Josh Bell

CS – 325

April 23rd, 2021

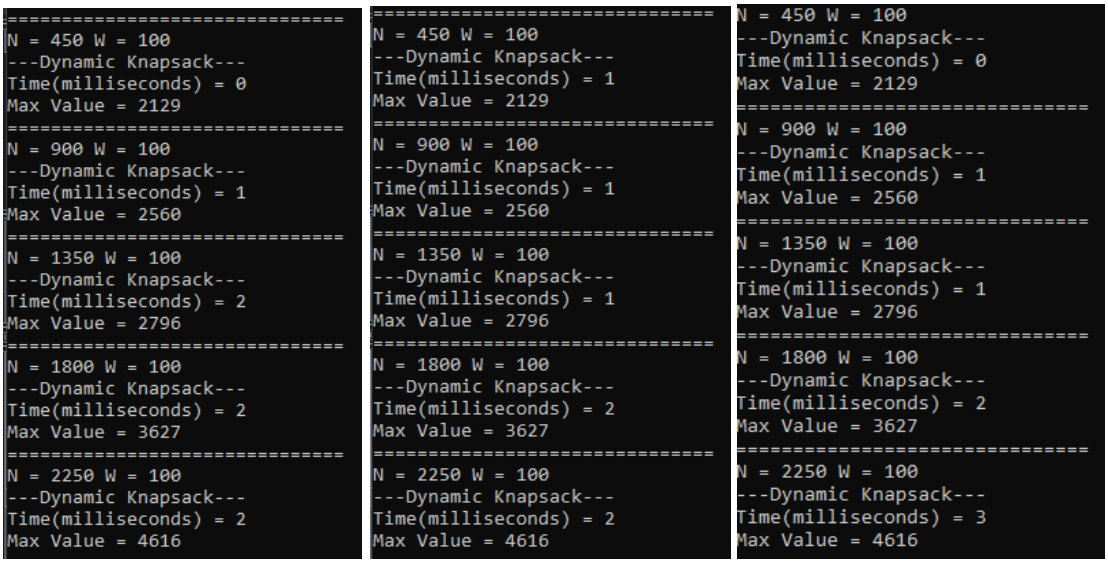
Homework 2

GitHub Repo for this assignment: https://github.com/JoshBell302/CS325-AoA-Assignment-2

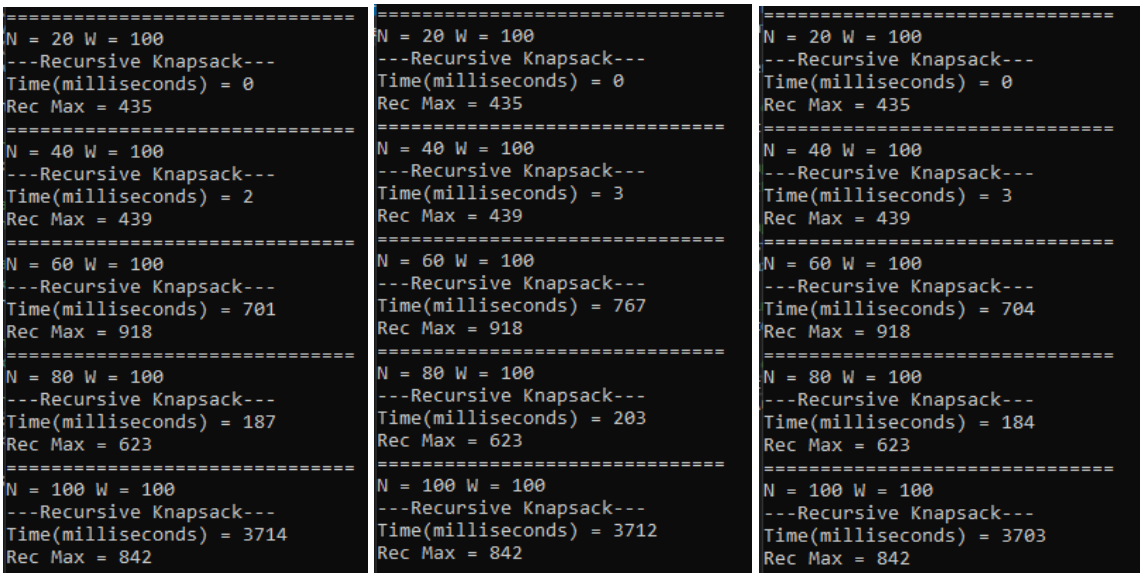
Problem 1: 0-1 Knapsack – Recursive vs. DP

