Algorithm Homework-3 Report

In this Project we designed and implemented 0-1 Knapsack Problem and 0-1 Multiple Knapsack Problem.

Part (a): *0-1 Knapsack Problem*

In this part of Project, firstly we implemented just Greedy Algorithm for 0-1 knapsack problem for analyze how close to optimal solution. Greedy algorithm is a bit easy algorithm. It depends on all elements ratio rate ranking. For this algorithm, Greedy tend to choose bigger ratio rate. Greedy takes values respectively from big values to small values and it throws the items to knapsack. If the capacity of the bag runs out, it means algorithm is completed.

However, this algorithm may overlook more optimal value. Therefore, we could not use this algorithm without any evoluation. So that, we evaluated the algorithm for close to optimal solutions. Even, for some input files we found optimal values.

****HOW WE DESINGED THIS ALGORİTHM?*

If we start from the beginning, firstly we read the input files and we took the weights and values in 2-dimesional array. First 2 index are number of items and capacity of the bag. By approach of the Greedy Algorithm we found ratio rates for every item. And we threw this ratio rates in a array. Then, we

arranged the array and we alinged respetively from big values to small ones.

By Greedy Algorithm approach we use items that have maximum ratio rate respectively. Until capacity of bag runs out we threw item to bag. Actually until this time we just implemented greedy algorithm. After this time, we started to improving our algorithm.

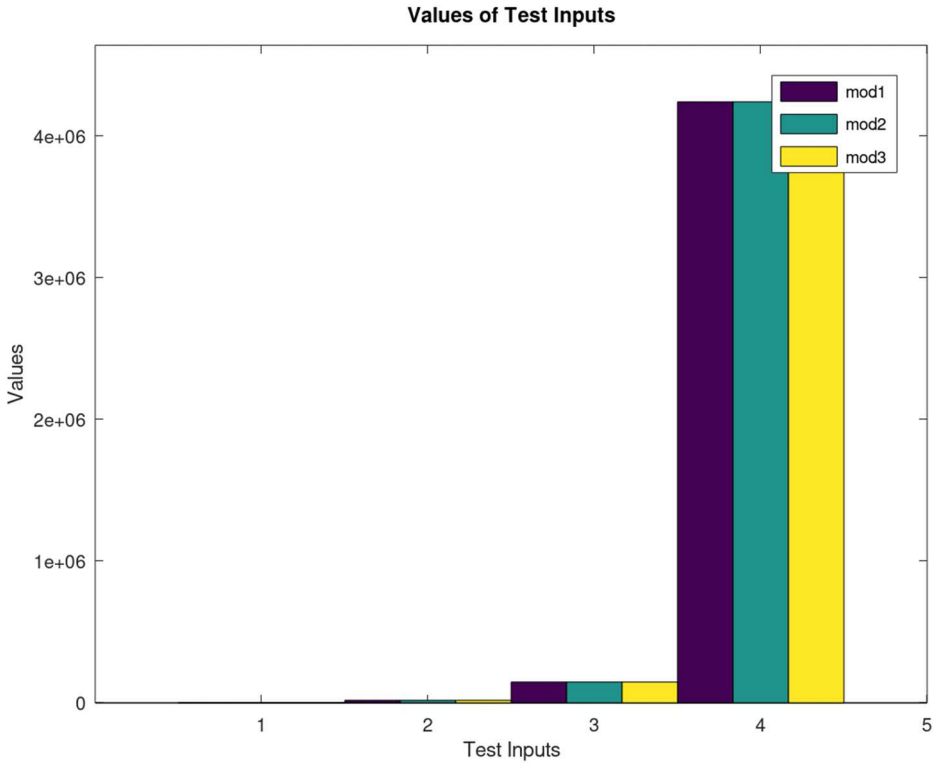
Firstly, our approach is depend on if we do not use any item, could we close to optimal value. Therefore, we used algorithm without using any items and we did it according to our rule. We defined 3 mod that are mod1, mod2 and mod3. In mod1, we started to remove from which has bigger ratio rate so index 0 because our array is respectively from bigger ratio rate to small ones.In next one mode2, we started to remove from index of half of array size. In last one mod3, we just used item that are starting from (total items)-(log size) with multiply some numbers (etc. 1, 2 and 3).

