University of New Mexico
Department of Computer Science
Midterm Examination I
CS362: Data Structure and Algorithms

Name:_____
Email:_____

Instructions:

1. **Please silence your cell phones.**
2. Write your name and email address legibly in the space provided above.
3. Write your name legibly at the upper right hand corner on each page.
4. You are allowed to use a one page double-sided "cheating sheet" that you have brought to the exam and a calculator. Nothing else permitted.
5. You must not discuss the questions with anyone except the professor in charge.
6. Write your answers legibly.
7. There are **5 problems** in this exam.
8. For algorithmic problems, **do not give detailed pseudo-code**. Describe the key ideas and key steps of your solutions, correctness argument, and provide running time analysis. In the case, where you are using a well-known algorithm (e.g., merge sort or linear time selection) as a subroutine **without modification**, just cite the well-known algorithm and its running time. Do not repeat the steps of the well-known algorithms.
9. Don't spend too much time on any single problem. All questions are weighted equally. If you get stuck, move on to something else and get back later.
10. In the problems where you are asked to design an efficient algorithm, your points earned will depend on the efficiency of your algorithm. However, any solution is better than nothing.
11. Good luck, and enjoy the exam!

1. (10pt) Solve the following recurrence relation using your favorite method (e.g., recursion tree, master theorem, full history expansion, guess and verification, …)
Solve for $T(n)$ as a function of n and specify the asymptotic behavior of $T(n)$ by identifying its leading growth term with constant coefficient stripped off:

$$(a)\begin{cases} T(n)=1, \text{ for } n\le 1 \\ T(n)=T(n-1)+n \end{cases}$$

$$(b)\begin{cases} T(n)=1, \text{ for } n\le 1 \\ T(n)=T\left(\frac{n}{2}\right)+T\left(\frac{n}{4}\right)+n \end{cases}$$

Solution:

(a) $T(n)=T(n-1)+n=\big(T(n-2)+(n-1)\big)+n=T(n-2)+(n-1)+n=\cdots$

$=T\big(n-(n-1)\big)+2+3+\cdots+(n-1)+n=1+2+\cdots+n=\frac{n(n+1)}{2}=\Theta(n^2)$

(b) Use recursion tree analysis, we can conclude that:

$$T(n)=n+\frac{3}{4}n+\left(\frac{3}{4}\right)^2 n+\cdots=\frac{\frac{3}{4}\left(\frac{3}{4}\right)^{\infty}-1}{\frac{3}{4}-1}n=4n=\Theta(n)$$

2. (10pt) There are 25 horses, however, only 5 horses can be raced together at any moment. The goal is to determine the fastest, 2nd fastest, and 3rd fastest horses. Find the minimum number of races in which this can be done.

Solution:

Observe that since we are interested in the top 3 horses, for each race of 5 horses, we can eliminate the last 2 horses.

Thus, if we partition the horses into 5 groups of 5, and label these groups as A, B, C, D, and E. Let the results of the 5 races be, the subscripts indiciate the order of which the horses finish.

A_1, A_2, A_3, A_4, A_5.

B_1, B_2, B_3, B_4, B_5.

C_1, C_2, C_3, C_4, C_5.

D_1, D_2, D_3, D_4, D_5.

E_1, E_2, E_3, E_4, E_5.

Clearly, we can eliminate the horse that places 4th and 5th from each race.

Now suppose we race the 5 first place horses.

Without loss of generality, let's assume the order of which these 5 horses finish is: A_1, B_1, C_1, D_1, E_1.

Clearly, we can eliminate $C_2, C_3, D_1, D_2, D_3, E_1, E_2, E_3$.

Now we are left with $A_1, A_2, A_3, B_1, B_2, B_3$, and C_1.

It is clear that A_1 will be the fastest horse.

Since we are only interested in the first 3 places, B_3 can also be eliminated.

