#Exercise 5 Dynamic Programming

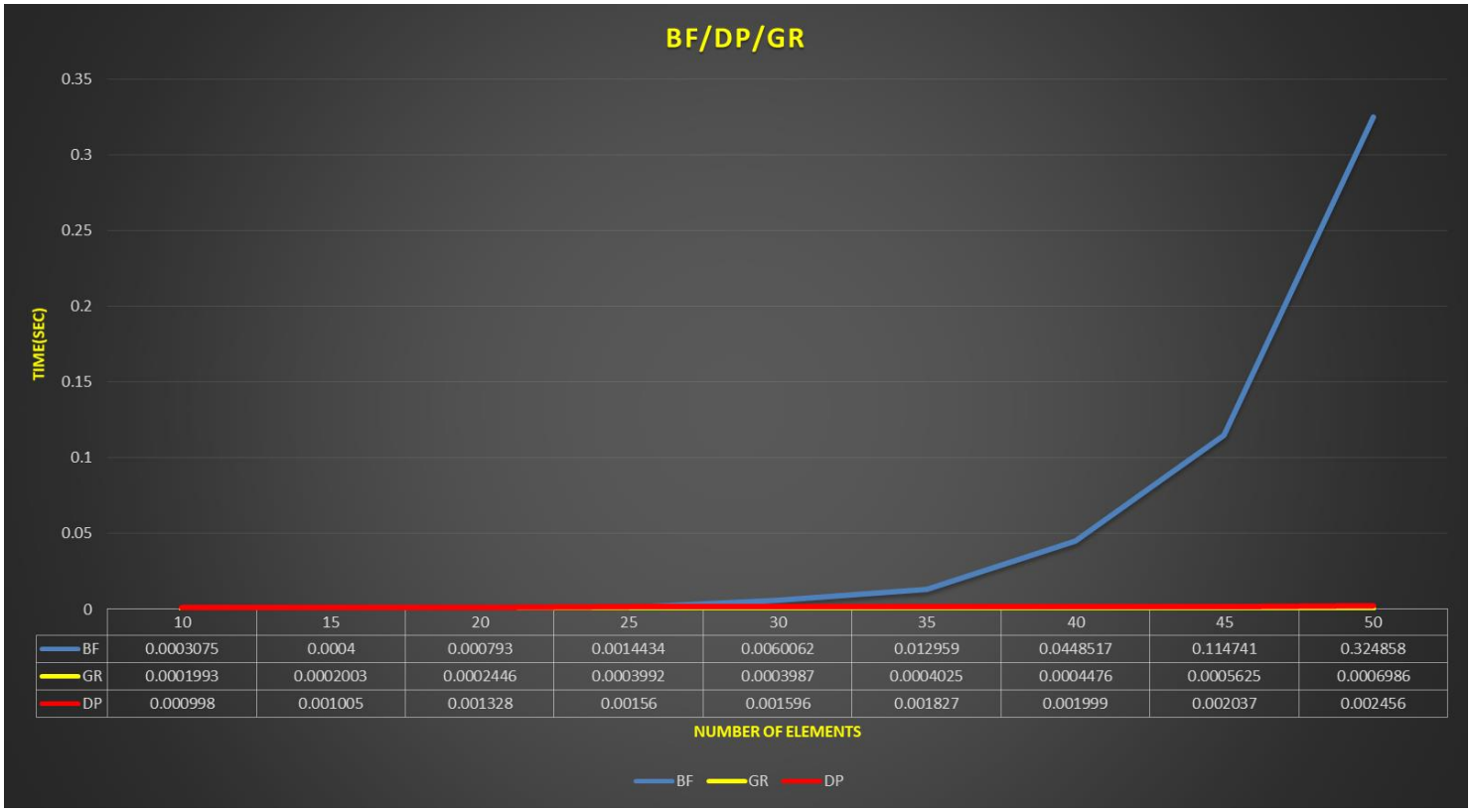
Uladzimir Ivashka, 150281

Sofya Aksenyuk, 150284

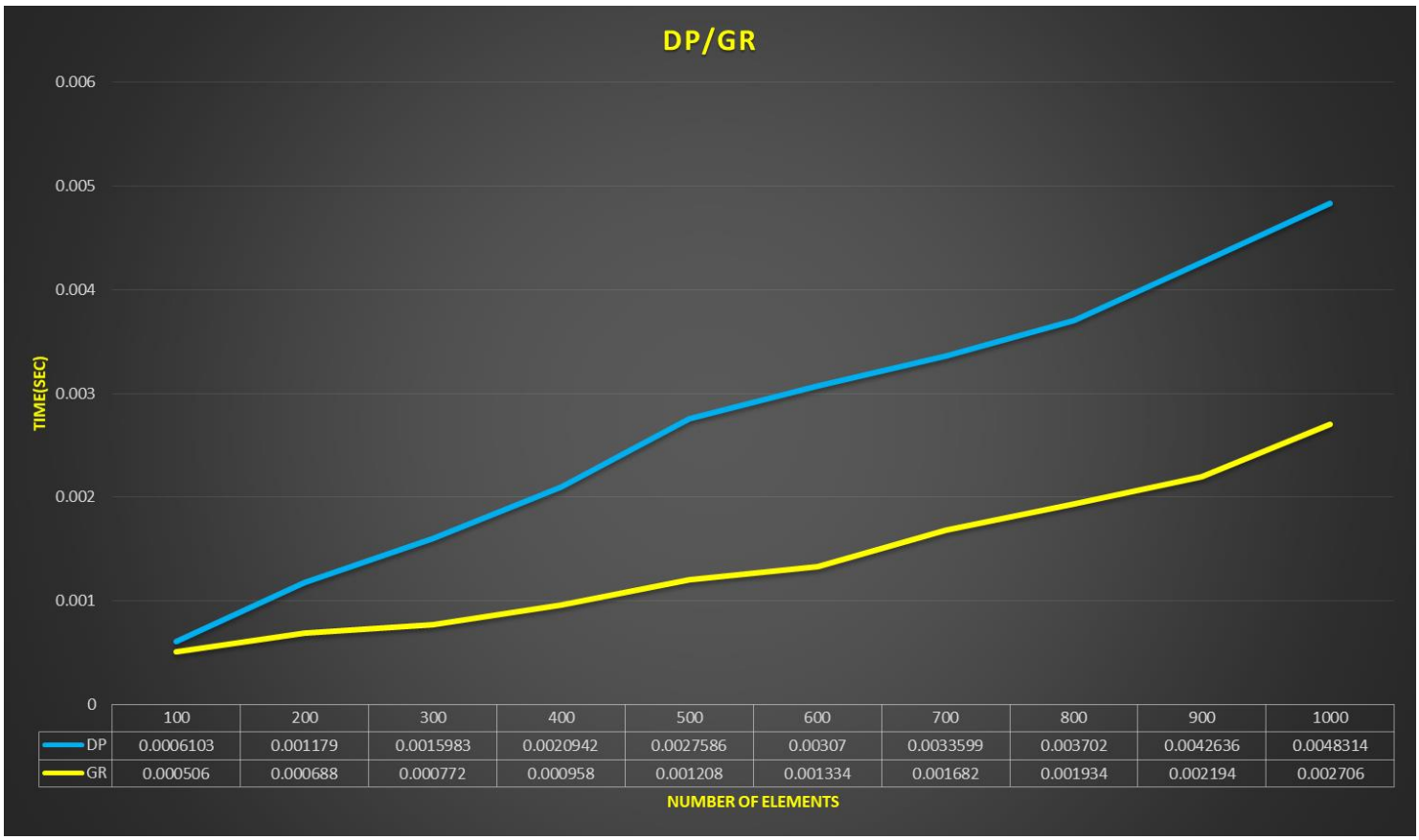
Remark:

Given results of all the below experiments are average of 10 runs.

Comparison the time to obtain a solution for Knapsack problem employing three algorithms, namely, Dynamic Programming, Brute Force (Exhaustive Search) and Greedy algorithms:

