Exam 1 Solutions – Winter 17

5. (14 points total)

*(a) Give pseudo-code for a **recursive** algorithm that computes 2^n for any nonnegative n based on the formula below and the fact that $2^0 = 1$.* **(8 points)**

$$2^n = 2^{n-1} + 2^{n-1}$$

Power(n)

if n=0 return 1 **(-2 for not starting at 0)**

else return Power(n-1) + Power(n-1) **(uses n/2 -3 points)**

(b) *Set up a recurrence relation* **(3 points)**

T(n) = 2T(n-1) + 1 or

T(n) = 2T(n-1) + c

solve it. **(2 points)**

$\Theta(2^n)$ **(big O -1)**

(c) *Is this a good algorithm for solving this problem?* (1 point)

No can be done in theta(lgn) or theta(n)

Power(n)

if n=0 return 1

else return 2 Power(n-1) + Power(n-1)*

T(n) = T(n-1) + c

$\Theta(2^n)$

Exam 1 Solutions – Winter 17

6.

```
Foo(n) {
  total = 0
  if n = 1 return 2
  else {
     total = Foo(n/4) + Foo(n/4)
     for i = 1 to n do
          for k = 1 to n do
               total = total + k
      return total  }
}
```

(a) Write a recurrence for the running time T(n) of Foo(n)

$T(n) = 2T(n/4) + n^2$ or

$T(n) = 2T(n/4) + c\,n^2$ or

$T(n) = 2T(n/4) + \Theta(n^2)$ **(5 points)**

(-1 for each wrong term)

(b) Solve the recurrence for the asymptotic running time. Assume that addition can be done in constant time. Give the tightest bound possible.

Using the master method: *Can use another method but must be Theta* **(-1 for bigO only)**

$a = 2, b = 4, \log_4(2) = 0.5, f(n) = n^2$ **(1 points)**

Case 3:

Case 3 of the master method

$n^2 = \Omega(n^{0.5+e})$ for any $e < 1.5$ **(1 point)**

This is case 3, so we will check the **regularity** condition to see if this holds.

$2f(n/4) = 2(n/4)^2 = 1/8n^2 \leq cn^2$ if $c = 0.5$ many choices of c are possible since n is positive. **(1 point)**

Therefore f the regularity condition holds, so $T(n) = \text{theta}(n^2)$ **(2 point)**

May also use iteration or recursion tree full credit for showing steps.

Exam 1 Solutions – Winter 17

7.

The ternary search algorithm is a modification of the binary search algorithm that splits the input not into two sets of almost-equal sizes, but into three sets of sizes approximately one-third.

a) **(8 points)** Write pseudo-code for the ternary search algorithm. Below is an example of pseudocode

```
ternarySearch (A, value, start, end)
{
        if (start > end) {
            return false;
        }

        // First boundary: add 1/3 of length to start.
        third1 = start + (end-start) / 3;

        // Second boundary: add 2/3 of length to start.
        third2 = start + 2*(end-start) / 3;

        if (A[third1] == value) {
            return true;
        }
        else if (A[third2] == value) {
            return true;
        }
        else if (value < A[third1]) {
            // Search 1st third.
            return ternarySearch (A, value, start, third1-1);
        }
        else if (value > A[third2]) {
            // Search 3rd third.
            return ternarySearch (A, value, third2+1, end);
        }
        else {
            // Middle third.
            return ternarySearch (A, value, third1+1,third2-1);
        }
    }
```

b) Give the recurrence for the ternary search algorithm and determine the asymptotic running time of your algorithm.

$T(n) = T(n/3) + \Theta(1)$ or $T(n) = T(n/3) + k$ or $T(n) = T(n/3) + 4$ **(3 points)**

Solution:

$\Theta(\log n)$ or $\Theta(\lg n)$ or $\Theta(\log_3 n)$ **(3 points)**

Exam 1 Solutions – Winter 17

8. (16 points total) Scheduling jobs with bonuses:

Your new employer is very generous and gives bonuses for completing jobs on or before their due dates. This employer never gives a penalty for finishing a job late. Each job j_i ($1 \le i \le$) has two numbers associated with it, di and bi, where di is the due date and bi is the bonus. The length of each job is 1 day, each job can be scheduled on only one day and you can only work on one job per day. If job i completes on or before its due date di, you receive a bonus b_i. Your goal is to schedule all of the jobs in a way that maximizes the total bonus you receive.

(a) What type of algorithm would you use to solve this problem? Divide and Conquer, Greedy or Dynamic Programming. Why?

