1. Allocation Policy

In my malloc implementation, I use a doubly linked list to store all the free blocks. And all free blocks in list are sorted by the address of them.

The malloc & free functions are implemented as following:

ff_malloc(size) : Loop over the list to find the first fetch block, remove it from the list. If there no block big enough, ask system to allocate one. Split the block into 2 if necessary. Return the first of them whose size is exactly the input. And insert the second block back to the list.

bf_malloc(size) : Loop over the list to find the best fetch block, remove it from the list. If there no block big enough, ask system to allocate one. Split the block into 2 if necessary. Return the first of them whose size is exactly the input. And insert the second block back to the list.

ff_free(ptr): Insert this block back to free list. Check if it is adjacent to its previous or next block. If adjacent, merge the two blocks into one.

bf_free(ptr): Insert this block back to free list. Check if it is adjacent to its previous or next block. If adjacent, merge the two blocks into one.

2. Performance Analysis

Executation Time (s)

	small_range_rand_allocs	equal_size_allocs	large_range_rand_allocs
ff_malloc	15.51	17.32	64.1
bf_malloc	4.33	17.53	71.18

Framentation

	small_range_rand_allocs	equal_size_allocs	large_range_rand_allocs
ff_malloc	0.074	0.45	0.093
bf_malloc	0.027	0.45	0.040

In the process of developing this function, I had tries other two different policies. The first one is applying a singly linked list. However, for the singly linked list, it is necessary to loop all over the list to find the tail for new allocated block from system. And the second policy is keeping both lists for free block and used block. This policy obviously doubles the executation time. So, I abandon it after knowing that we are not responsible for invalid pointer input. Because if we need to check the validity of input pointer for free function, we need to keep track of the blocks we allocated.