Algorithms & Data Structures II (course 1DL231)
Uppsala University – Autumn 2017
Report for Assignment 1 by Team 60

Mona Mohamed Elamin Nevine Gouda

17th November 2017

1 The Birthday Present Problem

The Birthday Present can be considered as a subset sum problem. Where given a set of non-negative integers P, a value n which corresponds to the length of the set P, and a value t; we want to see if a subset $P' \subseteq P$ exists that sums to exactly the value t.

1.1 Recursive Equation

The Recursive implementation attempts to solve the problem as stated in Equation (4) by splitting the task into two sub tasks. Where one includes the last element in the set and subtracts its value from t, While the other task neglects it. And recursively it splits each task into 2 more tasks and sets n = n-1 with each iteration. And the base case will return True if and only if t becomes equal to 0.

However this approach will try all possible subsets in the worst-case. Which in return means that that the recursive algorithm has an exponential complexity. And a better approach would be solving it through Dynamic Programming.

$$BP(P,n,t) = \begin{cases} False & \text{if } t > 0 \text{ and } n = 0 \\ True & \text{if } t = 0 \\ BP(P,n-1,t) & \text{if } P[n-1] > t \\ BP(P,n-1,t) \| BP(P,n-1,t-P[n-1]) & \text{otherwise} \end{cases} \quad (1)$$

1.2 Dynamic Programming Implementation

In this implementation we create the dynamic programming 2 dimensional array and use the bottom up approach to fill it. In this approach we begin to fill the array using the base cases of the algorithm and then the rest of the array will be filled using previous known values. In other words, the value of 2 dimensional array[i][j] will be true if there is a subset of P[0...j-1] with sum equal to i., or false otherwise. Where P here is the input array. The final output is a Boolean to indicate if the provided list contain the desired subset or not and this output can be found in A[n][t] (the furthest bottom right cell in the 2D array). Our attempt in solving this using python can be found in Listing 1.

1.3 Birthday Present Subset

The first part of the algorithm is same to the previous one. However if a subset exists then it is returned otherwise an empty set is returned. we start from the last element in the dynamic array A[n][t] (the furthest bottom right cell in the 2 dimensional array). And then we try to determine which element in A causes it to be True A[x][y], then the element in input set which has index x-1 is added to the output subset. Afterwards we subtract the P[x-1] from sum. then we start from A[x-1][y-P[x-1]]. this will continue until the sum is equal to zero. Which will cause the algorithm to terminate and return the output subset. Our attempt in solving this using python can be found in Listing 2.

1.4 Complexity

To compute the time complexity we consider units of Time needed to complete the algorithm. this can be simplified by these steps:

1. Time needed to fill the dynamic array $A \approx t + (n + 1) + nt$.
2. The worst case to return the subset when the count decreases by one and i decreases until it reaches 0 therefore time needed $\approx t + n$.
3. Then the total time for the algorithm $\approx nt + 2(t + n) = O(nt)$.

2 Integer Sort Problem

Given a set of un-ordered positive integers, We should perform and in-place ordering and return the same list in a non-decreasing order.

2.1 Integer Sort implementation

Our attempt in solving this using python can be found in Listing 3. Where we create a counter list Y of size k, where k is the max value in the given list. Then we iterate over the given list, and for each element x in the input list we increment Y[x] by 1. Where at the end Y holds the number of times each element occurred in the input list. Then we iterate through Y and insert the indices with t times in the input list.

2.2 Complexity I

The worst case time complexity is computed following the next step:

1. Time needed to create the auxiliary array Y $\approx$ k+1.
2. Time needed to update $Y \approx c_1|A|$.
3. Time needed to sort $A \approx c_2(k + 1)$.
4. Total time needed by the algorithm $\approx (k + 1) + c_1|A| + c_2(k + 1) \approx c_1|A| + c_3k + c_4 = O(|A| + k)$
5. Therefore the algorithm has complexity of $O(|A| + k)$.