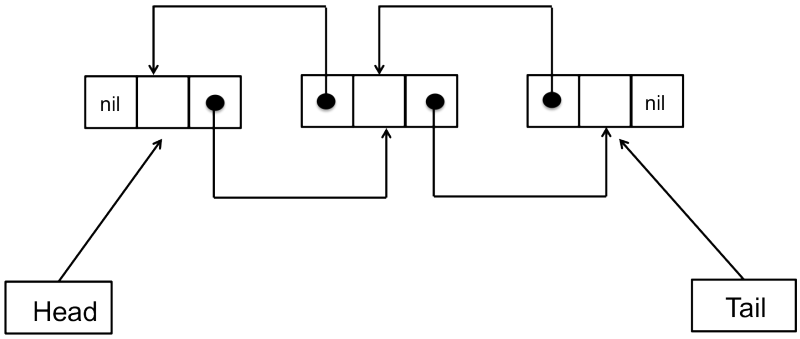
We can then run a race between A_2, A_3, B_1, B_2, and C_1 and the first two places will be the 2nd and 3rd fastest among the entire 25 horses.

Thus, with 7 races, we find the fastest, second fastest, and third fastest horses.

3. (10pt) Suppose you are given a doubly-connected linked list with *n* elements. Specifically, you should assume that the linked list maintains the head and tail of the list, and the pointers to the previous and next elements of any element. (See picture below for illustrations.)

Design an algorithm that can find the middle element of the list using the least number of steps, where the middle element refers to the $\left\lceil \frac{n}{2} \right\rceil$th element from the head.

Hint: The goal here is use the least number of steps, not the asymptotic behavior.



Solution:
Initialize two pointers, one pointed to the head and the other pointed to the tail.
In each iteration, the pointer that used to pointed to the head will move to its next, while the other pointer will move to its previous.
The algorithm terminates when the two pointers meet, which takes *n* steps.

4. (10pt) It is a well-known fact that the sorting of a general sequence of numbers must be done in $\Omega(n \log n)$ time. Answer the following questions:

(1) Suppose a professor has developed a general sorting algorithm with a running time of $O(n \log \sqrt{n})$ time. Does this contradicts with the well-know fact above?

(2) Is it possible to design a stack structure that can achieve push, pop, and retrieving the minimum element of the stack all in constant time? Explain why?

Solution:

(1) No, since $n \log \sqrt{n} = \frac{1}{2} n \log n = \Omega(n \log n)$

(2) No. For if such structure exists, then we will be able to perform sorting in linear time by inserting all the elements into the stack and keep popping the smallest element from the stack.

5. (10pt) Suppose that we are given a sequence of n values $x_1, x_2, \ldots, x_n$, and seek to quickly answer repeated queries of the form: given i and j, find the smallest value in $x_i, x_{i+1}, \ldots, x_j$.

Example: assume the numbers are: x_1=3, x_2=10, x_3=3, x_4=2, x_5=7, x_6=5, x_7=6, x_8=0, x_9=12.

If the query is i = 3, j = 7, then the answer should be x_4=2, which is the smallest among x_3=3, x_4=2, x_5=7, x_6=5, x_7=6.

If the query is i = 5, j = 9, then the answer should be x_8=0, which is the smallest among x_5=7, x_6=5, x_7=6, x_8=0, x_9=12.

Design a data structure that uses $O(n)$ space and answers each query in $O(log\ n)$ time. (Note that you will get partial credit for a solution even if it doesn't meet the space and time bounds.)

Solution:

Note that each number x_j can be viewed as a 2D point (x_j, j). Thus, the problem can be viewed as a 2D query problem, asking for the point with the maximum x-coordinate with in the y-coordinate range from i to j.

Our strategy is to use something like the range tree. Specifically, the main tree will be based on y-coordinate. For each sub-trees of the main tree, we can associate with its root an additional attribute which is minimum element of its sub-tree. This will take a query time of $O(\log n)$ time using $O(n)$ space.