Compare the Backtracking and Branch & Bound Approach by the implementation of 0/1 Knapsack problem. Also compare the performance with dynamic

programming approach.

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 20

// Function to find a subset that sums to the target using backtracking

void sumOfSubset(int arr[], int n, int target, int subset[], int subsetSize, int index) {

// If the target sum is reached, print the current subset

if (target == 0) {

printf("Subset found: ");

for (int i = 0; i < subsetSize; i++) {

printf("%d ", subset[i]);

}

printf("\n");

return;

}

// If there are no more items left or the target becomes negative, return

if (index == n || target < 0) {

return;

}

// Include the current element in the subset and recurse

subset[subsetSize] = arr[index];

sumOfSubset(arr, n, target - arr[index], subset, subsetSize + 1, index + 1);

// Exclude the current element from the subset and recurse

sumOfSubset(arr, n, target, subset, subsetSize, index + 1);

}

// Function to initiate the backtracking process

void findSubsetSum(int arr[], int n, int target) {

int subset[MAX\_SIZE];

sumOfSubset(arr, n, target, subset, 0, 0);

}

int main() {

// Example input: Set of integers and the target sum

int arr[] = {3, 34, 4, 12, 5, 2};

int n = sizeof(arr) / sizeof(arr[0]);

int target = 9;

printf("Finding subsets of array that sum to %d...\n", target);

findSubsetSum(arr, n, target);

return 0;

}