2.3 Complexity II

When $k = O(|A|)$ the time complexity becomes

$$O(|A| + |A|) = O(2|A|) = O(|A|).$$

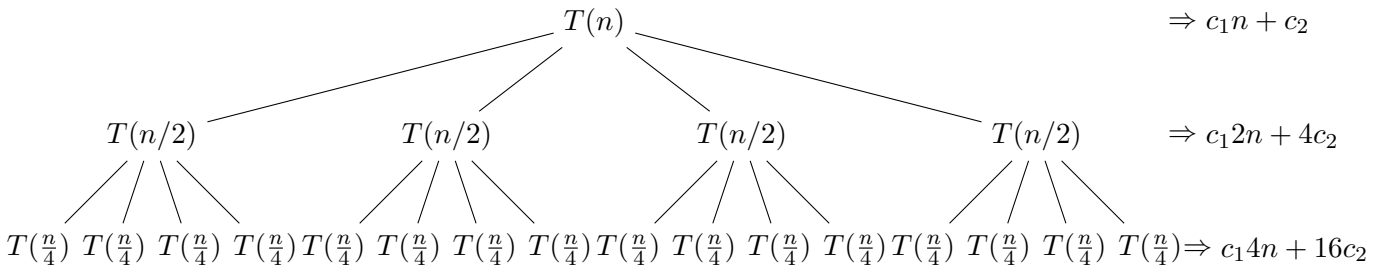
3 Binary Multiplication Problem

In this problem we are given 2 integer lists A, B. Where each element is either 1 or 0. And we want to perform binary multiplication A x B. Where if the input size of the arrays is n, then the expected output size should be an array of size 2n.

3.1 Divide And Conquer Approach

Out attempt to solve this using python can be found in Listing 4, Listing 5 and Listing 6. Where we took each list and recursively split each into 2 equal halves (Divide Phase). Where when we reach the base case we simply perform basic multiplication. Then Combine all the partial results using summation of the results and shifting them in the required manner (Conquer Phase).

3.2 Educated Complexity Guess



Based on the above figure we can conclude the total cost at any level i using Equation (2). Where i is the level, c_1 and c_2 are constants and n is the input size.

$$T_{\text{cost}}(i) = k_1.2^i.n + 4^i.k_2 \tag{2}$$

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log n} c_1 2^i n + 4^i c_2 \\ &= c_1.n\Big[\frac{2^{\log n+1} - 1}{2 - 1}\Big] + c_2\Big[\frac{4^{\log n+1} - 1}{4 - 1}\Big] \\ &= 2c_1 n.n - c_1 n + \frac{4c_2}{3}.n^2 - \frac{c_2}{3} \\ &\leq \quad 2c_1 n^2 \quad + \quad 4/3 c_2 n^2 \\ &= cn^2 \in \theta(n^2) \end{aligned} \tag{3}$$

3.3 Test Complexity Guess

Our guess: if T(n) satisfies the following recurrence then

$$T(n) = cn^2 + c_1 n + c_2 = \theta(n^2).$$

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 0 & \text{if } n = 1 \text{ and } (A[n-1] = 0 \text{ or } B[n-1] = 0) \\ 1 & \text{if } n = 1 \text{ and } (A[n-1] = 1 \text{ or } B[n-1] = 1) \\ 4T(n/2) + c_1 n + c_2 & Otherwise \end{cases} \tag{4}$$

Proof by induction

- **Base case**
  If $n = 1$ Then

$$T(n) = 0 \text{ or } T(n) = 1$$

- **Inductive hypothesis**
  Assume

$$T(n) = cn^2 + c_1 n + c_2$$

- **Goal**
  The goal is to prove

$$T(2n) = k(2n)^2 + 2k_1 n + k_2$$

$$\begin{aligned} T(2n) &= 4T(n) + 2c_1 n + c_2 \\ &= 4cn^2 + 4c_1 n + 4c_2 + 2c_1 n + c_2 \\ &= c(2n)^2 + 2k_1 n + k_2 \\ &= \theta(2n)^2 \end{aligned} \tag{5}$$

3.4 Master Theorem

The master theorem states that:

$$T(n) = aT(n/b) + f(n)$$

And our recursion function states:

$$T(n) = 4T(n/2) + c_1 n + c_2$$

Where it satisfies that

$$T(1) = \theta(1) \text{ and } T(0) = 0$$

Therefore the master theorem is applicable.
And looking in our algorithm we can see that

$$a = 4, b = 2$$
$$f(n) = c_1 n + c_2 = O(n) \geq k = 1$$
$$log_b a = log_2 4 = 2 > k$$