(b) Describe the algorithm verbally and give pseudo code with enough detail to obtain the running time. If you select a DP algorithm give the formula used to fill the table or array.

(c) What is the running time of your algorithm? Explain.

(a) **(4 points)** What type of algorithm would you use to solve this problem? **Greedy**

(b)) **(8 points)** Describe the algorithm verbally with enough detail to obtain the running time. If you select a DP algorithm give the formula used to fill the table or array.

Note since there are n days it will take at most n days

1. Sort jobs by bonuses in decreasing order. You will select the biggest bonus first

2. For i = 1 to n. Select the next job j_i to be scheduled according to the sorted bonus order. There are two outcomes

   a) The job can be completed early and you get the bonus Schedule the job j_i as close to the early completion date e_i as possible without exceeding the date. You get the bonus b_i
   b) The job cannot be scheduled by the early completion date. If there are no days left on or before e_i to schedule job j_i then schedule the job at the first available day closest to day n. The job is put at the end of the queue so not to conflict with other jobs.

(c)) **(4 points)** What is the running time of your algorithm? Explain.

1. The sorting takes Θ(nlgn)

2. To find the spot in the schedule can take in worst case 1+2+...+ n = $\Theta(n^2)$ depending on the data structure used.

Exam 1 Solutions – Winter 17

9. **(16 points total)**

OSU student, Benny, is taking his CS 325 algorithms exam which consists on n questions. He notices that the professor has assigned points { p_1, p_2, ..., p_n } to each problem according to the professor's opinion of the difficulty of the problem. Benny wants to maximize the total number of points he earns on the exam, but he is worried about running out of time since there is only T minutes for the exam. He estimates that the amount of time it will take him to solve each of the n questions is { t_1, t_2, ..., t_n }. You can assume that Benny gets full credit for every question he answers completely. Develop an algorithm to help Benny select which questions to answer. **Note:** NO partial credit is assigned to problems that are only partially completed.

(a) **(4 pts)** What type of algorithm would you use to solve this problem? Divide and Conquer, Greedy or **Dynamic Programming**. Why?

Note: This is similar to the 0-1 knapsack problem and a DP method will be optimal. Greedy will not be optimal since there is no partial credit.

(b) **(8 points)** Describe the algorithm verbally and give pseudo code with enough detail to obtain the running time. If you select a DP algorithm give the formula used to fill the table or array.

Let P[k, t] be the maximum number of points with questions {1, 2,.., k} and time t **(2 pt)**

Initialization:

P[0, t] = 0 for t = 0 to T **(1 pt)**

P[k, 0] = 0 for k = 1, .., n **(1 pt)**

For k = 1 to n

For t = 1 to T

If $t_k > t$

P[k, t] = P[k-1, t]; **(1 pt)**

Else

P[k, t] = max { P[k-1, t], P[k-1, $t-t_k$] + p_k } **(3 pt)**

(c) What is the running time of your algorithm? Explain. **(4pt)**

Theta(nT) where T is the time and n is the number of questions. This a the time to fill the table and is a pseudo-polynomial solution.

Exam 1 Solutions – Winter 17

10. (14 points total)

Papa Mario of Mario's Pizzeria has baked a huge pizza and cut it into n slices, but he is clumsy and the pizza wasn't evenly sliced. The n slices have size $s_1,s_2,...,s_n$. There are n hungry students who each want to eat a slice of pizza. Suppose the i^{th} student would be happy with any slice whose size is at least t_i. Give an efficient algorithm to determine whether it is possible to distribute the pizza slices so everyone is happy.

(a) What algorithm paradigm is most appropriate for this problem? **(2 points)**

- **Greedy algorithm**

(b) Verbally describe for your algorithm. Several variations are correct. (**8 points)**

Step1: Sort the pieces of pizza by size, largest to smallest ($s_1,s_2,...,s_n$)

Step 2: Sort the students by the size of slice they want largest to smallest. ($t_1,t_2,...,t_n$)

Step 3: Of the students in the list without pizza, give the n student who wants the largest slice t_j the the largest slice of available pizza s_j.

Step 4:

- Case1: If sj >= tj then that student is happy. The student and slice can be removed from the lists. Go to step 3 and repeat until all pieces have be distributed. Everyone is **happy**.
- Case 2: if at any time sj < tj then a slice is too small for a student. Stop the process. It is impossible to make everyone happy

(c) What is the asymptotic running time of your algorithm? (**4 points)**

Time to sort the lists of students and slices. Θ(nlgn)

There are other solutions with a greater running time -2 for $O(n^2)$