Our approach showed us if we start with removing which has bigger ratiorate(so mod1), elapsed time was increasing but we was closing the optimal solutions. We have to make a choice, more fast or more accurate. In this Project we chose more accurate but we did all possibilities with using mod's.

In below, there is a table which showing values with mod's.

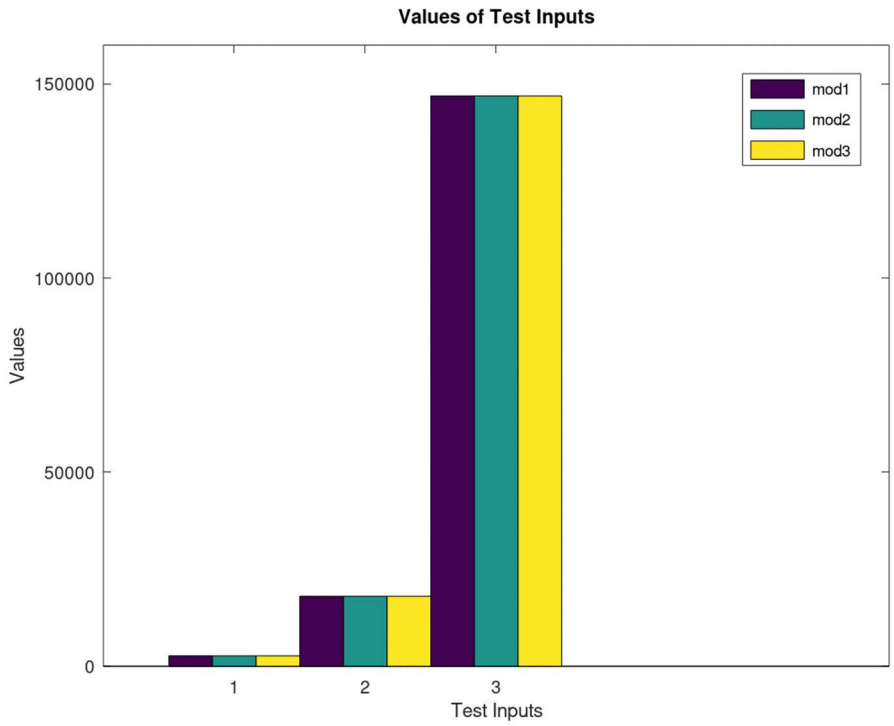
	test1a	test1b	test1c	test1d
mod1	2697	18046	146919	4241988
mod2	2677	18046	146919	4241988
mod3	2671	18045	146892	4241988

As we can see, if we choose start to removing 0th index we close to optimal values. However we spend more time for the sake of closing optimal solution. In below, we created a table which is showed elapsed times by mod's.



This graph shows us values of test inputs with mod's. Dark blue ones is more accurate. However, as we can, sampla input test1d have too much items so that it creating hassle to analyze table but we saw that comparison of sizes. To