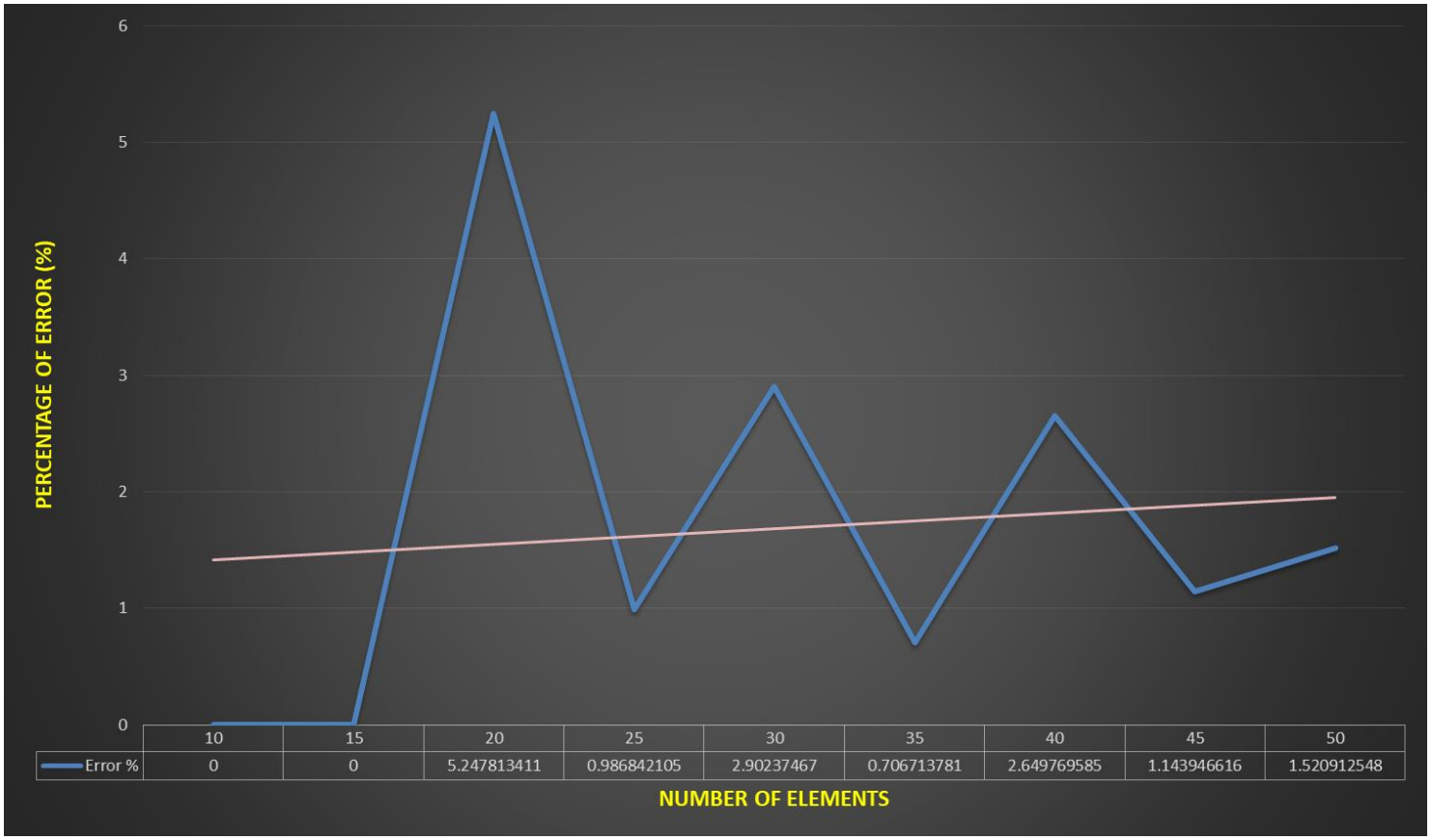


	10	15	20	25	30	35	40	45	50
BF	0.0003075	0.0004	0.000793	0.0014434	0.0060062	0.012959	0.0448517	0.114741	0.324858
GR	0.0001993	0.0002003	0.0002446	0.0003992	0.0003987	0.0004025	0.0004476	0.0005625	0.0006986
DP	0.000998	0.001005	0.001328	0.00156	0.001596	0.001827	0.001999	0.002037	0.002456



	100	200	300	400	500	600	700	800	900	1000
DP	0.0006103	0.001179	0.0015983	0.0020942	0.0027586	0.00307	0.0033599	0.003702	0.0042636	0.0048314
GR	0.000506	0.000688	0.000772	0.000958	0.001208	0.001334	0.001682	0.001934	0.002194	0.002706

