CS 325 Midterm Practice Problems - Solutions

1. (True/False) The running time of a dynamic programming algorithm is always Θ(P) where P is the number of subproblems.

Solution: False. The running time of a dynamic program is the number of subproblems times the time per subproblem. This would only be true if the time per subproblem is $O(1)$.

2. If f(*n*) = O(g(*n*)) and g(*n*) = O(h(*n*)), then f(*n*) = O(h(n)) True

3. Give asymptotic upper and lower bounds. Make as tight as possible.

T(n) = 2 T(n/3) + nlgn

Solution: By Case 3 of the Master Method, we have $T(n) = \Theta(n \lg n)$.

4. What does the fact given below imply regarding big-O, big-Ω and/or big-Θ relationships between the functions.

For all n > 40, 3g(n) ≤ f(n) ≤ 5g(n)

Solution: f(n) is Θ(g(n)) which implies that g(n) is Θ(f(n))

5. Order the following functions in increasing order of asymptotic (big-O) complexity.

$$f(n) = 2^{2^{1000}}, \quad g(n) = \sum_{i=1}^{n}(i+1), \quad h(n) = 2^n, \quad p(n) = 10^{10}n, \quad q(x) = n2^{n/2}$$

Solution:

f(n), p(n), g(n) q(n), h(n)

6. Show log(n!) is O(nlogn)

Solution:

log(n!) = log (n*(n-1)*(n-2)*…2*1)

= logn + log(n-1) + log(n-2) + .. + log 2 + log1 < nlogn

Therefore, O(nlogn)

CS 325 Midterm Practice Problems - Solutions

7. What is the asymptotical running-time complexity of Find-Array-Max?

```
Function FIND-ARRAY-MAX(A, n)
  if (n = 1) then
     return(A[1])
  else
     return(max(A[n], FIND-ARRAY-MAX (A, n − 1) ))
  end if
```

Solution:

$T(n) = T(n-1) + 1$, $T(1) = 0$.

$T(n) = \Theta(n)$

8. Mr. Smith has an algorithm which he has proved (correctly) to run in time $O(2^n)$. He coded the algorithm correctly in C, yet he was surprised when it ran quickly on inputs of size up to a million. What are at least two plausible explanations of this behavior?

Solution:

- Perhaps the algorithm runs in linear time. $O(2^n)$ means at most time $c2^n$. Even a linear algorithm is $O(2^n)$.
- Perhaps the input to the program is not the worst case input. It could be the "easiest" input. The exponential behavior doesn't "kick in" until n is huge.

9. Given a set $\{x_1 \leq x_2 \leq \ldots \leq x_n\}$ of points on the real line, determine the smallest set of unit-length closed intervals (e.g. the interval [1.25,2.25] includes all x_i such that $\{1.25 \leq x_i \leq 2.25\}$) that contains all of the points. Give the most efficient algorithm you can to solve this problem, prove it is correct and analyze the time complexity.

The greedy algorithm we use is to place the first interval at $[x_1, x_1 + 1]$, remove all points in $[x_1, x_1 + 1]$ and then repeat this process on the remaining points.

Clearly the above is an $O(n)$ algorithm. We now prove it is correct.

Greedy Choice Property: Let S be an optimal solution. Suppose S places its leftmost interval at $[x, x + 1]$. By definition of our greedy choice $x \leq x_1$ since it puts the first point as far right as possible while still covering x_1. Let S' be the scheduled obtained by starting with S and replacing $[x, x + 1]$ by $[x_1, x_1 + 1]$. We now argue that all points contained in $[x, x + 1]$ are covered by $[x_1, x_1 + 1]$. The region covered by $[x, x + 1]$ which is not covered by $[x_1, x_1 + 1]$ is $[x, x_1)$ which is the points from x up until x_1 (but not including x_1). However, since x_1 is the leftmost point there are no points in this region. (There could be additional points covered by $[x + 1, x_1 + 1]$ that are not covered in $[x, x + 1]$ but that does not affect the validity of S'). Hence S' is a valid solution with the same number of points as S and hence S' is an optimal solution.

CS 325 Midterm Practice Problems - Solutions

Optimal Substructure Property: Let P be the original problem with an optimal solution S. After including the interval $[x_1, x_1 + 1]$, the subproblem P' is to find an solution for covering the points to the right of $x_1 + 1$. Let S' be an optimal solution to P'. Since, $\text{cost}(S) = \text{cost}(S') + 1$, clearly an optimal solution to P includes within it an optimal solution to P'.

10. You are going on another long trip (this time your headlights are working). You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1, < a_2 < \ldots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance an), which is your destination.

You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the penalty for that day is $(200\text{-}x)^2$. You want to plan your trip so as to minimize the total penalty – that is the sum, over all travel days, of daily penalties. Give an efficient algorithm that determines the minimum penalty for the optimal sequence of hotels at which to stop.

Solution:

Let S[j] be the minimum total penalty when you stop at hotel j.

Let S[0] = 0
For j >= 1, j <= n
    S[j] = inf
    For i = 0, i < j
        S[j] = min { S[j], S[i] + $(200\text{-} (a_j – a_i))^2$ }
Return: S[n]

11. (True / False)

a) $n^2 + n = O(n^2)$? - **true**

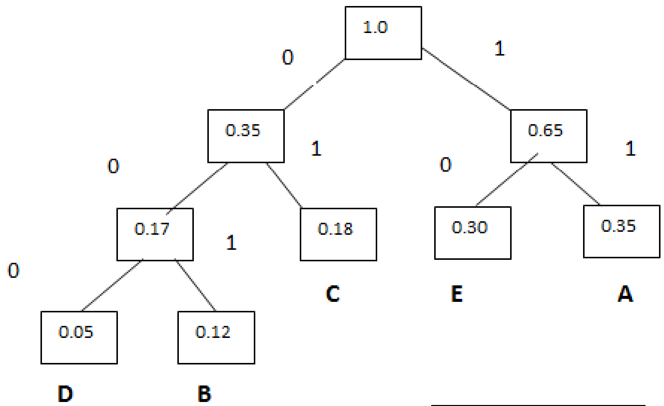
b) $\lg(n^2) = O(n)$? **true**

c) A function that calls a O(n) function three times and has constant time for the rest of the algorithm. The overall asymptotic run-time of the algorithm is O(n). **True**

CS 325 Midterm Practice Problems - Solutions

12. Suppose we have an alphabet with only five letters A, B, C, D, E which occur with the following frequencies. Construct a Huffman code.