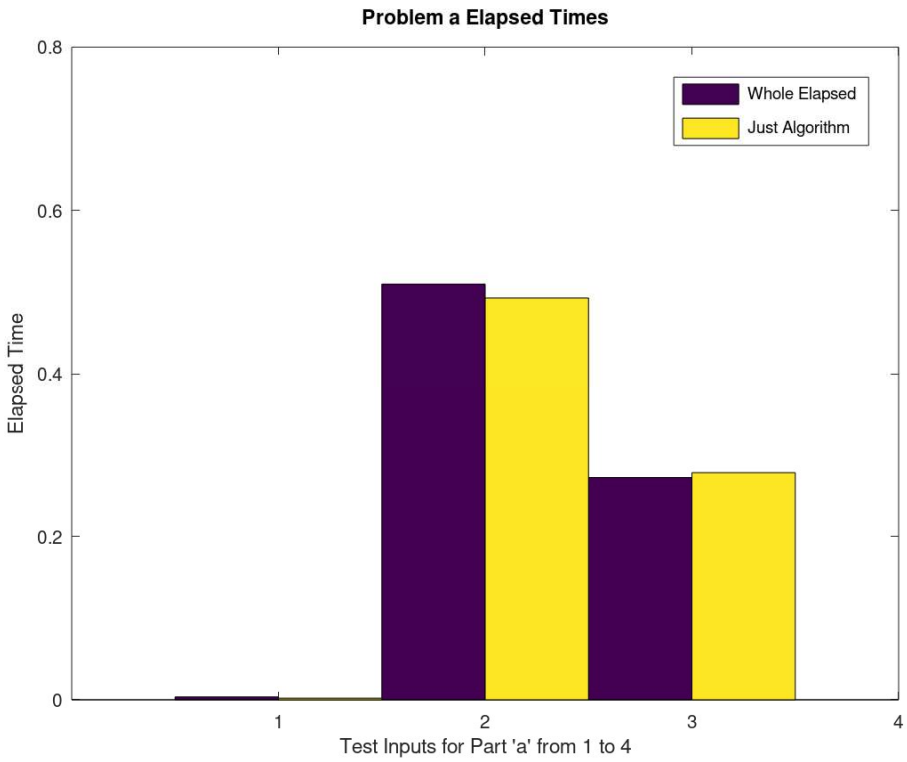
prevent it we created graph with out sample input test1d in below.



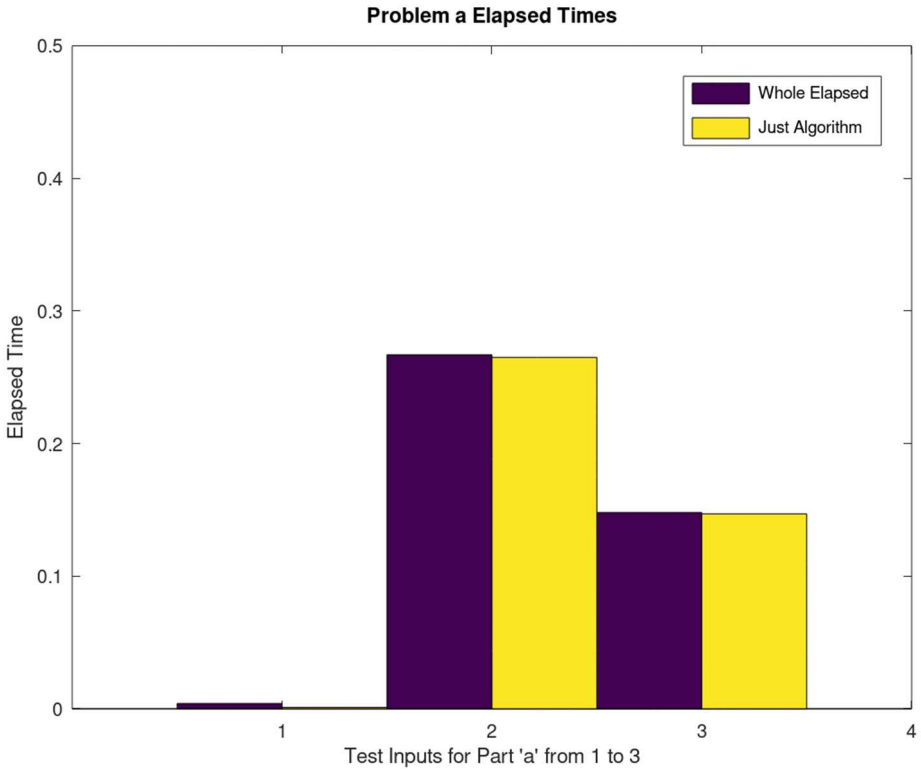
We mentioned that if we start from removing which has bigger ratio rate item(if we use mod1), our elapsed time is increasing but we are closing to optimal value. In below, the table shows us elapsed time with mod's.