Therefore according to the master theorem

$$T(n) = \theta(n^2)$$

4 Honour Declaration

We hereby certify that this report and all its uploaded attachments were produced solely by our team, except where explicitly stated otherwise and clearly referenced. We hereby state that each teammate can individually explain any part starting from the moment of submitting our report, and that our report and attachments are not and will not be freely accessible on a public repository.

```
1  def birthday_present(P, n, t):
2      '''
3      Sig: int[0..n-1], int, int -> Boolean
4      Pre: P is a list of non-negative integers and t >= 0
5      Post: True if and only if a subset exists with summation equals t
6      Example: P = [2, 32, 234, 35, 12332, 1, 7, 56]
7               birthday_present(P, len(P), 299) = True
8               birthday_present(P, len(P), 11) = False
9      '''
10     # Initialize the dynamic programming matrix, A
11     # Type: Boolean[0..n][0..t]
12     try:
13         #Variant: i, j
14         #Invariant: A[i-1][j-1] the previous calculated element in never change
15         A = [[None for i in range(t + 1)] for j in range(n + 1)]
16
17         for j in range(1, (t + 1)):
18             #Variant: j
19             #Invariant: A[1..n-1][0..t-1] all elements in A not row 0, never change
20             A[0][j] = False
21
22         for i in range(0, (n + 1)):
23             #Variant: i
24             #Invariant: A[0..n-1][1..t-1] all elements in A not column 0, never change.
25             A[i][0] = True
26
27         for i in range(1, (n + 1)):
28             #Variant: i
29             #Invariant: none
30             for j in range(1, (t + 1)):
31                 #Variant: j
32                 #Invariant: A[0..i-1][0..j-1] all element in A previously calculated never change.
33                 if P[i - 1] > j:
34                     A[i][j] = A[i - 1][j]
35                 else:
36                     A[i][j] = A[i - 1][j - P[i - 1]] or A[i - 1][j]
37
38         return A[n][t]
39     except IndexError:
40         print "IndexError: Insert a list of positive integers only"
41         return False
```

Listing 1: Birthday Present

```
1  def birthday_present_subset(P, n, t):
2      '''
3      Sig: int[0..n-1], int, int -> int[0..m]
4      Pre: P is a list of non-negative integers and t >= 0
5      Post: list of a subset with a summation equals t and is empty if no subset exists
6      Example: P = [2, 32, 234, 35, 12332, 1, 7, 56]
7               birthday_present_subset(P, len(P), 299) = [56, 7, 234, 2]
8               birthday_present_subset(P, len(P), 11) = []
9      '''
10     try:
11         A = [[None for i in range(t + 1)] for j in range(n + 1)]
12         P_subset = []
13         count = t
14         for j in range(1, (t + 1)):
15             #Variant: j
16             #Invariant: A[1..n-1][0..t-1] all elements in A not in row 0, never change.
17             A[0][j] = False
18
19         for i in range(0, (n + 1)):
20             #Variant: i
21             #Invariant: A[0..n-1][1..t-1] all elements in A not column 0, never change.
22             A[i][0] = True
23
24         for i in range(1, (n + 1)):
25             #Variant: i
26             #Invariant: none
27             for j in range(1, (t + 1)):
28                 #Variant: j
29                 #Invariant: A[0..i-1][0..j-1] all elements in A previously calculated never change.
30                 if P[i - 1] > j:
31                     A[i][j] = A[i - 1][j]
32                 else:
33                     A[i][j] = A[i - 1][j - P[i - 1]] or A[i - 1][j]
34
35         if A[n][t] is True:
36             i = n
37             j = t
38             while count != 0:
39                 #Variant: i
40                 #Invariant: P_subset previously appended elements never change.
41                 if i < 0:
42                     break
43                 if A[i][j] is True and A[i - 1][j] is False:
44                     P_subset.append(P[i - 1])
45                     count = count - P[i - 1]
46                     j = j - P[i - 1]
47                     i = i - 1
48                 else:
49                     i = i - 1
50         return P_subset
51     except IndexError:
52         print "IndexError: Insert a list of positive integers only"
53         return []
```

Listing 2: Birthday Present Subset

```
1  def integer_sort(A, k):
2      '''
3      Sig: int array[1..n], int -> int array[1..n]
4      Pre: A is a list of non-negative integers and each element in A is either less than or equal k
5      Post: a non-decreasingly sorted version of array A
6      Example: integer_sort([5, 3, 6, 7, 12, 3, 6, 1, 4, 7]), 12) =
7                      [1, 3, 3, 4, 5, 6, 6, 7, 7, 12]
8      '''
9      try:
10         Y = [0] * (k+1)
11         for i in range(len(A)):
12             #Variant: i
13             #InVariant: elements of A never change.
14             x = A[i]
15             if x < 0:
16                 print "ValueError: Please enter a list of non-negative integers"
17                 return A
18             Y[x] += 1
19         j = 0
20         for x in range(len(Y)):
21             #Variant: x
22             #InVariant: elements of Y never change.
23             t = Y[x]
24             if t == 0:
25                 continue
26
27             for i in range(j,j+t):
28                 #Variant: i
29                 #InVariant: A[0..i-1] elements of A previously calculated never change.
30                 A[i] = x
31             j += t
32         return A
33     except IndexError:
34         print "IndexError: Insert a list of values less than or equal k"
35         return A
```

