CSED211: Lab. 12

Malloc Lab.

박용곤, 조현욱 TA

csed211-ta@postech.ac.kr

POSTECH

2023.12.11

Table of Contents

What is malloc?

- malloc provide a simple and portable way to allocate/deallocate a memory block of desired size

- Linux kernel itself also provides very limited dynamic memory management primitives
  - **brk, sbrk** system call
  - Only expand/shrink the end of data segment (just like a stack)

- libc, a user-level library, implements malloc using those primitives

libc: malloc & free

- `void *malloc(size_t size)`
  - Allocate `size` bytes, and return a pointer to the address of the allocated object
  - `my_type *my_obj = (my_type *)malloc(sizeof(my_type));`
- `void free(void *ptr)`
  - Free the memory space pointed by `ptr`
  - `free(my_obj);`
- For more detail, https://linux.die.net/man/3/malloc

Challenges in malloc design #1: Execution Speed

- How fast can we:
  - Find a free memory block
  - Release a memory block

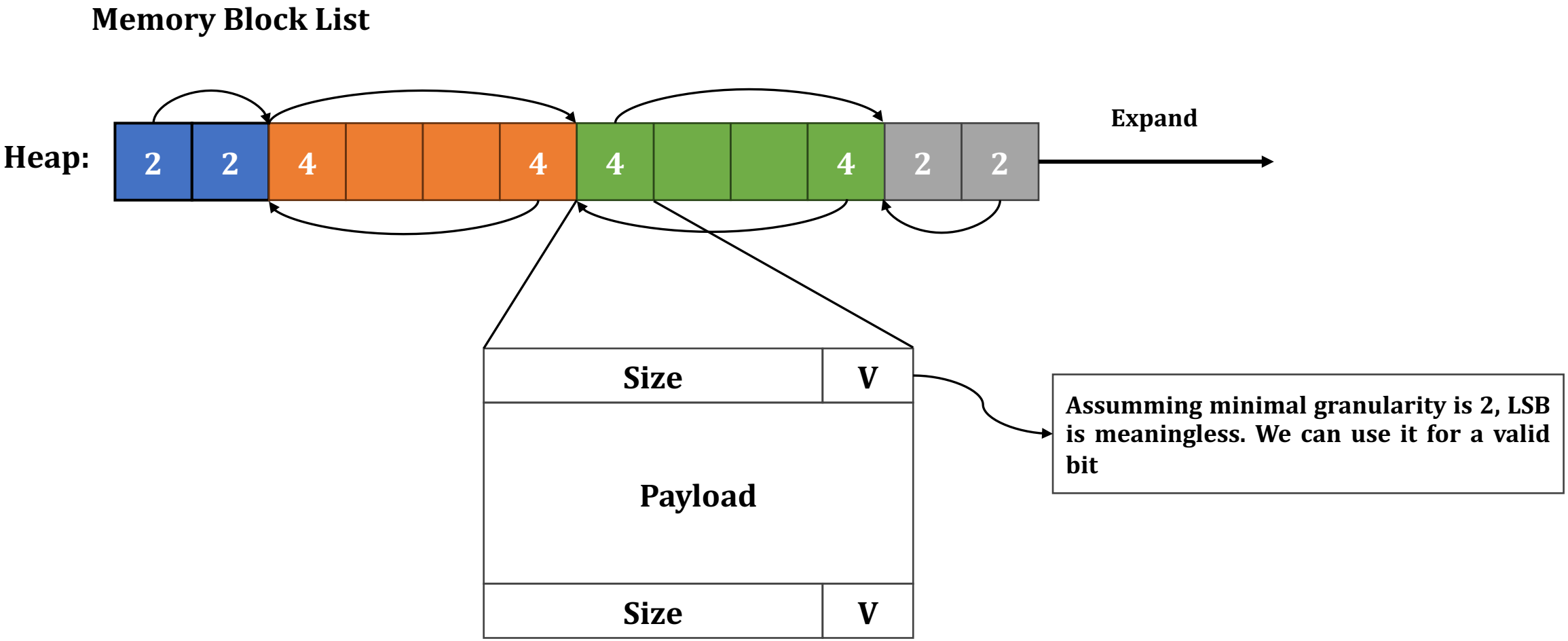
- Many kinds of algorithms
  - Bubble sort: O(n^2)
  - Merge sort: O(n log n)
  - Linear search: O(n)
  - Binary search: O(log n)
  - DFS/BFS: O(# of edges + # of vertices)