	test1a	test1b	test1c	test1d
mod1	0.0035	0.510	96.85	0.272
mod2	0.0039	0.265	46.95	0.148
mod3	0.0030	0.147	4.10	0.135

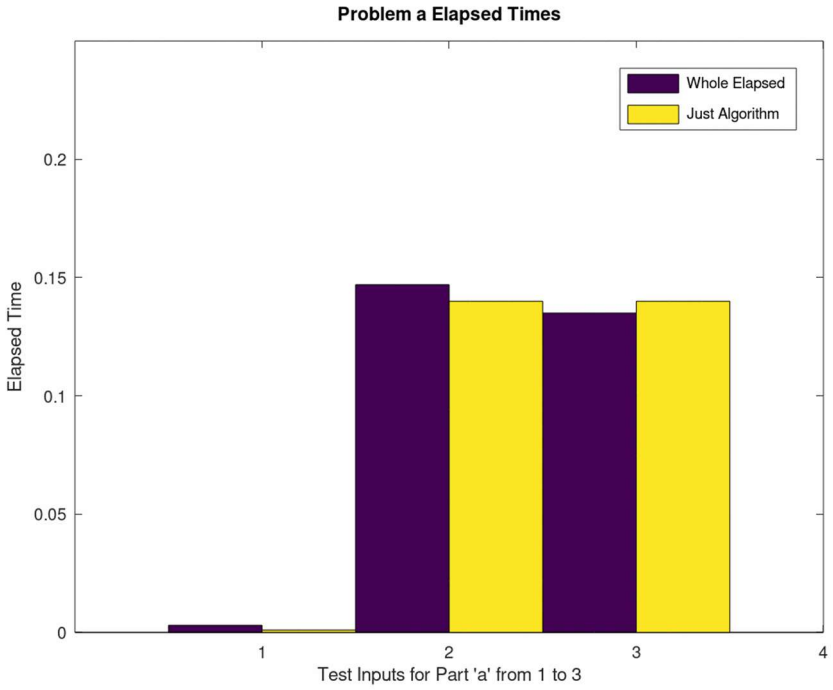
(seconds)



