

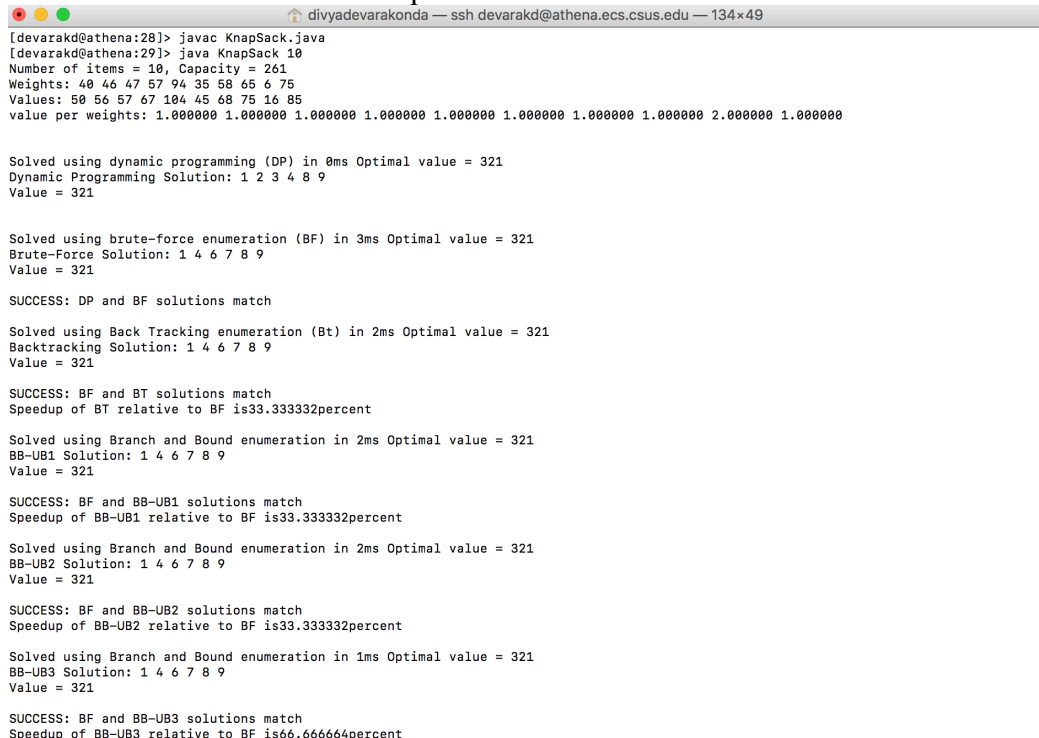
206 Assignment 4

Name: Divya Sindhuri Devarakonda
Sac Id: 219220782

1. Running times of Algorithms given in milli seconds (ms)

	Brute Force	Backtracking	B&B UB1	B&B UB2	B&B UB3	Dynamic Programming
N = 10	2	0	0	0	0	0
N = 20	80	52	20	15	7	0
N = 30	73719	48227	6978	3233	53	2
N = 40	Timeout	Timeout	Timeout	Timeout	98	3
Largest Input Solved in 10s	26	28	33	44	Appx 3300	Appx 4000

2. Screenshot to shot Match for input size 10



```
[devarakd@athena:28]> javac KnapSack.java
[devarakd@athena:29]> java KnapSack 10
Number of items = 10, Capacity = 261
Weights: 40 46 47 57 94 35 58 65 6 75
Values: 50 56 57 67 104 45 68 75 16 85
value per weights: 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 2.000000 1.000000

Solved using dynamic programming (DP) in 0ms Optimal value = 321
Dynamic Programming Solution: 1 2 3 4 8 9
Value = 321

Solved using brute-force enumeration (BF) in 3ms Optimal value = 321
Brute-Force Solution: 1 4 6 7 8 9
Value = 321

SUCCESS: DP and BF solutions match

Solved using Back Tracking enumeration (Bt) in 2ms Optimal value = 321
Backtracking Solution: 1 4 6 7 8 9
Value = 321

SUCCESS: BF and BT solutions match
Speedup of BT relative to BF is33.333332percent

Solved using Branch and Bound enumeration in 2ms Optimal value = 321
BB-UB1 Solution: 1 4 6 7 8 9
Value = 321

SUCCESS: BF and BB-UB1 solutions match
Speedup of BB-UB1 relative to BF is33.333332percent

Solved using Branch and Bound enumeration in 2ms Optimal value = 321
BB-UB2 Solution: 1 4 6 7 8 9
Value = 321

SUCCESS: BF and BB-UB2 solutions match
Speedup of BB-UB2 relative to BF is33.333332percent

Solved using Branch and Bound enumeration in 1ms Optimal value = 321
BB-UB3 Solution: 1 4 6 7 8 9
Value = 321

SUCCESS: BF and BB-UB3 solutions match
Speedup of BB-UB3 relative to BF is66.666664percent
```

3. Results Discussion

Brute Force :This algorithm tries all the possible combinations which will be equal to $2^{\text{pow } N}$ different possible solutions .Its exponential algorithm which checks the all the possible conditions. This algorithm for $N=40$ is running more than 10hrs.

Backtracking Algorithm: This algorithm is slight modification of the Brute force algorithm where we prune the trees when the capacity is full. Even this an exponential algorithm but as we are pruning some nodes this is better than Brute force algorithm but still it takes the same order of time as brute force.

The speed up when compared to brute force is for $n=30$ is just 21.429377percent

Branch and Bound 1: Calculated the upper bound by subtracting the untaken item values from the sum of all the values. This is loose upper bound, but it's still effective for 0/1 Knapsack when compared to backtracking. For $N=30$, it gives 99.92% speed up compared to Brute Force. Interestingly even though this bound is loosest bound among remaining two there are cases when this is better than Ub2 as Ub2 involves calculating of the weights each node which is order of $\Theta(n)$ at each node. In this algorithm calculating the remaining capacity is just $\Theta(1)$. The speed up when compared to brute force is for $n=30$ is 98.93425percent

Branch and Bound 2: calculated the upper bound by considering only the weights which fit into the knapsack at each node. This calculation cannot be done in the cost of $\Theta(1)$. At each node we should iterate through remaining weights see if that fits. This will be a tighter bound compared to Ub1 so we can prune more nodes. But the cost of finding this bound is drawback for this method. Consider the situation where all the weights are just differed by small value. Then even though ub1 is looser than this we can best time with Ub1 when compared to Ub2. The speed up when compared to brute force is for $n=30$ is 99.12797percent

Branch and Bound 3: The tightest upper bound among all the tree. The upper bound is calculated by considering the remaining weights and remaining capacity as fractional knapsack problem and we compute the fractional knapsack bound for this. The tighter the bound the more nodes we can prune. In ten seconds this ub3 can solve more than 300 inputs while other two loose bounds solve only around 40. The speed up when compared to brute force is for $n=30$ is 99.9968percent

Dynamic Programming:

The best among the all the algorithms which we are comparing in this report for knapsack. The input of 50 is solved in 1ms. In this technique we will try to solve the sub problems and store the values so instead of solving again we just get the already computed values. This is based on Memorization technique. All the inputs less than 45 are solved in 0ms. Which is the fastest technique. The speed of this algorithm is 100 percent when compared to the brute force algorithm.

Machine Configurations

Operating System: MAC OS

Processor: Intel Core i5

Memory: 8 GB

Cores: 2

Processors: 1