STAT 598Z Midterm

March 08, 2012

Time: 75 minutes

Your Name (Please print): _____

PUID (Please print): _____

Note:

1. This exam will contribute 10 points towards your final score
2. Use the provided scratch paper for your calculations
3. Take a print out of your code and attach it along with all relevant calculations
4. **Write your name clearly and legibly on all printouts**
5. Attempt as many problems as possible and explicitly state all your assumptions.
6. Show all intermediate steps for full credit. Python code must be clear and concise.
7. Any attempt at academic dishonesty (e.g. using a browser during the exam) will automatically result in 0 points.
8. Use of notes, books, laptops, cell phones, or any other aids (electronic or otherwise) is strictly prohibited. Turn off and put away your cell phone now!

Please sign below to indicate your agreement with the following honour code.

Honour code: I promise not to cheat on this exam. I will neither give nor receive any unauthorized assistance. I promise not to share information about this exam with anyone who may be taking it at a different time. I have not been told anything about the exam by someone who has already taken it.

Signature:_____ Date:_____

Questions	Possible Points	Actual Points
1	2	
2	3	
3	2	
4	3	
Total	**10**	

Problem 1 (2 pt) Recall that a palindrome is a string which spells the same if its spelt from front or back. For example `madam` is a palindrome. Write two Python functions `palindrome_for` and `palindrome_which` which use a for loop and a while loop respectively to check if a given input string is a palindrome. Your functions must take a string as input and return True if the string in a palindrome and False otherwise. Print the output of both your functions on the following strings:

- `madam`
- `amma`
- `abcde`
- `aradar`

Solution 1: See the attached code.

Problem 2 (1.5 + 0.5 + 1 pt) Recall the mergesort algorithm we discussed in the class. One of the key components of mergesort was a sub-routine for merging **two** sorted lists.

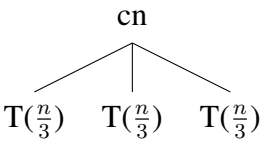
- Write a python function `threeway_merge` which takes as input **three** sorted lists `a`, `b`, and `c` and merges them to produce another sorted list `d`.
- What is the time complexity of your algorithm in O notation. Provide clear and concise arguments.
- Your friend claims that her mergesort algorithm which is based on `threeway_merge` is asymptotically better than the standard mergesort based on merging two lists at a time. Do you agree? Justify your stand in a mathematically precise way by using the O notation.

Solution 2:

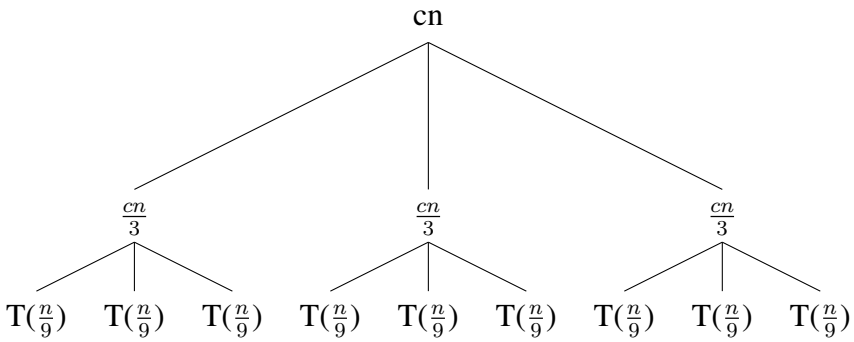
- See the attached code. Here, we apply `twoway_merge` twice. Once on the lists `a` and `b` to produce a sorted list `l` and then merge `l` and `c` to produce the final sorted list `d`.
- Suppose the size of the three lists `a`, `b`, and `c` be n_a,n_b and n_c respectively. Then merging `a` and `b` requires $O(n_a + n_b)$ comparisons. The two soted lists `l` and `c` after this step have sizes n_a+n_b and n_c respectively. The second step of merging `l` and `c` will take $O(n_a+n_b+n_c)$ comparisons. So, the total time complexity of `threeway_merge` will be $O(n_a + n_b) + O(n_a + n_b + n_c) = O(n_a + n_b + n_c)$.
- The time taken by mergesort algorithm using `threeway_merge` to sort n numbers can be written as the following recurrence:

$$T(n) = 3T(\frac{n}{3}) + cn \tag{1}$$

Here, cn denotes the time taken to merge three sorted sub-array of size $[\frac{n}{3}]$ into one single sorted array. Note that, to sort an array of size n, we need to sort three sub arrays of size $\frac{n}{3}$ and merge them together expending cn time. The recurrence tree for this sort is as follows:



Now, to sort each of the array of size $\frac{n}{3}$, we need to sort three subarrays of size $\frac{n}{9}$ and spend $\frac{cn}{3}$ time to merge them.



This process will stop after k levels, where $k = \log_3(n)$ and each level we spend cn

amount of effort. The total time complexity of the algorithm is easily seen to be $T(n) = O(n \log_3(n))$. The time complexity for the standard mergesort based on merging two lists at a time is $T'(n) = O(n \log_2(n))$. Since, the ratio of the time complexities $\frac{T(n)}{T'(n)} = \frac{\log(3)}{\log(2)}$, the two algorithms have the same asymptotic time complexity.

Problem 3 (1 + 1 pt) Devise a recursive algorithm for finding the sum of the first n **odd** positive integers.

- write down the pseudo-code of your algorithm.
- State the recurrence, draw the recursion tree, and find the complexity of your algorithm with respect to n in Θ notation.

Solution 3:

- The pseudocode is as follows:

```
if n==1:
    return n
else return Sumodd(n-1)+2*n-1
```

- The recurrence function is as follows:

$$T(n) = T(n-1) + (2n-1) \tag{2}$$

The recurrence tree for this algorithm is:

$$T(n) \to T(n-1) \to T(n-2) \to \ldots \to T(1) \tag{3}$$

The complexity of this algorithm is linear i.e. $\Theta(n)$.

Problem 4 (3 pt) Generate samples from the triangular pdf

$$p(x) = 2x \text{ for } 0 \leq x \leq 1 \tag{4}$$

using **two** different methods. Write Python functions to verify your algorithms by generating 10,000 samples from the above distribution and plotting their histogram. Describe your scheme in the space below, and attach printouts of both source code and the plotted histogram for full credit.

Solution 4: We describe two schemes below. The python codes are as attached. The first scheme is based on Accept-Reject method and the second one is inverse transform method. For the Accpet-Reject method, note that $p(x)$ is bounded by a multiple of 2 of the uniform distribution.

$$p(x) = 2x \leq 2.1 \text{ for } 0 \leq x \leq 1 \tag{5}$$

Therefore we can apply the Accept-Reject method with $M = 2$ and the proposal density $g(x) = 1,\ x \in [0, 1]$.

Algorithm 1 Accept-Reject Method

repeat $X \sim U(0, 1)$ and $U \sim U(0, 1)$
until $U \leq X$;
return X

The second algorithm is based on the inverse transform method. Given the density of $p(x)$ the cumulative distribution function (c.d.f.) is as follows:

$$P(t) = \int_0^t 2x dx = t^2 \tag{6}$$

The second scheme is as follows: Generate $U \sim U(0, 1)$ and then generate X by $P^{-1}(U) = \sqrt{U}$.