This should succeed.
   return my_malloc(size);
 }
```

ということは、best_fitに中身が必ずないといけない
→できるだけ値の小さなbin_listから、入れるところを探してbest_fitとprev_of_best_fitを更新し、NULLを防げばいい？そうすればbest_fitを実行できるようになる？ (Ryokoさんのコードを参考にしました🙇‍♀️)

```
while(index < 5){
 metadata = my_heap[index].free_head;
 prev = NULL;
```

```
while(metadata && metadata->size < size){
  prev = metadata;
  metadata = metadata->next;
}
 // Initialize the prev_of_best_fit and best_fit.
if(metadata){
  prev_of_best_fit = prev;
  best_fit = metadata;
  break;
}
// If we cannot find the place to put in, search for a larger list.
else{
  index++;
}
}
```

できた！
1006,70,674,40,818,51,4807,72,3015,75,