Assignment 7

Algorithms, Spring 2023

Honor code: *Work on this assignment alone or with one partner. Between different teams, collaboration is at level 1 [verbal collaboration only]. There are lots of resources online, such as animations, visualizations, practice problems, videos, and solutions— which you are encouraged to explore to deepen your understanding. However, you must be careful not to search for the specific problems in the assignment with the intent of getting hints for the solution. Searching for the assignment problems on the internet violates academic honesty for this class.*

Load balancing:[1] You have been hired to design algorithms for optimizing system performance. Your input is an array $J[1..n]$ where $J[i]$ or J_i represents the running time of job i; jobs do not have specific start and end times, but they can be started at any time (this is a different scenario than in the interval scheduling /activity selection problem). The running times are integers.

In this problem you wil design and implement an algorithm for determining whether there is a subset S in J such that the running time of the elements in S sum up precisely to the same amount as the sum of the elements not in S; more formally, $\sum_{J_i \in S} J_i = \sum_{J_i \in J-S} J_i$. The algorithm should run in time $O(n \cdot K)$, where K is the sum of the running times of the n jobs.
Examples:

```
vals = []
True, subset=[]

vals = [1]
False

vals = [1,2,1]
True, subset=[1,1]

vals=[1, 2, 3, 4]
False

vals = [1, 5, 11, 5]
True, subset=[1, 5, 5]
```

[1] Leetcode #416: Given a non-empty array containing only positive integers, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

To solve this problem, you will answer all the questions below in a Python notebook. Use as examples the notebooks for the rod cutting and the robber problems, which you can download from the class website.

1. **Recursive:** Your first task is to come up with a general recursive solution for this problem (without dynamic programming).

   Your recursive function shoud be called *subsetSum*, and should return a boolean.

   ```
   # describe its parameter and return value
   def subsetSum(vals, ....)
   ```

   Provide a wrapper function, called *equalSubset*, which will call your recursive function *subsetSum* with the correct parameters in order to answer the question on array *vals*.

   ```
   # PARAMETER vals: an array of values, assume all values are positive
   # integers RETURN: true if the array can be partitioned into two
   # subsets such that the sum of the elements in both subsets is equal
   # False otherwise
   def equalSubset(vals):
       #...
       #call subsetSum(vals, ...)
   ```

   Once both functions are implemented, you should be able to test various arrays like so:

   ```
   vals1=[1,5,11, 5]
   vals2=[1, 2, 3, 5]
   vals3=[1, 1, 1, 1]
   equalSubset(vals1)
   True
   equalSubset(vals2)
   False
   equalSubset(vals3)
   True
   ```

   We will run and test your code with the following function:

   ```
   # PARAMETER myFun is a function that takes an array as arg and returns True of False
   def testIt(myFun):
      vals = [
           [[1,5,11, 5], True],
           [[1, 2, 3, 5], False],
           [[1, 1, 1, 1], True],
           [[1, 1, 2], True],
           [[], True],
           [[1], False],
   ```

```
        [[1,2], False],
        [[1,2,3,4,5,6,7,8,9,10, 45], True],
        [[1,2,3,4,5,6,7,8,9,10, 40], False],
    ]

    failed = 0
    for v in vals:
        print("testing ", myFun, " on ", v[0])
        res = myFun(v[0])
        if res == v[1]:
            print("Returns ", res, ", should be ", v[1], ". Passed.")
        else:
            failed = failed +1
            print("Returns ", res, ", should be ", v[1], ". Failed.")

    ntests = len(vals)
    passed = ntests-failed
    print("Testing done. ", passed, " passed, ", failed, " failed." )
```

When I run *testIt(equalSubset)* on my implementation of *equalSubset*, I get the following:

```
testing  <function equalSubset at 0x11809a820>  on  [1, 5, 11, 5]
Returns  True , should be  True . Passed.
testing  <function equalSubset at 0x11809a820>  on  [1, 2, 3, 5]
Returns  False , should be  False . Passed.
testing  <function equalSubset at 0x11809a820>  on  [1, 1, 1, 1]
Returns  True , should be  True . Passed.
testing  <function equalSubset at 0x11809a820>  on  [1, 1, 2]
Returns  True , should be  True . Passed.
testing  <function equalSubset at 0x11809a820>  on  []
Returns  True , should be  True . Passed.
testing  <function equalSubset at 0x11809a820>  on  [1]
Returns  False , should be  False . Passed.
testing  <function equalSubset at 0x11809a820>  on  [1, 2]
Returns  False , should be  False . Passed.
testing  <function equalSubset at 0x11809a820>  on  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 45]
Returns  True , should be  True . Passed.
testing  <function equalSubset at 0x11809a820>  on  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 40]
Returns  False , should be  False . Passed.
Testing done.  9  passed,  0  failed.
```

What is the worst-case running time on an array of size n? No need of proof, just the answer is sufficient.

2. **Memoized DP:** Your second task is to memo-ize your recursive solution from part (1).

