Leilani Horlander-Cruz

CSCE 313-504

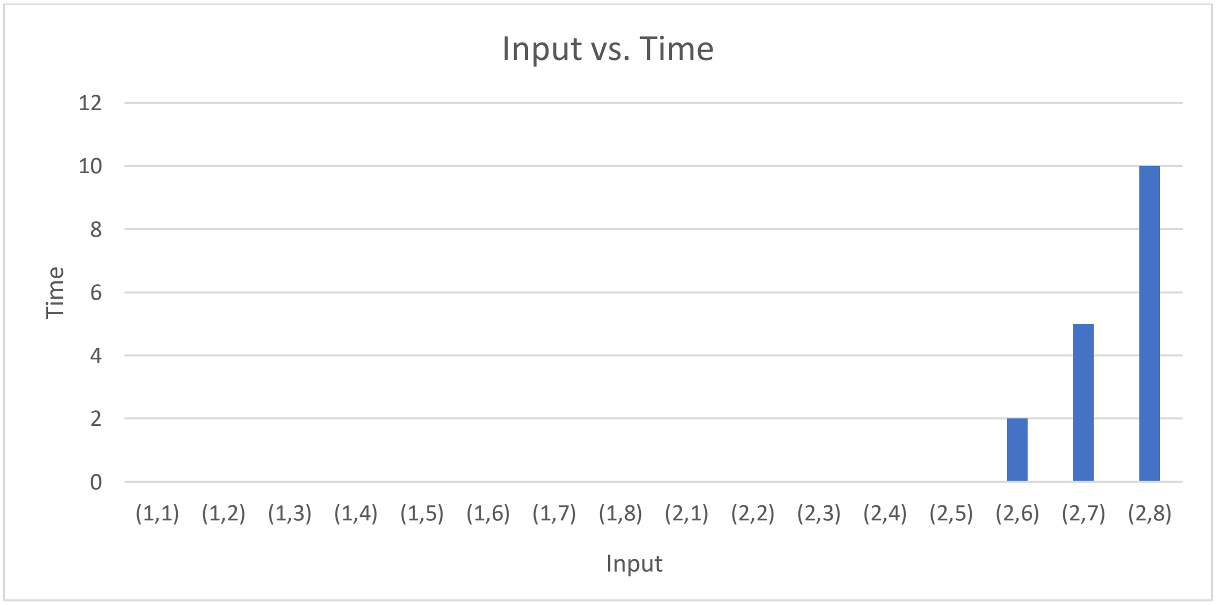
10-1-17

Milestone 3 Analysis

The implementation of the program performs well until the ackerman function calls for a and b larger than 2 and 5, respectively. This is due to the high recursive runtime of the function. The memory management implementation is very fast in terms of speed, but it wastes a lot of memory internally (internal fragmentation). It wastes the least amount of memory when searching for blocks that are a size of a power of 2 minus the size of a free list header. However, oftentimes, this function will give too much space for the size needed, especially when searching for a block size less than the basic block size, since this is the minimum size that can be allocated at any given time. The longest functions in the implementation of the program are breakdown and checkBuddy because, in the worst case, they will iterate across the entire array.

In order to minimize memory wasting, reducing the size of free list headers would allow for less memory to be wasted during each allocation of memory. Another way to do this would be not to store the headers in the allocated memory, which would allow the user to have access to all of the memory they request in the allocation call, despite using up more memory overall.

In order to improve performance of the implementation, I would add as many power of 2 blocks as possible in init_allocator, allowing the user to access more memory closer to their request, since this is the way the implementation works best. checkBuddy will bring some challenges with this, however, since each block allocated inside of init_allocator can only be a buddy with itself. Attempting to do otherwise will result in segmentation faults.


Input vs. Time
Time
(1,1)
(1,2)
(1,3)
(1,4)
(1,5)
(1,6)
(1,7)
(1,8)
(2,1)
(2,2)
(2,3)
(2,4)
(2,5)
(2,6)
(2,7)
(2,8)
Input