Challenges in malloc design #2: Memory Consumption

- N-byte malloc request consumes at least N-byte
- Data Structure Overhead:
  - Ex) Two words in doubly-linked list (**`next`** and **`prev`** pointer)
- Internal Fragmentation:
  - Ex) 3 Byte is requested, but returns 4 Byte (1 Byte wasted)
- External Fragmentation:
  - Ex) There is total 4 Byte of free memory, but increased the heap to satisfy 4 Byte malloc request (4-Byte wasted)

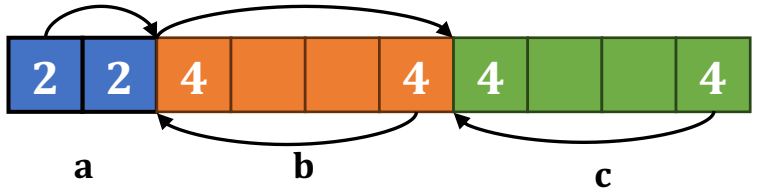
Example: Implicit Free List – Key Idea

- Use doubly-linked list to track memory blocks

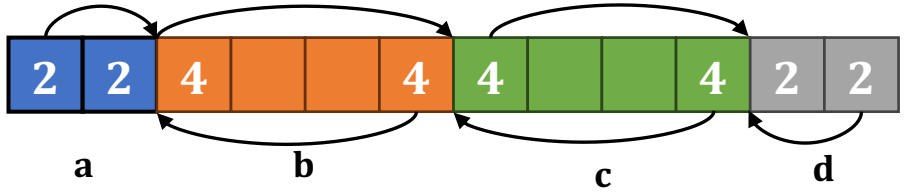


Example: Implicit Free List - Operations

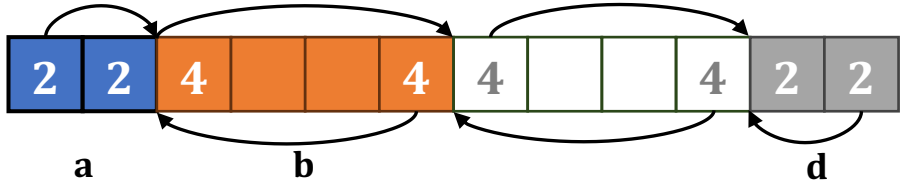
(1) Initial State



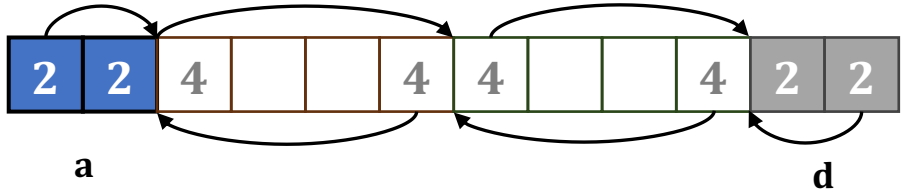
(2) `d = malloc(2);` Expand



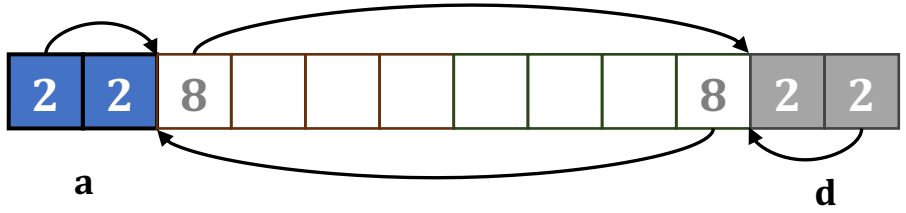
(3) `free(c);` Invalidate



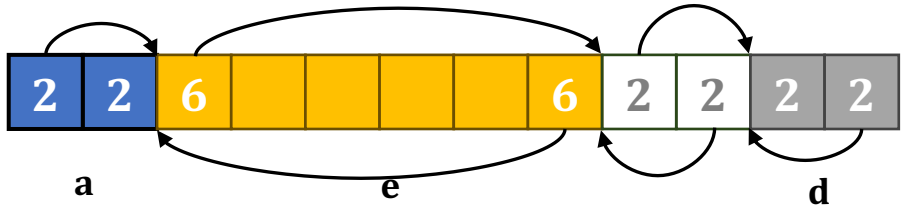
(4-1) `free (b);` Invalidate



(4-2) Merge free space (a.k.a. coalesce)



