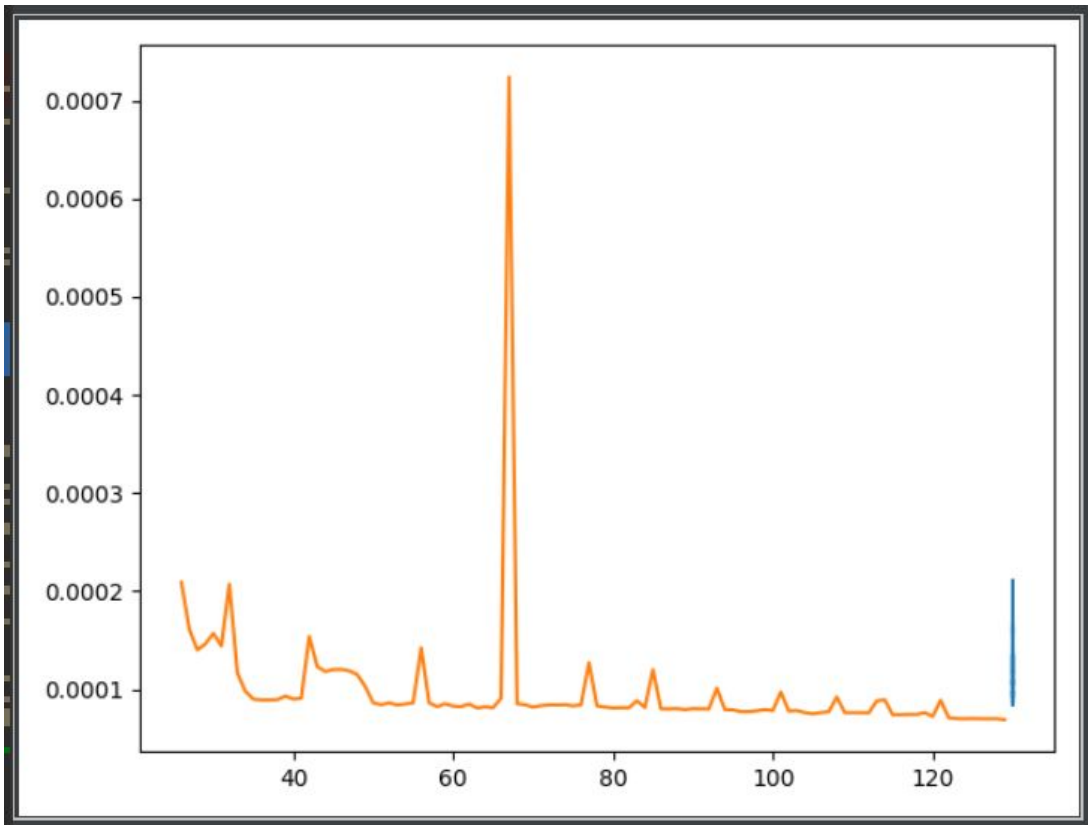
Alex Barajas-Ritchie
Alondra Lona
CS 415
Dr. Gil

Task 3 Written Report

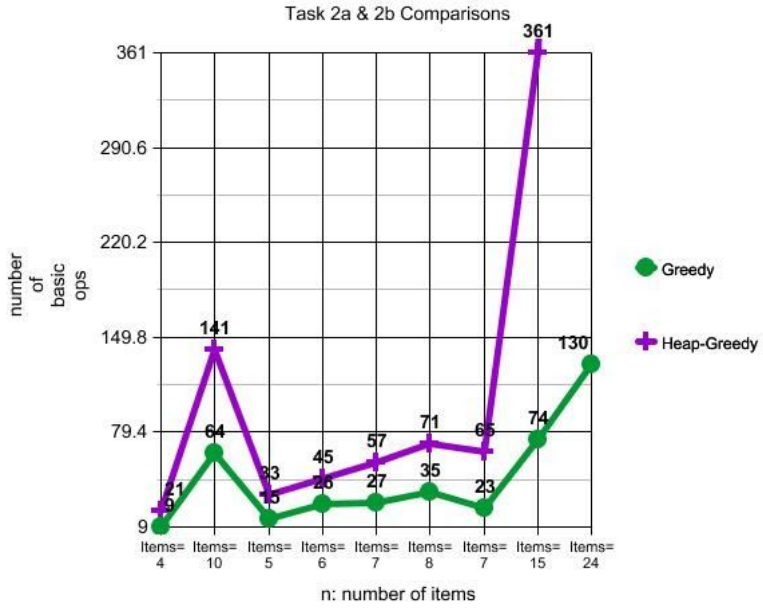
Task 3: 1a vs 1b

We were able to determine the value of K by observing the behavior of how the hashing was distributed during the storing of each F(I, J). we can see from the chart that there is in fact a sweet spot for the value of K



The blue line indicates the constant space taken up by Task1A. The table will be a constant i + j size. For this particular graph we see that Task1A took up a space of 130. Our K for Task1B slowly approached 130. We see that run time of the task1b implementation slowly decreases as it approaches the size of (i * j). This illustrates the space time trade off of the algorithm.

Task3: 2a vs 2b

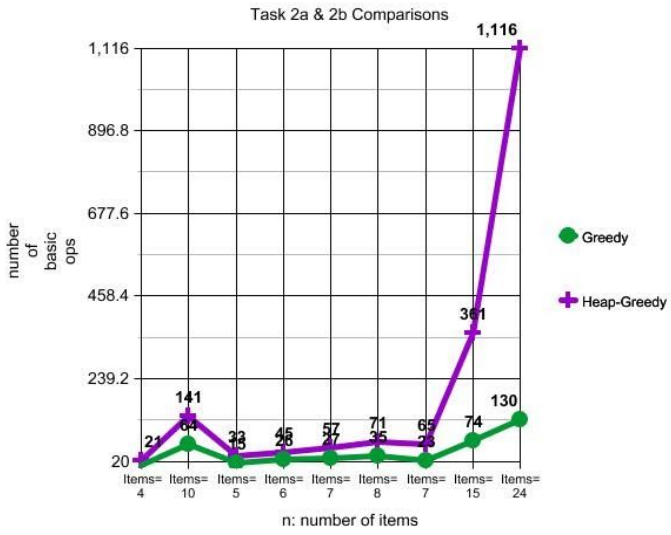


Our number of basic operations are comparisons.

In the graph above and below, the x-axis corresponds with each file. For example, Items=4 -> file0, Items=10 -> file1, Items=5 -> file2, etc.

I implemented quicksort which has a complexity of O(nlogn), while heapsort has O(n + k log n), where k is the number of objects that are selected by the greedy algorithm.

We can also see when we use file 8, and we have 24 items, our Heap-Greedy calculates 1116 comparisons, which in this graph we don't see, but I have included this graph where we are able to see where it falls.



Additionally, I have arranged the items in increasing n values, which are the number of items in our knapsack and can see the curves clearer. In the bottom graph, we are able to see that using a non-heap greedy approach, is way more efficient than using a heap. That is because it is an inplace sorting algorithm and when we are creating our partition for task 2a, it yields less comparisons. We are able to use the partitions and that will result in two arrays, and one of them will be sorted.