   Your recursive memo-ized function shoud be called *subsetSumDP_memoization*, and should return a boolean.

```
# describe its parameter and return value
def subsetSumDP_memoization(vals, table,  ....)
```

   Provide a wrapper function, called *equalSubsetDP_memoization*, which will call your recursive function *subsetSumDP_memoization* with the correct parameters in order to answer the question on array *vals*.

```
# PARAMETER vals: an array of values, assume all values are positive
# integers RETURN: true if the array can be partitioned into two
# subsets such that the sum of the elements in both subsets is equal
# False otherwise
def equalSubsetDP_memoization(vals):
    #...
    #create a table and initialize it
    #call subsetSum(vals, table, ...)
```

   Once both functions are implemented, you should be able to test various arrays like so:

```
vals1=[1,5,11, 5]
vals2=[1, 2, 3, 5]
vals3=[1, 1, 1, 1]
equalSubsetDP_memoization(vals1)
True

equalSubsetDP_memoization(vals2)
False

testIt(equalSubsetDP_memoization)
testing  <function equalSubsetDP_memoization at 0x1180ad430>  on  [1, 5, 11, 5]
Returns  True , should be  True . Passed.
testing  <function equalSubsetDP_memoization at 0x1180ad430>  on  [1, 2, 3, 5]
The jobs cannot balance because the total is odd
Returns  False , should be  False . Passed.
testing  <function equalSubsetDP_memoization at 0x1180ad430>  on  [1, 1, 1, 1]
Returns  True , should be  True . Passed.
testing  <function equalSubsetDP_memoization at 0x1180ad430>  on  [1, 1, 2]
Returns  True , should be  True . Passed.
testing  <function equalSubsetDP_memoization at 0x1180ad430>  on  []
Returns  True , should be  True . Passed.
testing  <function equalSubsetDP_memoization at 0x1180ad430>  on  [1]
The jobs cannot balance because the total is odd
```

```
Returns  False , should be  False . Passed.
testing  <function equalSubsetDP_memoization at 0x1180ad430>  on  [1, 2]
The jobs cannot balance because the total is odd
Returns  False , should be  False . Passed.
testing  <function equalSubsetDP_memoization at 0x1180ad430>  on  [1, 2, 3, 4, 5, 6, 7, 8,
Returns  True , should be  True . Passed.
testing  <function equalSubsetDP_memoization at 0x1180ad430>  on  [1, 2, 3, 4, 5, 6, 7, 8,
The jobs cannot balance because the total is odd
Returns  False , should be  False . Passed.
Testing done.  9  passed,  0  failed.
```

3. **Extra credit: Iterative solution** If you want to take this one step further, you can come up with an iterative solution that avoids recursion alltogether:

   ```
   # PARAMETER vals: an array of values, assume all values are positive
   # integers RETURN: true if the array can be partitioned into two
   # subsets such that the sum of the elements in both subsets is equal
   # def equalSubsetDP_iterative(vals): create the table fill it in the
   # right order
   ```

   Once done, you can test it the same way:

   ```
   testIt(equalSubsetDP_iterative)
   ```

4. **Empirical evaluation:** The code for the empirical evaluation is provided, and if you write the functions following the guidelines it will work as is:

5. **Full solution:** Finally, you will extend your solution for part (2) to find and print the subset.

   ```
   # PARAMETER vals: an array of values, assume all values are positive
   # integers
   #RETURN: true if the array can be partitioned into two
   # subsets such that the sum of the elements in both subsets is equal
   # def equalSubsetDP_memoization_withSubset(vals):
      #
      #as before, create a table
      canbalance = #call subsetSumDP_memoization(vals, table, ...)
      if canbalance:
           subset = #call a function  that returns the subset
           print("subset:", subset)
      return canbalance
   ```

   Once implemented, you could test it like so:

```
vals1=[1,5,11,5]
equalSubsetDP_memoization_withSubset(vals1)
[1, 5, 11, 5]
The total value is:  22
subset:  [5, 5, 1]
Out[35]:
True
```

Note: For this assignment you will turn in only the notebook, nothing else. Please submit on Canvas.

Evaluation

The assignment will be evaluated along several criteria:

1. **Correctness**: Is your solution correct?
2. **Justification**: Is your answer justified?
3. **Style**: Does it look professional and neat? Is the explanation written carefully in complete sentences, and well-organized logic? Is it easily human-readable? Is it easy to understand?

    - Assignments should be typed. Feel free to annotate the pdf to add figures and formulas which are too time-consuming to type.
    - Write each problem on a separate page or leave plenty of space between problems so that we can write comments.
    - Try to put yourself in the position of the reader. If you hadn't been thinking of this problem for 3 hours, would your answers make sense to you?
    - Try to finish the assignment early, then step away for a day or two, and then come back to it and read it again. Chances are you'll find something you can write more clearly.
    - Look at posted solutions for style advice (if solutions are not posted, ask).