Fundamentals of Algorithms
CS502-Fall 2011

SOLUTION ASSIGNMENT #2

Deadline

Your assignment must be uploaded/submitted at or before **21st Nov. 2011**

Uploading instructions

Please view the **assignment submission process** document provided to you by the Virtual University to upload the assignment.

Rules for Marking

It should be clear that your assignment will not get any credit if:

- The assignment is submitted after due date.
- The submitted assignment does not open or run.
- **The assignment is copied.**

Objectives

This assignment will help you to understand the concept of recurrence relations and way to solve then and writing asymptotic notation after analyzing and solving recurrences .The other main focus is to learn dynamic programming applications and edit distance problem solution using dynamic programming technique which will be ultimately enhance your vision and logics to think critically and analytically.

Guidelines

1. In order to attempt this assignment you should have full command on Lecture #5-9 and Lecture # 15-16
2. In order to solve this assignment you have strong concepts about following topics
   - ✓ Recurrence Relations and growth rate of the functions
   - ✓ Radix sort
3. Normally these formulas are very handy:

If $x^y = z$ then $y = \log_x z$

$a^{\log_b n} = n^{\log_b a}$

Also

$\sum_{i=1}^{n} a_i = \frac{n}{2}(a_1 + a_n)$ $\sum_{i=1}^{n} i = \frac{n}{2}(n+1)$ $\sum_{k=0}^{m} r^k = \frac{1-r^{m+1}}{1-r}$

$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ (for n >=1)

Some basic information for solving assignment question 1 is given below.

Growth Rate of Function:

If some function $f_1(n) > f_2(n)$ for positive values of x then the function $f_1(x)$ is said to have greater growth rate then $f_2(x)$. For example $f_1(n)=n^{100}$ and $f_2(n)= n^{99}$ it is obvious that $f_1(x)$ has greater growth rate ($2^{100} > 2^{99}$).This concept relate to complexity of algorithm ,an algorithm having greater growth rate function means the algorithm has greater complexity here $f_1(x)$ is more complex then $f_2(x)$.

Books to read for solution

Cormen, Leiserson, Rivest, and Stein (CLRS) 2001, **Introduction to Algorithms**, (2nd ed.) McGraw Hill.

Estimated Time 3.5 hours

Your concepts and logics will take actual measure of time ;however first question should not take more than 2 hour and for question 2 you may solve in 1.5 hours It all depends upon your sheer concentration.

Question# 1 **(10)**

Arrange the following in the **Least to Most** complexity order. Here "n "is the input size for the some complexity function and j< k and j & k are numbers greater than 2.Every function is separated by "comma" and note these are 20 functions to arrange.

$n/10000,\ 10n^{6k/2},\ n^{12j/4},\ \sqrt{n}\lg n,\ n^n,\ 1000000000,\ 2^n,\ \sqrt[7]{n}\lg n,\ n!,\ (2^n n\sqrt{n})/\sqrt{n^2},$

$n!/\sqrt{n},\ 2^n n\log n/\sqrt{n^2},\ n!/\log n,\ 1000000,\ n/\sqrt[4]{n},\ n(\log n)\sqrt[9]{n},$

$n/\sqrt[11]{n},\ n(\log n)\sqrt[3]{n},\ k^n,\ j^n$

Hints toward solution

- Think in more general way.
- If you feel difficulty for comparison then judge for lager inputs say 10^{100} etc.
- Note 2^n and n! are both complex but you can judge that n! is more complex by putting values of larger "n" .
- Further n! in generic terms also beats functions like k^n, j^n although it feels very strange for large values of "j" and "k" but the core point need to be remembered that taking the "j" and "k" very large will become constant for particular scenario and we have "variable " "n" and there will time come when "n" becomes so large as then ordinary imagination then "n!" will beat so we consider "n!" more complex then the functions k^n, j^n .AND if some students have assumed the alternate then partial marks will be rewarded for their thinking.

Note: Here one amazing thing the most complex one function is of the nature n^n factorial definition and its expansion shows the results:
n! =n (n-1)(n-2)……..2.3.1.
While
n^n = n (n)(n)……n which will obviously beat n!
Think for the vision enhancements and think beyond the boundaries.