Letter	A	B	C	D	E
frequency	0.35	0.12	0.18	0.05	0.30



Letter	encoding
A	11
B	001
C	01
D	000
E	10

13. For each of the following give a tight Θ( ) bound on the number of times the z ← z + 1 statement is executed and justify your solution.

$j \leftarrow 0$
while $(j < n)$ **do**
  $j \leftarrow j + 2$
  $z \leftarrow z + 1$

AN ANSWER. Since j goes through the values 0, 2, 4, 6, ... until j reaches n (if n is even) or $n+1$ (if n is odd), the while loop goes through at most $\lceil n/2 \rceil$ many iterations. Hence, z is in $\Theta(n)$.

for $k \leftarrow 0$ **to** n **do**
  for $j \leftarrow 0$ **to** k **do**
    $z \leftarrow z + 1$

AN ANSWER. *Inner loop:* Since j goes from 0 to k, the inner loop has $(k+1)$-many iterations. So, z is increased by $(k+1)$. *Outer loop:* k goes from 0 to n. So z is increased by

$$\sum_{k=0}^{n}(k+1) = 1 + 2 + \cdots + n + (n+1) = \frac{(n+1)(n+2)}{2} \in \Theta(n^2).$$

$i \leftarrow n$
while $(i > 1)$ **do**
  $i \leftarrow \lfloor i/2 \rfloor$
  $z \leftarrow z + 1$

AN ANSWER. Since i takes on the sequence of values $n = n/2^0$, $\lfloor n/2 \rfloor = \lfloor n/2^1 \rfloor$, $\lfloor n/4 \rfloor = \lfloor n/2^2 \rfloor$, ..., $\lfloor n/2^j \rfloor$ until i is ≤ 1. The smallest value of i such that $n/2^i \leq 1$ is $\lceil \log_2 n \rceil$. So there are $(1 + \lceil \log_2 n \rceil)$-many iterations and $z \in \Theta(\log_2 n)$.

CS 325 Midterm Practice Problems - Solutions

14. You just started a consulting business where you collect a fee for completing various types of projects (the fee is different for each project). You can select in advance the projects you will work on during some finite time period. You work on only one project at a time and once you start a project it must be completed to receive your fee. There is a set of n projects $p_1, p_2, .. p_n$ each with a duration $d_1, d_2, .. d_n$ (in days) and you receive the fee $f_1, f_2, .., f_n$ (in dollars) associated with it. That is project p_i takes d_i days and you collect f_i dollars after it is completed.

Each of the n projects must be completed in the next D days or you lose its contract. Unfortunately, you do not have enough time to complete all the projects. Your goal is to select a subset S of the projects to complete that will maximize the total fees you earn in D days.

(a) What type of algorithm would you use to solve this problem? Dynamic Programming. Why? It is similar to the 0-1 knapsack.

(b) Describe the algorithm verbally. If you select a DP algorithm give the formula used to fill the table or array.

Let OPT(i,d) be the maximum fee collected for considering projects 1,…, i with d days available.

The base cases are OPT(i, 0) = 0 for i = 1, …, n and OPT(0, d) = 0 for d = 1, …, D.

```
For i = 1 to n {
  For j = 1 to D {            //updated from d to j
        If  (di > j ) {
                 OPT(i, j) = OPT( i-1, j)    // not enough time to complete project i }
        Else  {
                 OPT(i, j) = max (  OPT( i-1, j),    //  Don't complete project i
                                     OPT( i-1, j-di) + fi  ) //  Complete project I and earn fee fi
                                   )
        }
   }
}
```

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

$O(nD)$ or $\Theta(nD)$

CS 325 Midterm Practice Problems - Solutions

15. Rod Cutting**:** (from the text CLRS) 15.1-2 – Many possible solutions

Here is a counterexample for the "greedy" strategy:

length i	1	2	3	4
price p_i	1	20	33	36
p_i/i	1	10	11	1

Let the given rod length be 4. According to a greedy strategy, we first cut out a rod of length 3 for a price of 33, which leaves us with a rod of length 1 of price 1. The total price for the rod is 34. The optimal way is to cut it into two rods of length 2 each fetching us 40 dollars.

16. Modified Rod Cutting: (from the text CLRS) 15.1-3

```
MODIFIED-CUT-ROD(p, n, c)
  let r[0..n] be a new array
  r[0] = 0
  for j = 1 to n
      q = p[j]
      for i = 1 to j - 1
          q = max(q, p[i] + r[j - i] - c)
      r[j] = q
  return r[n]
```

The major modification required is in the body of the inner **for** loop, which now reads $q = \max(q, p[i] + r[j-i] - c)$. This change reflects the fixed cost of making the cut, which is deducted from the revenue. We also have to handle the case in which we make no cuts (when i equals j); the total revenue in this case is simply $p[j]$. Thus, we modify the inner **for** loop to run from i to $j-1$ instead of to j. The assignment $q = p[j]$ takes care of the case of no cuts. If we did not make these modifications, then even in the case of no cuts, we would be deducting c from the total revenue.