BLG 381E ADVANCED DATA STRUCTURES
MIDTERM - NOVEMBER 21, 2012, 13:30-15:30 PM (2 hours)

1 (5 pt)	2 (15 pt)	3 (30 pt)	4 (30 pt)	5 (20 pt)	Total (100 pt)

On my honor, I declare that I neither give nor receive any unauthorized help on this exam.

Student Signature:___________________________

Write your name on each sheet.
Write your answers neatly (in English) in the space provided for them.
You must show all your work for credit.
Books and notes are closed.
Good Luck!

Q1[5 points]: Growth of Functions

For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions $\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$
Using this formal definition **show that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$ and determine positive constants c_1, c_2, and n_0.**

Following the definition in the question:

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

for all $n \geq n_0$. Dividing by n^2 yields

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2 .$$

By solving this inequality, a sample choice of solution would be: $c_1=1/14$, $c_2=1/2$, and $n_0=7$.

Q2[15 points]: Recurrences

Q2a) [3 pts]: List the three methods for solving recurrences

The substitution method
The recursion-tree method
The master method

Q2b) [6 pts]: Solve the following recurrence: *T(n)=T(2n/3)+1*

Using master theorem, a = 1, b = 3/2, f(n) = 1

$$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$$

Case 2 applies, since

$$f(n) = \Theta(n^{\log_b a}) = \Theta(1)$$

therefore

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

Q2c) [6 pts]: Solve the following recurrence: *T(n)=2T(n/2)+nlgn*

The recurrence falls into the gap between case 2 and case 3 of Master theorem.

From Exercise 4.4-2 of the textbook, if

$$f(n) = \Theta(n^{\log_b a} \lg^k n), \text{ where } k \geq 0$$

then the solution is

$$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$$

following this, the solution is

$T(n) = \Theta(n\lg^2 n)$

If you want to benefit from Master theorem:

Master theorem:
Let a≥1 and b>1 be constants, let f(n) be a function, and let T(n) be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n)$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then *T(n)* can be bounded asymptotically as follows.

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon>0$, then $T(n) = \theta(n^{\log_b a})$.
2. If $f(n) = \theta(n^{\log_b a})$, then $T(n) = \theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon>0$, and if $af(n/b) \leq cf(n)$ for some constant $c<1$ and all sufficiently large n, then $T(n) = \theta(f(n))$.

Q3[30 points]: Quicksort

Q3a) [10 pts]: Write down the QUICKSORT and PARTITION procedure's pseudocode.

```
QUICKSORT(A, p, r)
1  if p < r
2     then q ← PARTITION(A, p, r)
3          QUICKSORT(A, p, q − 1)
4          QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4       do if A[j] ≤ x
5             then i ← i + 1
6                  exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

Q3b) [5 pts]: Show that the worst-case running time of Quicksort is $\Theta(n^2)$.

$$
\begin{aligned}
T(n) &= T(n-1) + T(0) + \Theta(n) \\
&= T(n-1) + \Theta(n)
\end{aligned}
$$

Solving recurrence by iteration:

$$
\begin{aligned}
T(n) &= \Theta(n) + T(n-1) \\
&= \Theta(n) + \Theta(n-1) + \Theta(n-2) + \dots + \Theta(1) \\
&= \sum_{k=1}^{n} \Theta(k) \\
&= \Theta\left(\sum_{k=1}^{n} k\right) \\
&= \Theta(n^2)
\end{aligned}
$$

Q3c) [15 pts]: What is the expected running time of randomized quicksort? Prove it.

Hint 1: Use indicator random variable
Hint 2: Use substitution method
Hint 3:

$$\sum_{k=2}^{n-1} k lgk \le \frac{1}{2} n^2 lgn - \frac{1}{8} n^2$$

Let $T(n)$ = the random variable for the running time of randomized quicksort on an input of size n, assuming random numbers are independent.

For $k = 0, 1, \ldots, n-1$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

$E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \vdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)$$

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n))\right]$$

$$= \sum_{k=0}^{n-1} E[X_k (T(k) + T(n-k-1) + \Theta(n))]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)]$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n} \sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n)$$

$$= \frac{2}{n} \sum_{k=1}^{n-1} E[T(k)] + \Theta(n)$$

Summations have identical terms.

(The $k = 0, 1$ terms can be absorbed in the $\Theta(n)$.)

Prove: $E[T(n)] \le a n \lg n$ for constant $a > 0$.

- Choose a large enough so that $a n \lg n$ dominates $E[T(n)]$ for sufficiently small $n \ge 2$.

Use fact: $\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$

$$\begin{aligned} E[T(n)] &\le \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n) \\ &= \frac{2a}{n}\left(\frac{1}{2} n^2 \lg n - \frac{1}{8} n^2\right) + \Theta(n) \\ &= an \lg n - \left(\frac{an}{4} - \Theta(n)\right) \\ &\le an \lg n, \end{aligned}$$

if a is chosen large enough so that $an/4$ dominates the $\Theta(n)$.

Q4[30 points]: *A 3-ary heap is like a binary heap, but with the exception of the root, non-leaf nodes have 3 children instead of 2 children.*

Q4a) *[5 pts]: How would you compute the parent and children indices of a node in a 3-ary heap in the array representation?*

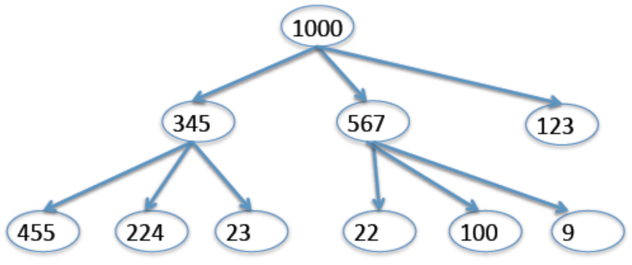
PARENT(i)=round(i/3)
LEFT[(i)=3*i - 1
MIDDLE(i)= 3*i
RIGHT[(i)= 3*i + 1

Q4b) [10 pts]: Give an efficient implementation of MAX-HEAPIFY3 for a 3-ary max-heap.

