This lab gets you to work with recursive and greedy algorithms and also practically compare their efficiency by testing them on randomly generated inputs.

Marks (max 5): Questions 1-4: 1 each | Questions 5-6: 0.5 each

Question 1

Looking back at Questions 2 and 3 of Lab 3, write a Python function:

```
def randomEventArray(s)
```

which returns an array of length `s` that in each position has an `Event` object with random `startTime` and `endTime`. Make sure that `startTime` and `endTime` are integers between 0 and 24, and that `startTime` < `endTime`. Moreover, the array should be ordered by event `startTime`. Then, write Python functions:

```
def scheduleTime(E)
def scheduleGDTime(E)
```

that schedule `E` using respectively the functions `schedule` and `scheduleGD` that we saw in the lecture of week 4 and return the time taken to do the scheduling. Test your two timing functions on at least 3 examples each and compare the times obtained.

Question 2

Let `coin = [200, 100, 50, 20, 10, 5, 2, 1]`. Write a modified version:

```
def coinSplitGD2(m)
```

of the coin split method `coinSplitGD(m)` which works as follows. It returns an array of integers of the same length as the array `coin` that it uses, where each position gives the number of coins needed of the coin at that position.

So if `m` is 143, it will return [0, 1, 0, 2, 0, 0, 1, 1] meaning no 200-coins, one 100-coin, no 50-coins, two 20-coins, no 10-coins, no 5-coins, one 2-coin and one 1-coin.

Question 3

Write a modified version of the greedy scheduling method so that instead of returning the maximum number of events, it returns an array of the `Event` objects chosen, sorted by `startTime`. So, with the example given in the lecture (week 4), it will return:

```
[Event(9,12), Event(13,15), Event(16,19), Event(20,22)]
```

Question 4

Write a Python function:

```
def closestSubset(s,A)
```

that takes an integer `s` and an array of positive integers `A` and returns an array consisting of elements of `A` which add up to `s`. If there is no subset that adds up to `s`, the function should instead return the subset which adds up to the value closest to `s`. For example:

- if `A` is [12, 79, 99, 91, 81, 47] and `s` is 150, it will return [12, 91, 47] as 12+91+47 is 150
- if `A` is [15, 79, 99, 6, 69, 82, 32] and `s` is 150 it will return [69, 82] as 69+82 is 151, and there is no subset of `A` whose sum is 150.

Test the method with arrays generated by `randomIntArray(s,n)` from Lab 3. Try with:

```
A = randomIntArray(20,1000)
subset = closestSubset(5000,A)
```

Hint: You can use a naive recursive solution for this problem. That is, to find the closest subset adding to `s` starting from position `i`, your algorithm should compare the best solution that includes `A[i]` with the best solution that does not include `A[i]`, and so on. Such an algorithm is $O(2^n)$, with n the length of `A`, so it will take a significant amount of time to process an array of length 20, and get overwhelmed for arrays much longer than that.

Question 5

We next try a simple greedy algorithm for the problem of Question 4: start with an empty subset and go through each element in `A`, adding it to the subset if doing that would result in the subset's sum getting closer to the required sum.

For example for `A = [32, 79, 90, 91, 82, 7, 34, 12]` and `s = 150`, the algorithm will start with subset `[]` and:

- it will pick 32 and add it to the set, making it `[32]` (the sum of `[32]` is `32`, which closer to `150` than the sum of `[]` which is 0)
- it will then pick 79 and add it to the set, making it `[32,79]` (sum of `[32,79]` is `111` which is closer to `150` than `32`)
- it will then not pick 90, as sum of `[32,79,90]` is `211` which is further to `150` than `111`

and so on, until it will eventually return `[32,79,7,34]`.

Write code which implements this algorithm, and test it for how close it reaches the best answers and the time it takes compared to the method of Question 4.

This algorithm is $O(n)$, so it is quick, but it may not give the best solution. For example, for `A = [100, 150]` and `s = 150` the algorithm will return `[100]` instead of `[150]`.

Question 6

See if you can implement a greedy algorithm which gives results closest to the best answers than the algorithm given in Question 5, but works faster than the algorithm given in Question 4. Here is a possible idea:

Use the algorithm of Question 5 to provide an initial subset. Then, for each element in this subset, go through the elements of `A` which are not in the subset, and see if there is one that can replace it to give a subset with a sum closer to the required sum.

This algorithm is $O(n^2)$. It gives an improved solution to the one given by the Question 5 algorithm, but still cannot be guaranteed to give the best solution.

Question 7 (naive search with cutoff – no marks)

Write a Python function for coin splitting

```
def coinSplitCO(m)
```

that is a variation of the naive recursive function (given in the notebook of lecture 4) which cuts off any recursive call that cannot give a better result than the best result that has been already obtained via other recursive calls.

Hint: Use an auxiliary function

```
def coinSplitRecCO(m, startCoin, used, best)
```

that uses a 1-element array `best` to store the best coin splitting that has been achieved so far. Initially set `best` to `[None]`, and update it whenever a complete splitting is computed that has less coins than the one in `best`. Moreover, the input variable `used` stores the number of coins that have been used so far. At the beginning of each recursive call, if `used` is at least as big as the best value then the recursive call should simply return `None`.

Naive recursive solution (assumes last coin has value 1):

```
def coinSplit(m):
    return coinSplitRec(m,0)

def coinSplitRec(m, startCoin):
    if m == 0:
        return 0
    while coin[startCoin] > m:
        startCoin += 1
    if startCoin == len(coin)-1:
        return m
    withIt = 1 + coinSplitRec(m-coin[startCoin],startCoin)
    withoutIt = coinSplitRec(m,startCoin+1)
    return min(withIt,withoutIt)
```