Kevin Tang
CSCE 313

PA1: Buddy Allocator Analysis

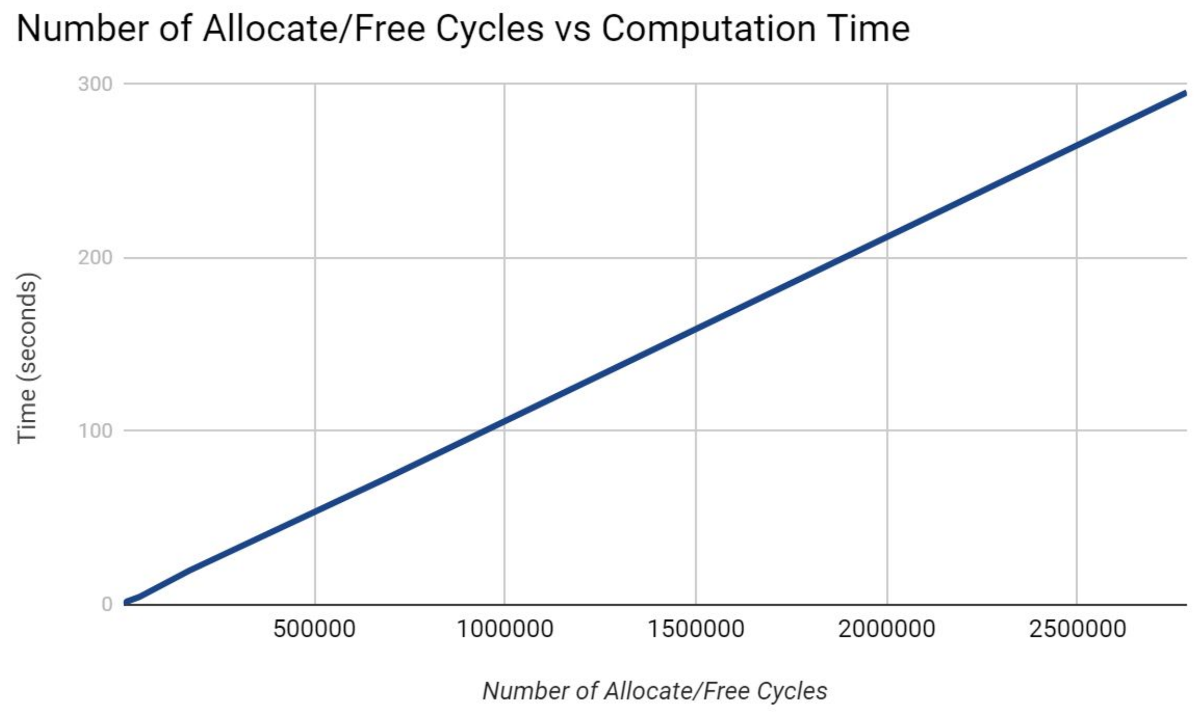
Running the program:
To run you need to simply navigate to the directory of the project in linux terminal and type

```
make
./memtest -b <Basic Block Size> -s <Total Memory Size>
```

If you don't give any of these arguments then the program will default to block size = 128 (bytes) and total memory size = 128 * 1024 * 1024 (bytes). Also if the input block size or total memory size is not a power of 2, then the true input will be adjusted to the smallest power of 2 that is larger than the given inputs.

Data:
Ackermann Algorithm Resulting Time (m and number of allocate/free cycles vs time in seconds)
*n is set to 3 for all tests

M Value	Number of allocate/free cycles	Time
M = 1	106	0.009007
M = 2	541	0.050922
M = 3	2432	0.241864
M = 4	10307	1.98609
M = 5	42438	4.515956
M = 6	172233	19.629736
M = 7	693964	73.422335
M = 8	2785999	295.426881



Analysis:

It is clear that based off of the data that the computation time is O(n) where n is the number of allocate and free cycles. This indicates that our allocate and free functions amortized cost will be constant time per call, which is very efficient. Some bottlenecks or poor implementation that can affect how much each constant time cost is determined on number of merges or splits called on memory. Worst case scenario for a alloc call is when you need to split the largest memory block to the smallest memory block, and worst case for free call is when you need to merge the smallest memory block into the largest memory block size.

These worst case scenarios can't be removed due to the requirement that the largest memory block needs to be available at all times so that our buddy allocator is able to provide memory to a larger range of requests. Also the splitting of memory blocks can't be avoided because you want to provide the smallest memory block to save larger blocks for other alloc requests. Another bottleneck can be identified in the linked list. The linked list used in the project includes both free and used block headers. So worst case is when the headers in the linked list has many block headers that are not free. The program will have to iterate to check all the block headers in the linked list for one. A better implementation is to make the linked list only include block headers that are free.

Overall, the buddy allocator is pretty efficient in amortized time but can have worst case scenarios where it is logarithm time for one alloc or free call.