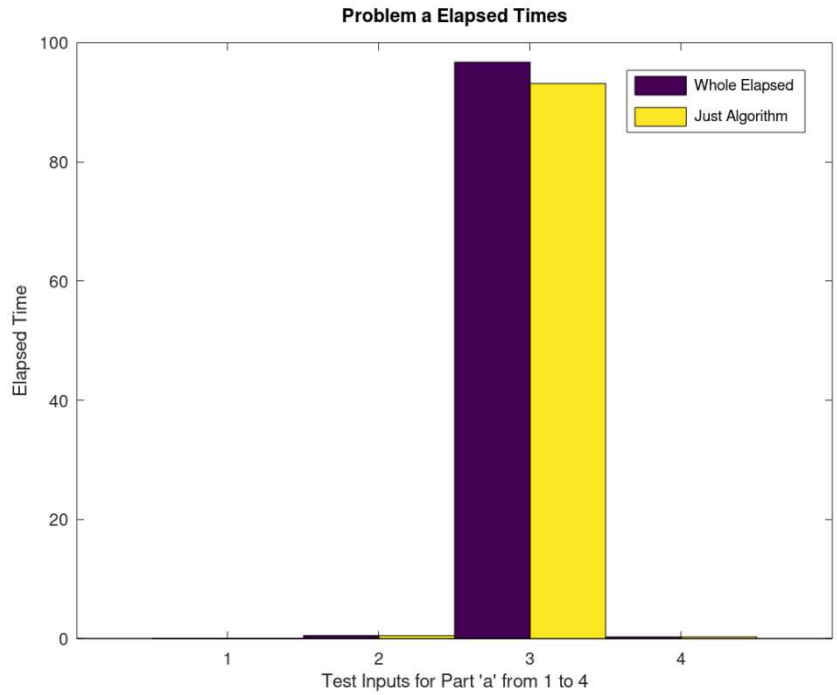
Test inputs(a-b-d) with mod1



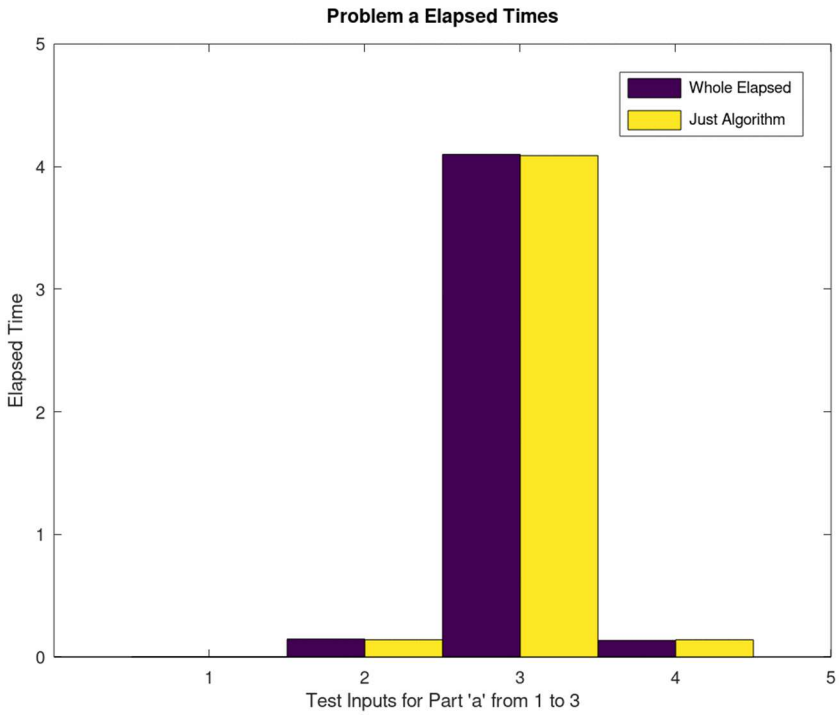
Test inputs(a-b-d) with mod2



Test inputs(a-b-d) with mod3



Test inputs(a-b-c-d) with mod1



Test inputs(a-b-d) with mod3

As we can notice that sample input test1c is so big and it elapsed time is long. Therefore we can not analyze correctly in whole sample inputs in one graph so we created 2 different graph with test1c and without test1c.

- We used Array as Data Structures

- **Time Comlexivity :** Time comp. of our algorithm is $O(n^2)$. Because in part of our code, we used insertion sort for array operation.

  $O(n^2)$

- **Space Efficieny :** 5 one dimensional and 1 two dimensional array

While we were designing and implementing our algorithm we encountered some problems and we solved it. For exapmle, while we were choosing items that have same ratio rate we faced some problems. When we were choosing 'x' ratio item, the algorithm were choosing same item every time but we wanted that if a item has used, dont use it again. For solving this problem, if the item was used we put -1 for do not use it again.

copyRatio[k]=-1;

In addition, we encountered classic C problems. For example, we initiliaze all array element to 0, but some times, we saw that some element is so big and did not equal to zero.

.....

After all, if we only used Greedy algorithm, we get this values. The table in below shows us values and elapsed time if we use only Greedy Algorithm without our reorganization of algorithm.

	test1a	**test1b**	**test1c**	**test1d**
Values	2649	18038	146888	4241499

	test1a	**test1b**	**test1c**	**test1d**
Time Elap.	0.003988	0.0173	0.1936	0.0209

Part (b): *0-1 Multiple Knapsack Problem*

In this part of Project we also used Greedy Algorithm for solving Multiple Knapsack Problem.

It selects among n items to load m knapsacks in order to maximize the resulting total profit. Obviously, for each knapsack, the total weight of the selected items should not exceed its capacity. In addition, this problem can not solved by polynomial time algorithm.

In this algorith,we put the items to knapsacks respectively. Then we put the items to knapsacks reverse respectively. For example, there is 3 knapsack which are 80, 60 and 70. Firstly,we put items which has bigger ratio in 80 knapsack. If 80 knapsack will full out, we continue from 60 and then 70. In reverse respectively it is the opposite.