Given weights and values of n items, select items to place in a knapsack of capacity W to maximize the total value in the knapsack. That is, given two integer arrays val[] and wt[ ] which represent values and weights associated with n items respectively and an integer W which represents knapsack capacity, determine the maximum value subset of val[] such that sum of the weights of this subset is ≤W. Items cannot be broken or used more than once, you either select the complete item, or don't select it. Implement both a recursive and dynamic programming algorithm to solve the 0-1 knapsack problem. Both algorithms should return the maximum total value of items that can fit in the knapsack.

a) **Implement both the recursive and dynamic programming algorithm in one program named knapsack (.cpp, .c or .py). Your program should randomly generate test cases that are solved using both the DP and Recursive algorithm. The program should output to the terminal: n, W, time for the DP algorithm, max for the DP, time for the Recursive algorithm, max for Recursive. The max values should be the same.**
   Code can be found on my GitHub Repo, link is above.
b) **Conduct experiments to collect running times for randomly generated input. Since there are two variables, n and W, you can hold one constant while varying the other and vis-a-versa. This may result in several graphs. If the recursive algorithm is too slow you can collect data using different values of W and n. plot the data, calculate the best fit equation and graph the best fit curves.**
   I had to have the values for 'N' be much larger for DP then Recursive due to the speed at which DP ran compared to Recursive. I ran three tests for each method…
   DP with N = 450, 900, 1350, 1800, 2250 and W = 100



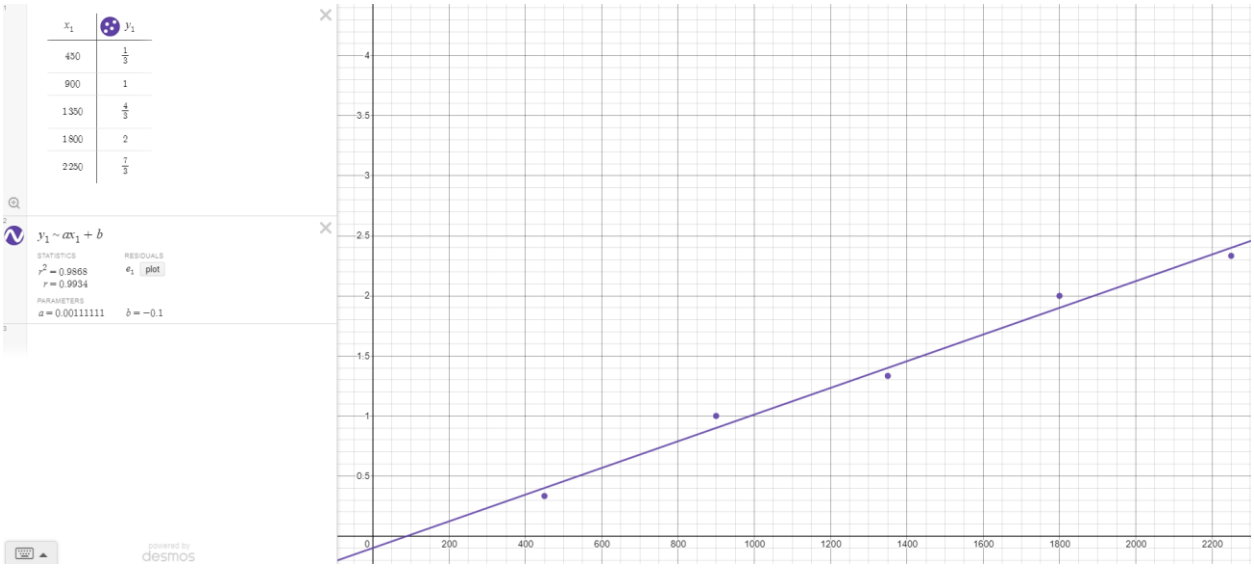
Rec with N = 20, 40, 60, 80, 100 and W = 100