```
MAX-HEAPIFY3(A, i)
  1  l ← LEFT(i)
  2  m ← MIDDLE(i)
  3  r ← RIGHT(i) ;
  4  largest ← i
  5  if l ≤ heap-size[A] and A[l] > A[i]
  6     then largest ← l
  7     else largest ← i
  8  if m ≤ heap-size[A] and A[m] > A[largest]
  9     then largest ← m
  10 if r ≤ heap-size[A] and A[r] > A[largest]
  11    then largest ← r
  12 if largest ≠ i
  13    then exchange A[i] ↔ A[largest]
  14         MAX-HEAPIFY(A, largest)
```

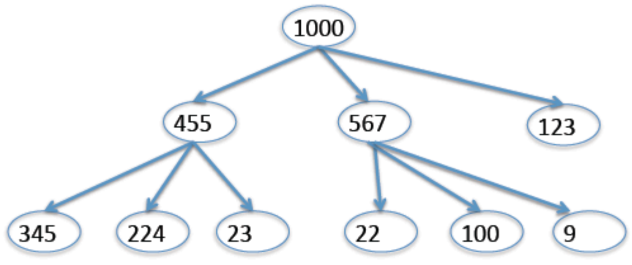
Q4c) [5 pts]: Show how MAX-HEAPIFY3(A,2) works (using array representation) for input array A on input 345 which is at index 2:

A = [1000, **345**, 567, 123, 455, 224, 23, 22, 100, 9]



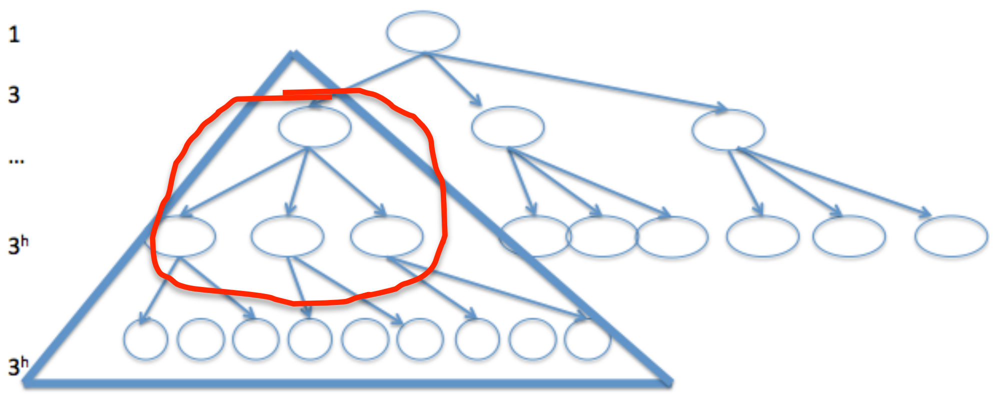
```
MAXHEAPIFY3(A,2)
   l=5; m=6; r=7;              //lines 1-3
   largest = i=2;              //line 4
   largest = l = 5             //lines 5-10
   exchange(A(2),A(5))         //line 13
   MAXHEAPIFY3(A,5)  //line14