(5) `e = malloc(6);` Split



(You may find the first-fit or the best-fit)

Example: Implicit Free List - Evaluation

- Execution Speed
  - \+ free = O(1); set invalid & coalesce
  - \- malloc = O(# of memory blocks); linear search
- Memory Space Consumption
  - \+ Low internal fragmentation
  - \+ Small data structure overhead (2 words); next & prev displacement
  - \+ Low external fragmentation if best-fit
  - \- Severe external fragmentation if first-fit

Practice

Practice: libc malloc

- We will see how libc malloc works
- Compile and run **practice.c** using gdb
  - `gcc -g practice.c -o practice`
  - `gdb practice`
- Do not try to fully understand this implementation
  - It is very complicated, and you don't need it for assignment

```c
#include<stdio.h>
#include<stdlib.h>

int main() {
  void *f1 = malloc(16);
  printf("0x%x\n", f1);
  void *f2 = malloc(32);
  printf("0x%x\n", f2);
  void *f3 = malloc(16);
  printf("0x%x\n", f3);
  void *f4 = malloc(32);
  printf("0x%x\n", f4);
  printf("Stage 1\n");

  free(f1);
  free(f3);

  f1 = malloc(32);
  printf("0x%x\n", f1);

  free(f2);
  free(f4);

  return 0;

}
```

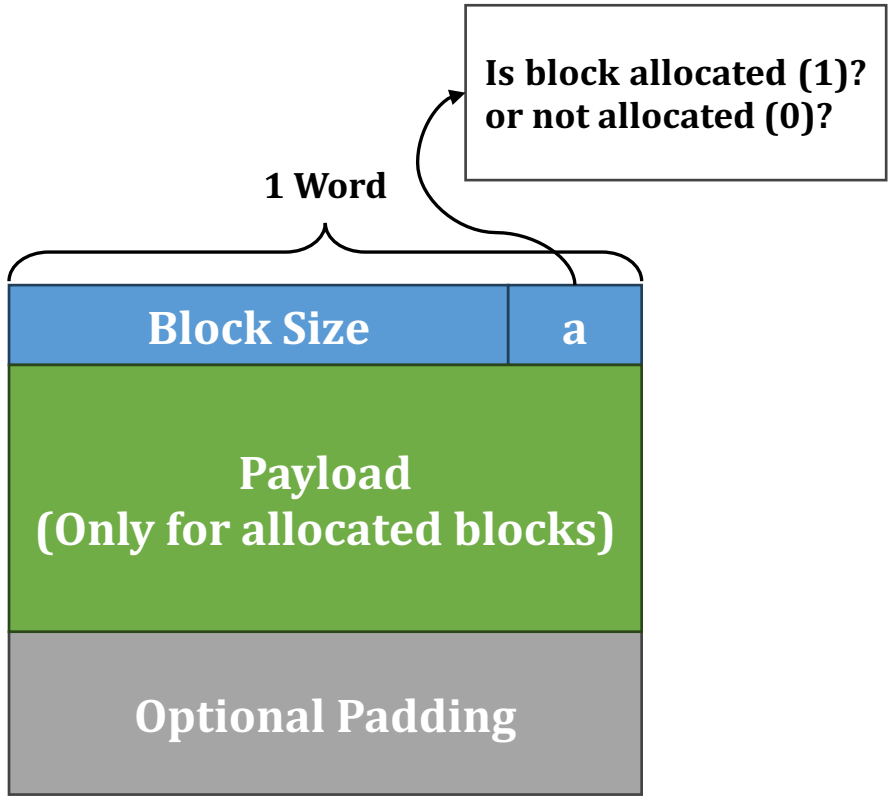
Practice: libc malloc

- libc malloc have a header, and the size is written
- Space overhead for each chunk is 0x10