Listing 3: Integer Sort

```
1  def binary_mult(A, B):
2      """
3      Sig: int[0..n−1], int[0..n−1] ==> int[0..2∗n−1]
4      Pre: A and B are two lists of same size n
5      Post: returns a list of size 2n where n is the size of the input lists
6      Var: length(A) and length(B)
7      Example: binary_mult([0,1,1],[1,0,0]) = [0,0,1,1,0,0]
8      """
9
10     # Padding if one number is shorter/longer than the other
11     try:
12         if len(A) == 0 or len(B) == 0:
13             return []
14         if len(A) != len(B):
15             A, B = pad(A, B)
16
17         # The base case
18         if len(A) == 1:
19             return [A[0] ∗ B[0]]
20
21         n = len(A)
22         if n % 2:
23             # Padding so that we can split number in two equal halves
24             A = shiftR(A, 1)
25             B = shiftR(B, 1)
26             n += 1
27
28         # Splitting the lists into 2 halves each
29         Ah = A[0:n / 2]
30         Al = A[n / 2:]
31         Bh = B[0:n / 2]
32         Bl = B[n / 2:]
33
34         #Rercursive calls (Divide Stage)
35         a = binary_mult(Ah, Bh)
36         b = binary_mult(Ah, Bl)
37         c = binary_mult(Al, Bh)
38         d = binary_mult(Al, Bl)
39         b = binary_add(b, c)
40         a = shiftL(a, n)
41         b = shiftL(b, n / 2)
42
43         # Add the results upwards (Conquer Stage)
44         temp = binary_add(a, b)
45         ans = binary_add(temp, d)
46         return shiftR(ans, (2 ∗ n) − len(ans))[−2 ∗ n:]
47     except:
48         print "An Error Occured, please check your input values"
```

Listing 4: Binary Multiplication

```
1  def binary_add(A, B):
2      """
3      Sig: int[0..n−1] int[0..n−1] ==> int[0..n−1]
4      Pre: A and B are two lists of same size n
5      Post: returns the addition of A and B of size n where n is the size of the input lists
6      Example: binary_add([1,0,1],[0,1,0]) = [1, 1, 1]
7      """
8      if len(A) != len(B):
9          A, B = pad(A, B)
10     if sum(A) == 0:
11         return B
12     elif sum(B) == 0:
13         return A
14     ret = []
15     carry = 0
16     for i in range(len(A) − 1, −1, −1):
17         #Variant: i
18         #Invariant: A, B, ret[0..i−1]
19         if A[i] + B[i] + carry == 2:
20             carry = 1
21             ret.append(0)
22         elif A[i] + B[i] + carry == 3:
23             carry = 1
24             ret.append(1)
25         else:
26             ret.append(A[i] + B[i] + carry)
27             carry = 0
28
29     if carry:
30         ret.append(1)
31     return ret[::−1]
```

Listing 5: Binary Multiplication continued

```
1  def shiftR(A, n):
2      """
3      Sig: int[0..n−1], int n==> int[0..t+n−1]
4      Pre: n is a non negative integer
5      Post: returns the lists A with n zeros added on the left side
6      Example: shiftR([1,0,1],2) = [0, 0, 1, 0, 1]
7      """
8      if len(A) == 0:
9          return []
10     ret = [0] * n
11     return ret + A
12
13
14 def shiftL(A, n):
15     """
16     Sig: int[0..n−1], int n==> int[0..t+n−1]
17     Pre: n is a non negative integer
18     Post: returns the lists A with n zeros added on the right side
19     Example: shiftL([1,0,1],2) = [1, 0, 1, 0 ,0]
20     """
21     if len(A) == 0:
22         return []
23     ret = [0] * n
24     return A + ret
25
26
27 def pad(A, B):
28     """
29     Sig: int[0..n−1], int[0..m−1] ==> int[0..t−1]
30     Pre: A and B are two lists of non equal sized lists
31     Post: returns the lists A and B of the same size t
32     Example: pad([1,0,1],[1,0]) = [1,0,1],[0,1,0]
33              pad([0,1],[0,1,0]) = [0,0,1],[0,1,0]
34     """
35     # Padding if one number is shorter/longer than the other
36     rA = A[::]
37     rB = B[::]
38     if len(A) > len(B):
39         rB = shiftR(B, len(A) − len(B))
40     if len(B) > len(A):
41         rA = shiftR(A, len(B) − len(A))
42
43     return rA, rB
```

Listing 6: Binary Multiplication continued