Given Order	Least to Most Complexity Arrangements	Hints
$n/10000$	**1000000**	**Fixed values are considered efficient as for very large "n" these become so small.**
$10n^{6k/2}$	**1000000000**	
$n^{12j/4}$	$\sqrt[7]{n}\ \lg n$	**Think as if** $(\sqrt{n}\ \sqrt{n} = n) < \sqrt[7]{n}\ \lg n$ **as here both terms are less than** $\sqrt{n}$
$\sqrt{n}\lg n$	$\sqrt{n}\lg n$	
n^n	$n\ /\sqrt[4]{n}$	
1000000000	$n\ /\sqrt[11]{n}$	**Note here the dividing factor is small than** $\sqrt[4]{n}$ **That's why answer will be larger as compared to above one.**
2^n	$n/10000$	**Although for small values of "n" it looks efficient than above ones but when we increase the size at larger extent it is at its true place i. e. find such "n" where** $\sqrt[11]{n} > 10000$
$\sqrt[7]{n}\ \lg n$	$n(\log n)\sqrt[9]{n}$	
$n!$	$n(\log n)\sqrt[3]{n}$	**obviously as** $\sqrt[9]{n} < \sqrt[3]{n}$

$(2^n n\sqrt{n})/\sqrt{n^2}$	$n^{12j/4}$	**As j<k and note here what ever the value of "j" or "k" will be taken it will become constant then the variable "n" for very large values of "n" you can compare the complexities of these with other functions**
$n!/\sqrt{n}$	$10n^{6k/2}$	
$2^n nlogn/\sqrt{n^2}$	2^n	
$n!/logn$	$2^n nlogn/\sqrt{n^2}$	
1000000	$(2^n n\sqrt{n})/\sqrt{n^2}$	
$n\ /\sqrt[4]{n}$	j^n	**As "j " and "k " are greater than "2"**
$n(logn)\sqrt[9]{n}$	k^n	
$n\ /\sqrt[11]{n}$	$n!/\sqrt{n}$	
$n(logn)\sqrt[3]{n}$	$n!/logn$	**As divider function is smaller i.e.** $\sqrt{n} > logn$
k^n	$n!$	
j^n	n^n	

Question# 2 **(10)**

Carry out the radix sort on the following five digits numbers and also develop complexities function and then write worst case Theta Θ notation for the radix sort algorithm.

45141,16545,11478,12196,12133,21322,31422,31511,11262,27210

Solution table stepwise:

Input	1st Iteration	2nd Iteration	3rd Iteration	4th Iteration	Fifth iteration	output
45141	**2721[0]**	**272[1]0**	**12[1]33**	**1[1]262**	**[1]1262**	**11262**
16545,	**4514[1]**	**315[1]1**	**12[1]96**	**2[1]322**	**[1]1478**	**11478**
11478,	**3151[1]**	**213[2]2**	**45[1]41**	**3[1]422**	**[1]2133**	**12133**
12196,	**2132[2]**	**314[2]2**	**27[2]10**	**1[1]478**	**[1]2196**	**12196**
12133,	**3142[2]**	**121[3]3**	**11[2]62**	**3[1]511**	**[1]6545**	**16545**
21322,	**1126[2]**	**451[4]1**	**21[3]22**	**1[2]133**	**[2]1322**	**21322**
31422,	**1213[3]**	**165[4]5**	**31[4]22**	**1[2]196**	**[2]7210**	**27210**
31511,	**1654[5]**	**112[6]2**	**11[4]78**	**4[5]141**	**[3]1422**	**31422**
11262,	**1219[6]**	**114[7]8**	**31[5]11**	**1[6]545**	**[3]1511**	**31511**
27210	**1147[8]**	**121[9]6**	**16[5]45**	**2[7]210**	**[4]5141**	**45141**

Radix-Sort
A stable sort algorithm for sorting elements with *d* digits keys, where each digit is in base *b*, meaning in the range [0,*b*-1].
The algorithm uses a stable sort algorithm to sort the keys according to each digit starting with the least significant digit (rightmost).
Radix-Sort(A[1..n]) **for i ← 1 to d** **Use stable sort to sort A on digit i**
Analysis of Algorithim
Run-time Complexity: **Assuming the stable sort runs in O(n+b) (such as counting sort) the running time is O(d(n+b)) = O(dn+db). And average case analysis is also Θ(d(n+b)) but** **If d is constant and b=O(n), the running time is O(n).**