Stage 1

```
(gdb) x/30gx 0x602010-0x10
0x602000:   0x0000000000000000  0x0000000000000021
0x602010:   0x0000000000000000  0x0000000000000000
0x602020:   0x0000000000000000  0x0000000000000031
0x602030:   0x0000000000000000  0x0000000000000000
0x602040:   0x0000000000000000  0x0000000000000000
0x602050:   0x0000000000000000  0x0000000000000021
0x602060:   0x0000000000000000  0x0000000000000000
0x602070:   0x0000000000000000  0x0000000000000031
0x602080:   0x0000000000000000  0x0000000000000000
0x602090:   0x0000000000000000  0x0000000000000000
0x6020a0:   0x0000000000000000  0x0000000000020f61
0x6020b0:   0x0000000000000000  0x0000000000000000
0x6020c0:   0x0000000000000000  0x0000000000000000
0x6020d0:   0x0000000000000000  0x0000000000000000
0x6020e0:   0x0000000000000000  0x0000000000000000
```

```
f1= malloc (16);
f2 = malloc (32);
f3 = malloc (16);
f4 = malloc (32);
```



Quiz

Stage 2

```
(gdb) x/30gx 0x602010-0x10
0x602000:   0x0000000000000000  0x0000000000000021
0x602010:   0x0000000000000000  0x0000000000000000
0x602020:   0x0000000000000000  0x0000000000000031
0x602030:   0x0000000000000000  0x0000000000000000
0x602040:   0x0000000000000000  0x0000000000000000
0x602050:   0x0000000000000000  0x0000000000000021
0x602060:   0x0000000000000000  0x0000000000000000
0x602070:   0x0000000000000000  0x0000000000000031
0x602080:   0x0000000000000000  0x0000000000000000
0x602090:   0x0000000000000000  0x0000000000000000
0x6020a0:   0x0000000000000000  0x0000000000020f61
0x6020b0:   0x0000000000000000  0x0000000000000000
0x6020c0:   0x0000000000000000  0x0000000000000000
0x6020d0:   0x0000000000000000  0x0000000000000000
0x6020e0:   0x0000000000000000  0x0000000000000000
```

a. 0x602000

b. 0x602050

c. 0x6020a0

```
f1 = malloc(16); free(f1);
f2 = malloc(32);
f3 = malloc(16); free(f3);
f4 = malloc(32);
```

```
f1 = malloc(32);
printf("0x%x\n", f1);
```

➔ What is the output?

a. 0x602000
b. 0x602050
c. 0x6020a0
d. None of the above

Homework

Due: 12/24, 23:59:59

What to do

- You need to complete five functions defined in `mm.c` using support routines
  - A detailed explanation is in the writeup file
  - `mm_init`
  - `mm_malloc`
  - `mm_free`
  - `mm_realloc`
  - \+ `mm_check` (for debugging, disable when you submit)

- Complete `mm.c` file and run:
  - `make`
  - `./mdriver` (use `-V` option to get more information)

- Your code will be evaluated in terms of (see 9. Evaluation section in writeup file)
  - Execution speed (disable `mm_check` for performance, but still `mm_check` will be graded)
  - Memory space consumption
  - Correctness & style

Grade Policy

- Correctness (20 points)
- Performance (35 points)
  - 5-point deduction for every 10 performance index
  - E.g., 35 points if 100 ~ 91, 30 points if 90 ~ 81, …
- Report (35 points)
  - Will be graded similarly to Data Lab 2
- Style (10 points) – part of report
  - Heap consistency checker: `mm_check` (5 points)
    - Explain your heap consistency checker in your report with **"attached code"**
  - Program structure & comments (5 points)
    - Explain your program structure in your report with **"attached code"**

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000214 26558
 1       yes   98%    5848  0.000216 27087
 2       yes   98%    6648  0.000268 24778
 3       yes   99%    5380  0.003503  1536
 4       yes   99%   14400  0.000259 55556
 5       yes   95%    4800  0.000423 11342
 6       yes   94%    4800  0.003713  1293
 7       yes   55%   12000  0.000268 44726
 8       yes   51%   24000  0.002734  8779
 9       yes  100%   14401  0.000193 74810
10       yes   87%   14401  0.000229 62941
Total          89%  112372  0.012020  9348

Perf index = 53 (util) + 40 (thru) = 93/100
```

Grade Policy

- Total 26 traces to grade your malloc
  - Traces are in the /traces directory
  - Each trace will have the same weight

