Homework 3

Due by 4:30pm on September 30, 2020

Viewings and Readings

[Links to the slides, recordings, and quiz are on the course web site.]
Review the week 3 lecture slides.
View the week 3 lecture recordings and the discussion session recording.
Complete the week 3 quiz.
Read sections 2.1-4 of the textbook.

Problems

Please write precise and concise answers. Your algorithm descriptions should use either clear, concise, and precise plain English or clear, concise, and precise pseudo-code that uses a style similar to the pseudo-code in your textbook. Submit your solutions to problems **1-3(a)** via D2L as a Word or PDF file or as scans/photos of legible handwritten notes. Submit your solutions to problems **3(b)-4** via Kattis.

1. Problem 31 page 61 in your textbook.

Suppose we are given an array A[1 .. n] of n distinct integers, which could be positive, negative, or zero, sorted in increasing order so that A[1] < A[2] < ... < A[n].

a. Describe a fast algorithm that either computes an index i such that A[i] = i or correctly reports that no such index exists.

```
You can use binary search to solve this problem.
Taking array A as input
Recursively:
Get the midpoint of A and set it to a var
Base case is first check if A[i] = i, if it does return and thats our solution
If it doesn't and the array is size one return that no index of A[i] = i could be found

If A[mid index] < mid index, that means we can ignore all indexes on the left half of A so recuresivly call the funtion for the right half of A, A
If A[mid index] > mid mid index, that means we can ignore all indexes on the right half of A so recuresivly call the funtion for the left half of A

This will recuresivly keep going until either a A[i] = i is found or no index matches that criteria
```

b. Suppose we know in advance that A[1] > 0. Describe an even faster algorithm that either computes an index i such that A[i] = i or correctly reports that no such index exists. [Hint: This is really easy.]

```
If A[1] > 1 that means that there is no solution, since A[i] has to be an integer greater than A[i-1].
If A[1] = n where i > 1 then at a minimum for the rest of A A[i] = i+n

So the solution is to just check if A[1] = 1, if it does return one
Else there is no solution
```

2. Problem 1 page 93 in your textbook. Give *two* solutions for *each* part (a) and (b): you write the first solution by appropriately modifying the pseudo-code on page 78 of the textbook (that algorithm does not use pruning) and you should write the second solution my appropriately modifying the pseudo-code on Week 3 slide 64 (that algorithm does use pruning).

1. Describe recursive algorithms for the following generalizations of the SubsetSum problem
a. **Given an array X[1 .. n] of positive integers and an integer T, compute the number of subsets of X whose elements sum to T.**

SOLUTION 1:

```
SubsetSum(X, i, T):
if T = 0
  return 1
else if T < 0 or i = 0
  return 0
else
  with <- SubsetSum(X, i - 1, T - X[i]) {{Recurse!}}
  wout <- SubsetSum(X, i - 1, T) {{Recurse!}}
  return with + wout
```

SOLUTION 2:

```
SubsetSum(S[1..n], T, X[1..n], r)
  if r = n+1 then
    return 1
  X[r] <- 0
  if Feasible(S, T, X, r) then
    without = SubsetSum(S, T, X, r+1)
  X[r] <- 1
    if Feasible(S, T, X, r) then
    with = SubsetSum(S, T, X, r+1)
  return with + without;

Feasible(S[1..n], T, X[1..n], r)
  S1 <- SUM(i=1->r)X[i]S[i]
  if S1 > T then
    return false
  S2 <- SUM(i=r+1 -> n) S[i]
  if S1 + S2 < T then
    return false
```

```
    return true
```

b. **Given two arrays X[1 .. n] and W[1 .. n] of positive integers and an integer T, where each W[i] denotes the weight of the corresponding element X[i], compute the maximum weight subset of X whose elements sum to T. If no subset of X sums to T, your algorithm should return -infinity**

SOLUTION 1:

```
SubsetSum(X, W, i, T):
if T = 0
  return X[i] * W[i]
else if T < 0 or i = 0
  return 0
else
  with <- SubsetSum(X, i - 1, T - X[i]) {{Recurse!}}
  wout <- SubsetSum(X, i - 1, T) {{Recurse!}}

  // not positive on this implementation but we need to sum all the members of a valid subset
  if with > 0 with = with + X[i] * W[i]
  if wout > 0 wout = wout + X[i] * W[i]
  return max(with, wout)
```

SOLUTION 2:

```
SubsetSum(S[1..n], W[1...n], T, X[1..n], r)
  if r = n+1 then
    return S[r-1] * W[r-1]
  X[r] <- 0
  if Feasible(S, W, T, X, r) then
    without = SubsetSum(S, W, T, X, r+1)
  X[r] <- 1
    if Feasible(S, T, X, r) then
    with = SubsetSum(S, W, T, X, r+1)

  // not positive on this implementation but we need to sum all the members of a valid subset
  if with > 0 with = with + S[r-1] * W[r-1]
  if wout > 0 wout = wout + S[r-1] * W[r-1]
  return max(with, wout)

Feasible(S[1..n], T, X[1..n], r)
  S1 <- SUM(i=1->r)X[i]S[i]
  if S1 > T then
    return false
  S2 <- SUM(i=r+1 -> n) S[i]
  if S1 + S2 < T then
    return false
  return true
```

3. Week 3 problem *geppetto* on Kattis.

(a) Describe the backtracking algorithm for the *geppetto* problem. Make sure you describe precisely how partial solutions are represented and how they are pruned.

```
We can recursively call a function to make pizzas to check all possible valid combinations
The base case is when we reach the last ingredient which means weve made a decision of to include or not to include all previous ingredients
In the function check if the next pizza would be valid if the next ingredient is added, we can do this by saving all invalid combinations in a dictio
If the next ingredient is valid we can mark that ingredient is added in an array of n ingredients 1's and 0's, then add one to the pizza counter, the
If the next ingredient is invalid we can skip the step of recursively calling the next ingredient since all future ones will also be invalid
Then either way we set the ingredient to not used and recuresivly call the next ingredient so we test all combinations where we dont use that ingredi


The partial solution is checking whether the next ingredient would create an invalid pizza or not.  It its valid we can add one to the total pizzas c

If the ingredient would cause the pizza to be invalid, we can prune the rest of the recursion that would happen with that ingredient
```

(b) Implement your solution using your preferred language and submit your implementation via Kattis.

4. Week 3 problem *dancerecital* on Kattis. Submit your solution via Kattis.