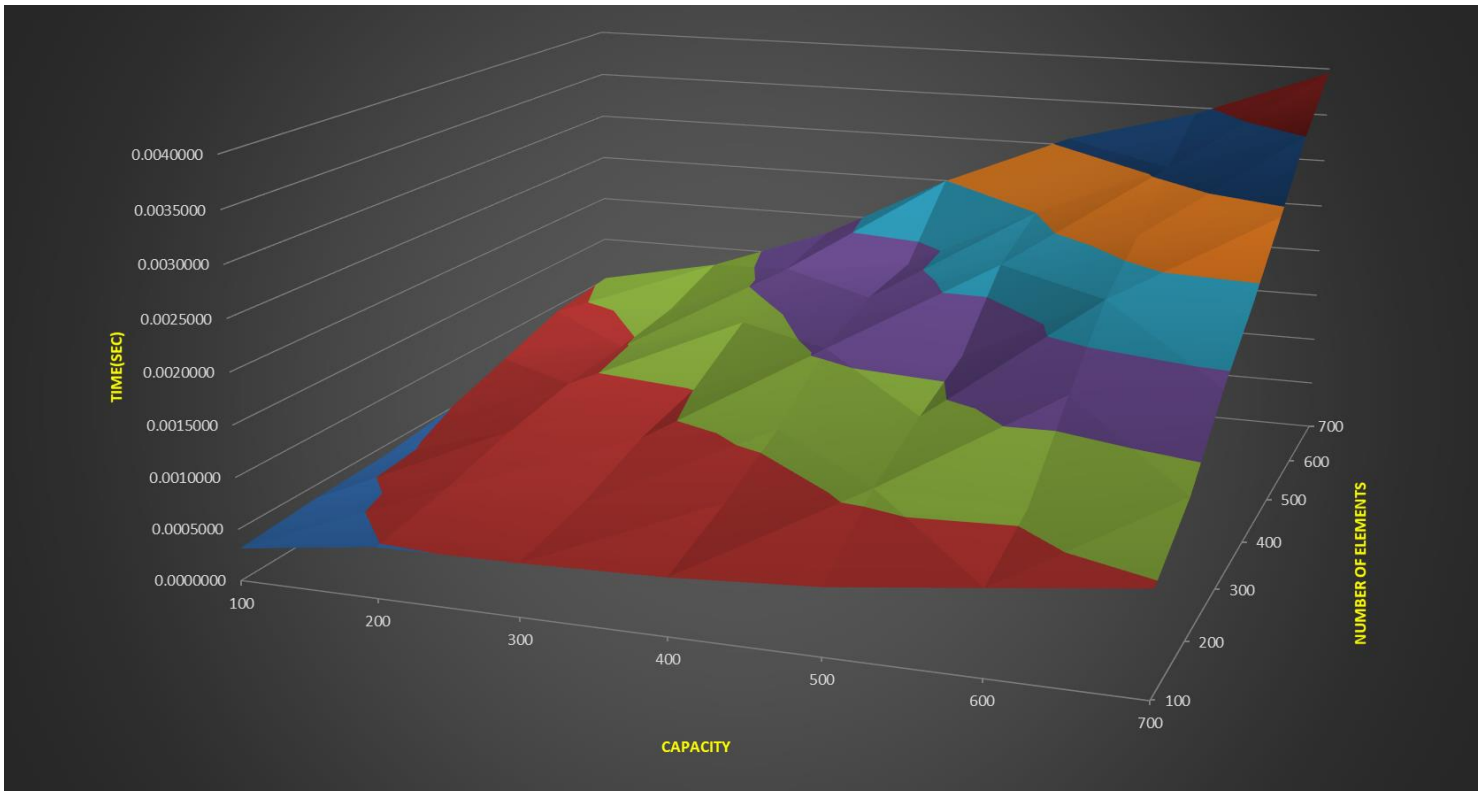
Percentage of result errors obtained running Greedy algorithm in comparison with DP and BF:



Comparison of obtained by BF, DP and Greedy algorithms results for the same input instances:

	BF	GR	DP
10	257	257	257
15	498	498	498
20	343	325	343
25	608	602	608
30	379	368	379
35	566	562	566
40	868	845	868
45	1049	1037	1049
50	789	777	789

Dependence of knapsack capacity for DP (since its time complexity is O(N*W)):



Conclusions:

According to the charts above, it can be concluded that Brute Force is the least efficient algorithm for Knapsack problem among all the presented ones. The provided computations support the theoretical background – its time complexity is equal to O(2^n) which is exponential function. Such a complexity can be explained by the algorithm straightforward work which simply considers all subsets of input data. Therefore, it finds the best solution for provided capacity of knapsack which proves that BF gives optimal solutions.

As for Dynamic Programming Algorithm, it works slower than Greedy but much faster than BF. Nevertheless, due to time complexity of O(N*W) (where N – number of input elements and W – knapsack capacity) computations of its complexity depend on assumed function values. The algorithm creates a NxW table and recursively fills it in a bottom-up approach that firstly calculates each of the elements of the first row and then works up to the final row which leads to our recurrence never trying to access elements of the table that haven't already been calculated. So that DP determines an optimal solution by first finding optimal solutions to created subproblems. Remark: DP does not cause memory leaks since previous rows are not being kept.

The smallest time results are obtained implementing Greedy algorithm which is predictable due to its polynomial time complexity of O(n log2 n). The essence of the algorithm makes its work that fast: it creates a list of unit values (element's value / element's weight) and sorts it in descending order till it exceed an input knapsack capacity. Thereby, it gives us only approximate results because of the selection of elements with the highest value to weight ratio.

Overall:

The hardness of the knapsack problem depends on the form of the input. If the weights and profits are given as integers, it is weakly NP-complete, while it is strongly NP-complete if the weights and profits are given as rational numbers.