- We used Array as Data Structures

- **Time Comlexivity :** There may be 2 time complexivity. These are **O(size^2) for size>Bag Number** and

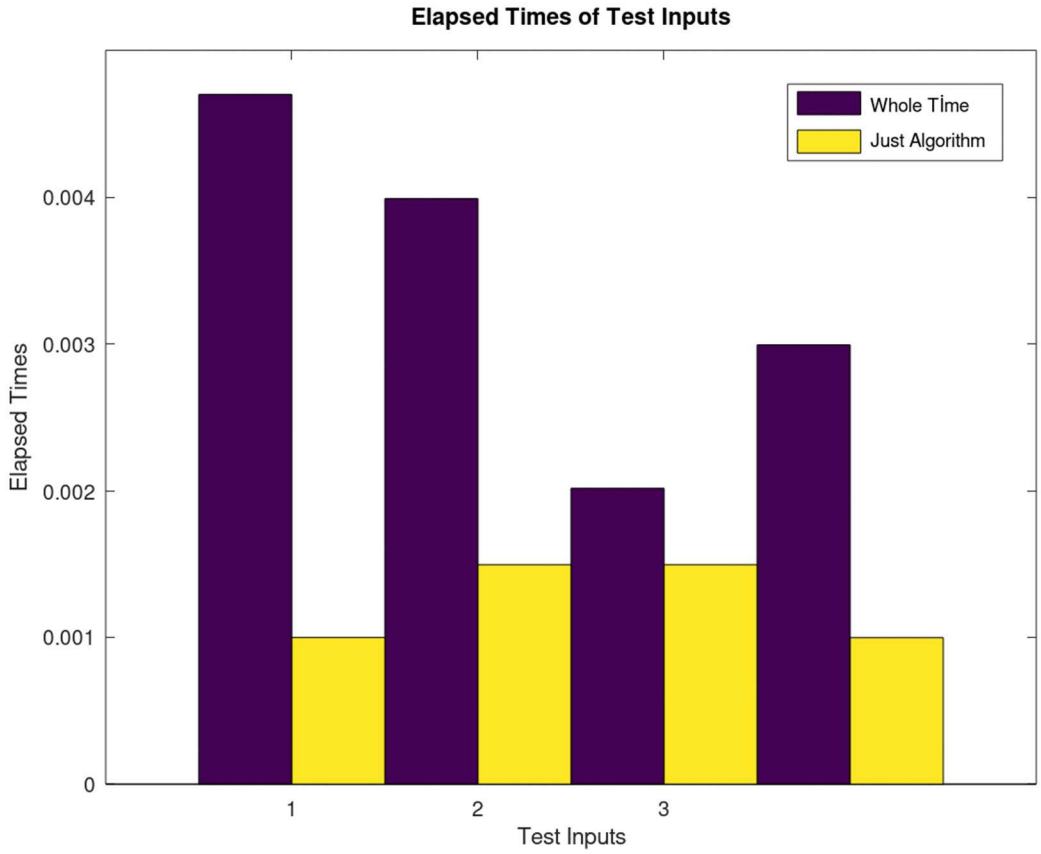
  O(2*size*BagNumber) for size<Bag Number.

- **Space Efficieny :** 4 one dimensional and 3 two dimensional array

In below, the table shows us the values of the Multiple Knapsack Problem.

	test2a	test2b	test2c	test2d	test2e
Values	335	423	369	326	146686

In below, the graph shows us 2 elapsed times. These are elapse the whole duration and just after reading inputs and just before writing outputs.



	test2a	test2b	test2c	test2d	test2e
Whole Time	0.0045	0.00399	0.00202	0.00299	0.226
Just Algo.	0.00099	0.0015	0.0015	0.00099	0.202

In above, there are elapsed times of test inputs of b part of the Project. As we mentioned above, there is 2 elapsed time

MALİK TÜRKOĞLU