CSCI 163 - Theory of Algorithms HW#3

Tushar Shrivastav

February 10, 2023

1 Problem 1

Problem: Write a brute force algorithm to solve the following problem:
INPUT: An array A[0....n-1] of n non-negative integers
OUTPUT: Return TRUE if there are two values that are exactly different by one.
Return FALSE if such values do not exist.
What is the asymptotic running time of your algorithm?

Solution:
The running time of the below algorithm is $O(n^2)$ as there are two for loops that run to the size of the input array.

Algorithm 1: Brute Force Algorithm

Data: An array $A[0...n-1]$ of n non-negative integers
Result: Return TRUE if there are two values that are exactly different by one. Return FALSE if such values do not exist.

```
for i ← 0 to n − 1 do
    for j ← i + 1 to n − 1 do
        if |A[i] − A[j]| = 1 then
            return TRUE;
        end
    end
end
return FALSE;
```

2 Problem 2

Problem:
Design an exhaustive search algorithm for this problem: Suppose n positive integers are given to you. Partition these numbers into two disjoint subsets in a way that both subsets have the same total sum. Try to make sure that your algorithm works for all possible inputs, you should consider cases when the input has no solutions and return appropriately. Note: If you need to generate all permutations or subsets, for example, you do not need to write an algorithm for the generation part. You can just write "generate all subsets", similar to what we did in class.
Solution:

Algorithm 2: Subset Sum Problem

Result: Returns True if a subset of A with sum equal to half of the total sum exists, False otherwise.

```
sum = 0;
for i ← 0 to n − 1 do
    sum += A[i];
end
if sum mod 2 = 1 then
    return FALSE;
end
half ← floor(sum/2)
sub[] ← generate_all_subsets(n/2)
for subset ← 0 to len(sub) do
    sum = 0
    for i ← 0 to (n/2) - 1 do
        sum += subset[i];
    end
    if sum = half then
        return TRUE;
    end
end
return FALSE;
```

3 Problem 3

Problem:
Write a top-down decrease-conquer algorithm to solve the following problem. INPUT: an integer array OUTPUT: the product of squares of all elements For example, if the array consists of the following elements: 5, 4, 3, 1, 9 the output is 5242321292= 291600 What is the asymptotic running time of your algorithm based on counting the number of multiplications? Show your work.
Solution:

Data: An array $A[0..n-1]$
Result: Product of squares of elements in $A[0..n-1]$

```
begin
    if n = 1 then
        return A[0] * A[0];
    end
    return (A[n − 1] * A[n − 1]) * square products[n − 1];
end
```

Asymptomatic Running Time
Recurrence Relation:

$$M(1) = 1$$

$$M(n) = 2 + M(n-1)$$

Backward Substitution:

$$M(n-1) = 2 + 2 + M(n-2])$$

$$M(n-2) = 2 + 2 + 2 + M(n-3)$$

Evident Pattern:

$$2i + M(n - i)$$
$$2n - 2 + M(n - n + 1)$$
$$2n - 2 + 1 = n$$

Thus,

$$2n - 1 \in \Theta(n)$$

4 Problem 4

Problem:
Write a pseudocode for a recursive (top-down) implementation of Insertion Sort using decrease-and-conquer technique. INPUT: integer array A[ ], int low and int high (the indices of the first and last element for array A.) **Solution:**

Algorithm 3: Recursive Insertion Sort

```
Data: A[low..high]: array to be sorted, with indices low and high
Result: A[low..high]: sorted array
function InsertionSortRecursive(A[low..high], low, high) begin
    if low < high then
        key ← A[low]
        j ← low − 1
        for i ← low to high do
            if A[i] < key then
                j ← j + 1
                swap(A[j], A[i])
            end
        end
        InsertionSortRecursive(A, low, j − 1)
        InsertionSortRecursive(A, j + 1, high)
    end
end
```

5 Problem 5

Problem:
Given a string s, and a list of strings called the dictionary, write a brute force algorithm that decides whether you can create the string susing the words in the dictionary. You are allowed to use the dictionary more than once.
Example1: s="beststudent"
dictionary = ['best', 'teacher', 'student', 'fast'] → output = True
Example2: s="fastcar"
dictionary = ['best', 'teacher', 'student', 'fast'] → output = False
The input to the algorithm is sand the dictionary. You can assume that you have a function that checks if a word is in the dictionary or not, IsInDictionary(word, dictio- nary) which returns true or false. Hint: It will be easier to implement this recursively.
Solution:

```
Data: string s and a list of strings dictionary
Result: Boolean indicating whether the string s can be created using words from dictionary
begin
    if len(s) = 0 then
        return TRUE;
    end
    for i ← 1 to len(s) do
        word ← ""; for j ← 0 to i − 1 do
            word ← word + s[j];
        end
        if IsInDictionary(word, dictionary) = TRUE then
            if canCreateString(s.substr(i), dictionary) = TRUE then
                return TRUE;
            end
        end
    end
    return FALSE;
end
```