MAXHEAPIFY3(A,5)
   l=14; m=15; r=16;           //lines 1-3
   largest = i=2;              //line 4
   //since   l, m and r are >heap-size(A)
   //we do not enter the if's in lines 5-10
   //since largest == i, we do not enter
   // the if in line 12 either.
```



Q4d) [5 pts]: Analyze MAX-HEAPIFY3's running time in terms of n (no of items in the heap).

Lines 1-13 are constant time operations and are executed only once per call, so they have **Θ**(1) complexity.
We need to find the number of nodes involved in line 14 to find the recursion.



At the worst case, the leaf level is 1/3 full.
The total number of nodes in the heap is n, height of the heap is h:

$$n = 1 + 3 + 3^2 + \cdots + 3^h + 3^h = \frac{3^{h+1} - 1}{2} + 3^h = \frac{5 * 3^h - 1}{2}$$

Therefore:

$$3^h = \frac{2n + 1}{5}$$

Number of involved nodes in this worst case is:

$$\frac{n - 3^h - 1}{3} + 3^h = \frac{n - 1}{3} + \frac{2}{3}3^h = \frac{n - 1}{3} + \frac{2}{3}\frac{2n + 1}{5} = \frac{9n - 3}{15} = \frac{3n - 1}{5} \leq \frac{3}{5}n$$

$$T(n) = T\left(\frac{3}{5}n\right) + \theta(1)$$

$T(n) = \Theta(\log_2 n)$ //Master Theorem Case 2

Q4e) *[5 pts]: When (for which range of array size n) is it more beneficial to use 2-ary heap and when is it more beneficial to use the 3-ary heap for the MAX-HEAPIFY operation?*
In order to answer this question, we need to find the running times more accurately, in the exam it is OK if you could not get the constants below right.

$T(n) = T\left(\frac{3}{5}n\right) + 14 = 14 * log_{\frac{5}{3}}n = \frac{14}{log_{10}\left(\frac{5}{3}\right)}log_{10}n$ MAXHEAPIFY3's running time

$Q(n) = T\left(\frac{2}{3}n\right) + 10 = 10 * log_{\frac{3}{2}}n = \frac{10}{log_{10}\left(\frac{3}{2}\right)}log_{10}n$ MAXHEAPIFY2's running time

$\frac{14}{log_{10}\left(\frac{5}{3}\right)} = 63,\ \frac{10}{log_{10}\left(\frac{3}{2}\right)} = 56$

$T(n) = 63log_{10}n,\ Q(n) = 56log_{10}n$

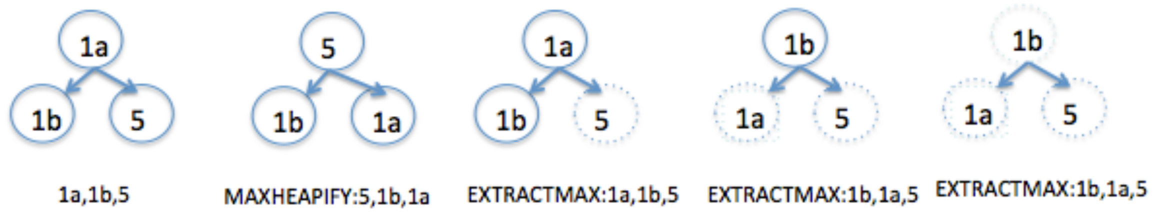
Therefore using of MAXHEAPIFY2 is more beneficial.

Q5[20 points]

Q5a) *[2 pts]: What is stable sorting?*
Stable sorting is a sorting algorithm where if for two records A and B the key values are the same, if recordA appears before recordB in the original list, they recordA appears before recordB in the sorted list. The order of records with the same key values with respect to each other are preserved during sorting. Counting sort is a stable sorting algorithm.

Q5b) *[10 pts]: Is heap sort a stable sorting algorithm? Why or why not?*

Heapsort is not a stable sorting algorithm.
Counter example:
original list: 1a, 1b, 5 (1a and 1b both have the key value 1)
sorted list: 1b, 1a, 5



Q5c) *[8 pts]: Sort the following 10 numbers using Radix sort. Show all the steps of your work.*
9, 100, 345, 123, 455, 224, 23, 22, 567, 1000

1st	2nd	3rd	4th
0	0	0	9
0	1	0	0
0	3	4	5
0	1	2	3
0	4	5	5
0	2	2	4
0	0	2	3
0	0	2	2
0	5	6	7
1	0	0	0

Sort acc to 4th dig-->

1st	2nd	3rd	4th
0	1	0	0
1	0	0	0
0	0	2	2
0	1	2	3
0	0	2	3
0	2	2	4
0	3	4	5
0	4	5	5
0	5	6	7
0	0	0	9

Sort acc to 3rd dig-->

1st	2nd	3rd	4th
0	1	0	0
1	0	0	0
0	0	0	9
0	0	2	2
0	1	2	3
0	0	2	3
0	2	2	4
0	3	4	5
0	4	5	5
0	5	6	7

Sort acc to 2nd dig-->

1st	2nd	3rd	4th
1	0	0	0
0	0	0	9
0	0	2	2
0	0	2	3
0	1	0	0
0	1	2	3
0	2	2	4
0	3	4	5
0	4	5	5
0	5	6	7

Sort acc to 1st dig-->

1st	2nd	3rd	4th
0	0	0	9
0	0	2	2
0	0	2	3
0	1	0	0
0	1	2	3
0	2	2	4
0	3	4	5
0	4	5	5
0	5	6	7
1	0	0	0

Sorted