```
[nanimdo@programming2 traces]$ ls
Makefile          cccp.rep             gen_binary2.pl     realloc-bal.rep
README            checktrace.pl        gen_coalescing.pl  realloc.rep
amptjp-bal.rep    coalescing-bal.rep   gen_random.pl      realloc2-bal.rep
amptjp.rep        coalescing.rep       gen_realloc.pl     realloc2.rep
binary-bal.rep    cp-decl-bal.rep      gen_realloc2.pl    short1-bal.rep
binary.rep        cp-decl.rep          random-bal.rep     short1.rep
binary2-bal.rep   expr-bal.rep         random.rep         short2-bal.rep
binary2.rep       expr.rep             random2-bal.rep    short2.rep
cccp-bal.rep      gen_binary.pl        random2.rep
```

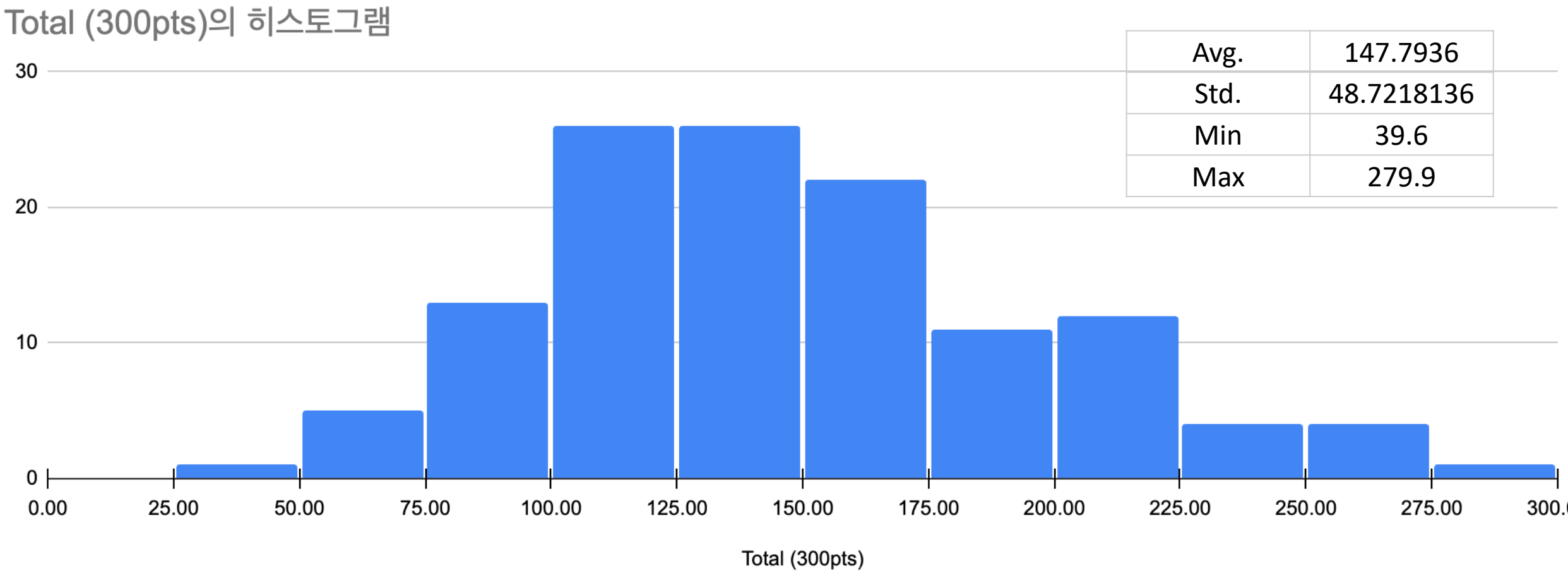
Rules (0 point if violated)

- Submission file format
  - Code: [student_id]_mm.c (e.g., 20234567_mm.c)
  - Report: [student_id].pdf (e.g., 20234567.pdf)
  - DO NOT COMPRESS THE FILES
- Your code should compile & run successfully
- Return pointers of your allocator must be 8-byte aligned
- Do not:
  - Change the interface in `mm.c`
  - Use any memory-management related library or system calls
    - Except provided `malloc, calloc, free, realloc, sbrk, brk`
  - Define any global/static compound data structure in `mm.c`
    - E.g., arrays, structs, trees, lists, …
    - You can define global/static scalar data structure such as int, float, pointer, …

Q & A

Midterm Exam Statistics

- Personal score will be announced very soon:
  - DO NOT ASK WHEN YOUR SCORE WILL BE ANNOUNCED



Avg.	147.7936
Std.	48.7218136
Min	39.6
Max	279.9