When taking the averages of each it results in these tables…

DP

N	Time
450	1/3
900	1
1350	4/3
1800	2
2250	7/3

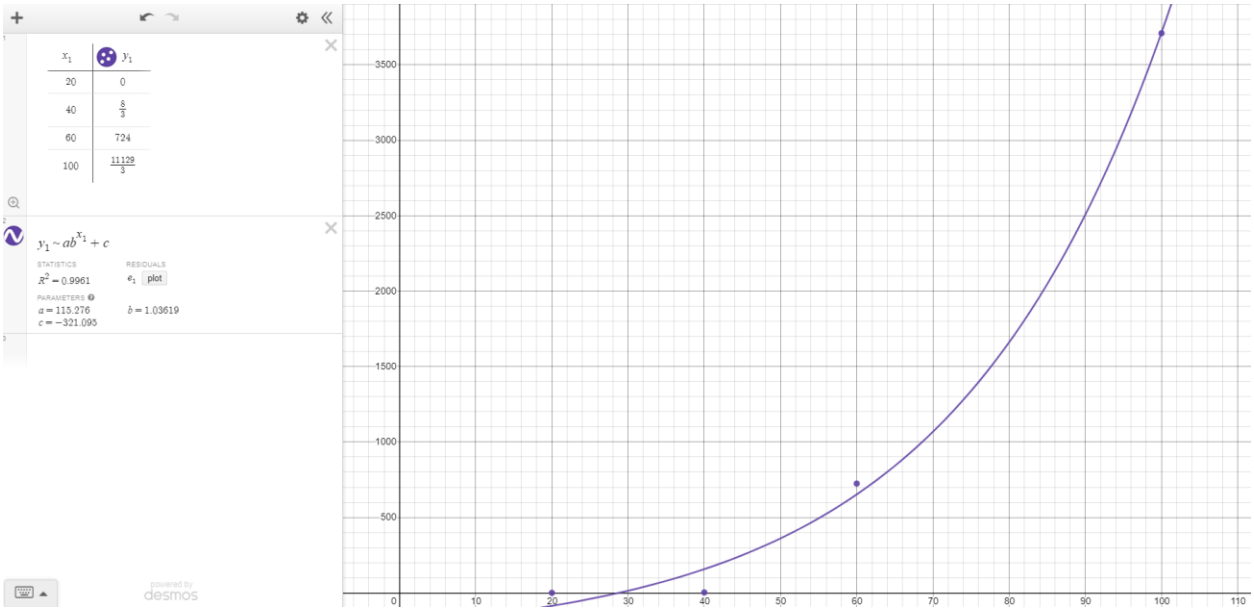


Best fit equation is: $y = 0.0011x - 0.1$ with a R^2 value of 0.9868

Rec

N	Time
20	0
40	8/3
60	724
80	574/3
100	11129/3

Notes on N=80 for Rec: Some weird randomization error occurs at this level, which will cause an outlier in the graph so I will be deleting this value when finding the best fit line.



Best fit equation is: $y = 115.276(1.03619^x) - 321.095$ with a R^2 value of 0.9961

c) **Discuss your implementation, results and how you collected the data. How does W change the running time?**
When creating the code, it was simple from the lecture notes and watching some videos online to get a good grasp of what was needed the implementation went smoothly. The results did not shock me, other then Rec N=80. Recursive took as long as I thought but DP took a bit more data than I thought. I started with set values to ensure the knapsack programs were correct then moved to a randomly generated version. I collected the data from taking three tests then averaging the values all together. W changes a bit at a time, but it will eventually plateau if N remains the same.

Problem 2: Shopping Spree

Acme Super Store is having a contest to give away shopping sprees to lucky families. If a family wins a shopping spree each person in the family can take any items in the store that he or she can carry out, however each person can only take one of each type of item. For example, one family member can take one television, one watch and one toaster, while another family member can take one television, one camera and one pair of shoes. Each item has a price (in dollars) and a weight (in pounds) and each person in the family has a limit in the total weight they can carry. Two people cannot work together to carry an item. Your job is to help the families select items for each person to carry to maximize the total price of all items the family takes. Write a dynamic programming algorithm to determine the maximum total price of items for each family and the items that each family member should select.

a) **Give a verbal description and give pseudo-code for your algorithm. Try to create an algorithm that is efficient in both time and storage requirements.**
For this program, the first thing we would do it to read from the "shopping.txt" file. Then ensure that we take the values and store them correctly for use in the algorithm.
Once we have correctly stored the data, we need to perform similar to a knapsack problem we did before, we just need to use the same "values" and "weights" but do it with each individual member of the family and store which items they grab and their total haul. Then print the correct data to the console.

b) **Implement your algorithm by writing a program named "shopping" (in C, C++or Python) that compiles and runs on the OSU engineering servers.**
I was unable to discover the bug for my code to complete the testing file. The code works for the first 2 test cases, but it is weirdly does not work with the other test cases. I was working with VSC 2019 and I was getting different results on the server, this really perplexed me, and I was wondering on what I can do in the future to prevent this from happening. I understand not getting full credit on this portion, but I believe I did most of the work, I hope that you see the same!

Please email me if you have any questions for me, bellj3@oregonstate.edu, I work around 40 hours a week and go to school full time so I apologize if I don't answer

immediately, but I am doing my hardest in this class and want to exceed please contact me if there's stuff I can alter to better my grade