I. The implement allocation policy

1. In order to store the heap using situation, I keep track of free list.

- Every time a pointer is freed, the corresponding block is added to the list.
- When it is reallocated, it will be kicked out of the list.
- There is always a start pointer and end pointer point at the start of the free list and end of the free list.
- The start free block will not be reallocated so the address is fixed. And the end pointer will be changed every time new node is added.

2. I use four fields to store the metadata information, which are:

- byte_num: indicating the size of allocated data
- isfree: recording whether current block is free or occupied
- next: indicating where the next free block start
- data: the returned address when calling malloc

3. Before try to call sbrk() the size+sizeof(metadata_t) to allocate the heap memory, first use two methods to check if there is freed memory that can fit in the new need.

■ First-Fit

- Traverse the link, when the current node is freed and the byte_num is smaller than required size, return the address of current block
- Also, if the remained space is larger than the size of metadata, then split the current block into an occupied block and another freed block.

■ Best-Fit

- Traverse the whole link, when the difference between byte_num of current block and size is the smallest, return the address of that place.

  Here, one place that needs to be paid attention to is that we need to make sure the loop break when current block size equal to size or the it will take long time for equal size allocation to look for.
- Also, if remained space is larger than the size of metadata, then splitting the current block into a freed block in the front and an occupied block at the end.

4. When free the block, I also check whether the block that is physically on the right side of it is in the free list. If it is in the list, then traverse the free link to find that block and merge the current block with it.

II. Results and analysis

1. Execution Time

The searching process will take most of time, like the process to search for fitting place and time to find if it is able to merge.

As we can see, overall, the execution time for first fit policy is better than best fit policy. This is because generally we need to search the whole list to find the best fit position while it is easier to search for the first place that can fit in current data.

Besides, the time it takes for equal is the least. I think this is because since the chunks' size are all the same, so it takes a little time to search for the fit place and also since it is free from left to right, the right block's isfree is always false, which means no need to iterate through the link.

I think the reason for executing large range random blocks are so much longer is that the size between different blocks are too much. There is great chance for a new block that wants to come in to search for whole list without any result.

```
aq23@vcm-12467:~/ece650/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
data_segment_size = 3608672, data_segment_free_space = 78016
Execution Time = 2.858262 seconds
Fragmentation  = 0.021619
aq23@vcm-12467:~/ece650/my_malloc/alloc_policy_tests$ ./equal_size_allocs
Execution Time = 1.701159 seconds
Fragmentation  = 0.450000
aq23@vcm-12467:~/ece650/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
Execution Time = 229.153123 seconds
Fragmentation  = 0.059991
```

Figure 1. Result for first fit

```
aq23@vcm-12467:~/ece650/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
data_segment_size = 3608672, data_segment_free_space = 78016
Execution Time = 2.869036 seconds
Fragmentation  = 0.021619
aq23@vcm-12467:~/ece650/my_malloc/alloc_policy_tests$ ./equal_size_allocs
Execution Time = 1.841031 seconds
Fragmentation  = 0.450000
aq23@vcm-12467:~/ece650/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
Execution Time = 230.013034 seconds
Fragmentation  = 0.059991
```

Figure 2. Result for best fit

2. Fragmentation

Overall, the fragmentation is relatively low, which is good because it means compared with all the space that is malloced, free space is quite small owing to utilizing the space efficiently. For small because the size difference is small, it's reasonable to have better utilization of space than large range random blocks.

Because in the source code of equal size allocation, it shows the record take place half way through, so the result is very much close to 0.5.

3. Conclusion

From my standpoint, first fit has better result in execution time and best fit has better result in space utilization.