MSSC 6000 - Homework 3

Henri Medeiros Dos Reis

March 7, 2023

Solution 1. (a) The search space for this problem is 3^n. Since every time we are faced with an item, we have 3 possible choices, either take it, or put it on knapsack 1, or put in knapsack 2.

(b) Only code.

(c) Using the answer to part (a), we have that the relationship between n and time is some constant of time per floating point operation. So we can use the data collected and the known relationship to estimate the time. The data is the following:

n	time	in sec	ratio	estimative ratio	estimative time in sec
10	0m0.378s	0.378	0.0378	0.1111111111	1.111111111
11	0m1.297s	1.297	0.1179090909	0.3333333333	3.666666667
12	0m5.224s	5.224	0.4353333333	1	12
13	0m22.742s	22.742	1.749384615	3	39
14	1m30.372s	90.372	6.455142857	9	126
15	6m5.198s	365.198	24.34653333	27	405
16	24m28.724s	1468.724	91.79525	81	1296
100				9.69774E+41	9.69774E+43

So the prediction is $9.7 * 10^{41}$.

Solution 2. (a) Only code.

(b) It definitely did. Having the solution was very helpful to find the solution without having to find the solution by hand.

(c) In case we have a large capacity, then the code will take very long before pruning new branches. Also, when testing the ordering of the items definitely matters as well. Having heavy items as the first items, speeds up the process.

(d) I was able to run around 35 items in under a minute, 9 out of 10 cases ran in under a minute.

(e) it is difficult to accurately predict how long it would take to run $n = 100$. However, we can try to make some assumptions and use the following collected data to make an estimate:

n	time
10	0.054
11	0.05
12	0.056
30	3.024
30	27.659
30	59.6
35	3.452
35	3.046
35	21.689
35	12.982
35	78.225
40	20.191
40	71.318
40	50.263
41	37.163
42	25.523
43	28.274
44	59.899
44	58.05
44	113.598
44	360.781
45	458.262

One method for estimating the relationship between two variables is to do linear regression. However, based on the given data, it is not clear what the relationship between n and time is. Given these limitations, a simple approach is to calculate the average time for each value of n and use these values to estimate the time for $n = 100$. Then the prediction is 460.000 seconds. Which is way smaller than the prediction for the last problem. However, it is possible that it would take just as long as the one in 1c, or quicker, it does depend on what the items are.

Solution 3. (a) Since this is a divide and conquer algorithm, the first thing we need is a base case. Which in this case is when the list passed as a parameter for the function has only one item, in that case, the solution is that item.

If there is more than one element, the algorithm finds the midpoint of the list. It then splits the list into two halves, left half and right half.

The algorithm then gets the last element in the left half and the first element in the right half. It compares these two elements to determine which half the maximum value is in, which will determine where the recursion should happen. If the last element in the left half is greater than or equal to the first element in the right half, the maximum value is in the left half. In this case, the algorithm recursively calls `find_max` with the left half as the new input. Otherwise, the maximum value is in the right half, and the algorithm recursively calls `find_max` with the right half as the new input.

This process is repeated until we reach the base case, which will return that value as the

maximum value.

(b) Only code.

(c) I created a function that would generated random n random numbers. Then I created two lists from those numbers, which one of them I would sort in increasing order and the other in decreasing order, after that, I combined both lists, the increasing at the front. From there, I would run my algorithm and test against the python function `max()`.

(d) To test it for the time complexity, we can write a simple function that finds the largest value by looping over all the elements. Then use the unix command `time` to run both algorithms in a set of lists with different n and compare the times from both algorithms.

Solution 4. The function that takes in a list of stock prices and returns the best day to buy and sell in order to maximize profit. The input list contains the stock prices, with the indexes of the list corresponding to the days on which the prices are recorded.

It works by dividing the list of prices in half recursively, until it reaches a base case of a single price, which has a simple solution. Then, it starts combining the solutions from the left and right halves of the list to finally find the best solution for the entire list.

In order to combine the solutions, we first find the smallest price in the left half and the largest price in the right half, then calculate the profit that we can make by buying at the smallest price in the left half and selling at the largest price in the right half. In case this profit is greater than the profit that could be made by buying and selling within the left half or within the right half, then the algorithm returns the indices of the day to buy and sell that correspond to the minimum and maximum prices.

If the profit from buying and selling across the left and right halves is not greater than the profit from buying and selling within the left or right halves separately, then the algorithm checks which half has the greater profit and returns the indices of the day to buy and sell within that half.

This logic is recursively called until it reaches the base case of a single price, at which point it returns the indices of that day.