BLG 335E ANALYSIS OF ALGORITHMS I
MIDTERM - NOVEMBER 13, 2013, 13:30-15:30 PM (2 hours)

1 (10 pt)	2 (15 pt)	2 (15 pt)	3 (15 pt)	4 (30 pt)	5 (15 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[10 points]: For a given function *g(n)*, we denote by *O(g(n))* the set of functions

$O(g(n)) = \{f(n)$: there exist positive constants c, and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$

Using this formal definition **show that $\frac{1}{2}n^2 - 3n = O(n^2)$ and determine positive constants *c* and n_0.**

1)

$$\frac{1}{2}n^2 - 3n = O(n^2)$$

$$0 \le \frac{1}{2}n^2 - 3n \le cn^2$$

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

$$0 \le \frac{1}{2} - \frac{3}{n} \le c$$

The left hand inequality can be made to hold for any value of $n \geq 6$. Thus, by choosing $c = \frac{1}{2}$, $n_0 = 6$, we can verify that $\frac{1}{2}n^2 - 3n = O(n^2)$

HINT: If you want to benefit from **MASTER THEOREM** for Q2 & Q3:

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 ϵ>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 ϵ>0, and if $af(n/b) \leq cf(n)$ for some constant *c<1* and all sufficiently large n, then $T(n) = \theta(f(n))$.

Q2) [15 pts]: What is the **worst case** running time of **INSERTION SORT**? Give the algorithm and prove the bound you have given.

2) Worst case running time of Insertion Sort: $\Theta(n^2)$

cost	INSERTION-SORT (A)
c_1	for $j \leftarrow 2$ to length[A]
c_2	do key $\leftarrow A[j]$
0	▷ Insert $A[j]$ into the sorted sequence $A[1 \ldots j-1]$.
c_4	$i \leftarrow j-1$
c_5	while $i > 0$ and $A[i] >$ key
c_6	do $A[i+1] \leftarrow A[i]$
c_7	$i \leftarrow i-1$
c_8	$A[i+1] \leftarrow$ key

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

If the array is reverse sorted $\Rightarrow$ worst-case

$\Rightarrow$ We must compare each element $A[j]$ with each element in the entire sorted subarray $A[1 \ldots j-1]$, and so $t_j = j$.

Noting that: $\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$, $\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$

$\Rightarrow$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$= \underbrace{\left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)}_{a} n^2 + \underbrace{\left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)}_{b} n - \underbrace{(c_2 + c_4 + c_5 + c_8)}_{c}$$

$\Rightarrow a n^2 + bn - c$

$\Theta(n^2)$

Q3) [15 pts]: What is the **best case** running time of **MERGESORT**? Give the algorithm and prove the bound you have given.

3)
```
MERGE-SORT (A, p, r)
  if p < r
    then q ← ⌊(p+r)/2⌋
         MERGE-SORT (A, p, q)
         MERGE-SORT (A, q+1, r)
         MERGE (A, p, q, r)
```

$$T(n) = \begin{cases} \Theta(1) & \text{if } n=1 \\ 2T(n/2)+\Theta(n) & \text{if } n>1 \end{cases}$$

Master Method

$a=2$, $b=2$, $f(n)=\Theta(n)$

Case 2

$\Rightarrow f(n) = \Theta(n)$

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

```
MERGE (A, p, q, r)
  n1 ← q - p + 1
  n2 ← r - q
  create arrays L[1..n1+1], R[1..n2+1]
  for i ← 1 to n1
     do L[i] ← A[p+i-1]
  for j ← 1 to n2
     do R[j] ← A[q+j]
  L[n1+1] ← ∞
  R[n2+1] ← ∞
  i ← 1
  j ← 1
  for k ← p to r
     do if L[i] ≤ R[j]
          then A[k] ← L[i]
               i ← i+1
          else A[k] ← R[j]
               j ← j+1
```

Q4) [15points] Consider the following algorithm to compute the minimum of an array A. Prove that min contains the minimum of the array in line 5. Hint: Use a **loop invariant** and induction.

```
MINIMUM(A[1..n])
1 min ← A[1]
2 for i ← 2 to n
3     do if min > A[i]
4             then min ← A[i]
5 return min
```

Q4)ANSWER [NOTE: Do not give an example, you need to prove that the algorithm works for all possible inputs. Do not show the time complexity, we do not ask for it.]

Loop invariant: At the beginning of the for loop on line 2, min contains the minimum of A[1..i-1]

Basis:
i=2, because of the assignment in line 1, min = min(A[1])=A[1)

Induction:
Assume that
At the beginning of the for loop on line 2, min contains the minimum of A[1..i-1]
Prove that for i+1, at the beginning of the for loop on line 2, min contains the minimum of A[1..i]

Case 1:
If if min <=A[i]: minimum of A[1..i] is not A[i] and is in A[1..i-1]. By the inductive hypothesis, the minimum of A[1..i-1] is already in min.

Case 2:
If if min > A[i]: min contains minimum of A[1..i-1] by the inductive hypothesis and since A[i] is smaller than min, A[i] is the minimum A[1..i]. By the assignment on line 5, min is assigned to A[i] and therefore it contains the minimum of A[1..i].

Termination:
When i=n+1, after the last iteration of the for loop, min contains the minimum of A[1..n], therefore min contains the minimum of the whole array A.

Q5) [30 points]

Q5a) [15 pts]: You are given the following key:satelliteData pairs. Write down the **pseudocode** of a **linear time sorting algorithm** that sorts this data in increasing order of the key values. Your algorithm must use **COUNTING SORT** and some additional code.
[9:X], [100:A], [99:B], [8:Y], [10:C], [100:Z], [98:T], [10:B], [8:E], [99:U]

Q5a) ANSWER:[NOTE: Radix and Bucket Sort using Counting Sort also got full mark]
There are two groups of key values:
In [8:10] interval : [9:X], [8:Y], [10:C], [10:B], [8:E],
In [98:100] interval: [100:A], [99:B], [100:Z], [98:T], [99:U]
Our algorithm is as follows:
Get the array elements within those two intervals,
Sort each one using **COUNTING SORT**
Then combine these two arrays by appending them

GROUPCOUNTINGSORT(A[1..n],n,m1, M1, m2, M2)

```
//Assume that array contains numbers in [m1:M1] or [m2:M2], m1<M1<m2<M2
//m1, M1: min and max of the first group
//m2, M2: min and max of the second group
//Allocate space for two ranges of numbers
A1 ← array [1..n]
A2 ← array [1..n]
n1=0; n2=0;
for i=1..n
        if A[i] >= m1 AND  A[i] <= M1
                then A1[n1] ← A[i] – m1
                        n1 ←n1 + 1
        else if A[i] >= m2 AND  A[i] <= M2
                then A2[n2] ← A[i] –m2
                        n2 ←n2 + 1
        else Error('A[i] outside boundaries')
COUNTINGSORT(A1,A,(M1-m1))
for i=1..n1
        A1[i] = A[i]+m1
COUNTINGSORT(A2,A,(M2-m2))
for i=1..n2
        A2[i] = A[i]+m2
B ← array [1..n]
for i=1..n1
        B[i] = A1[i]
for i=1..n2
        B[i+n1] = A2[i]
//B contains sorted A
```

Execution trace of the algorithm
(all the operations are on the keys, A[i] returns key of A[i], assignment assigns the key and the corresponding value pair.
Initial call: m1=8, M1=10, m2=98, M2=100, n=10
A1[1…10], A2[1…10]

A1 = [[1:X], [0:Y], [2:C], [2:B], [0:E],…],
A2 = [[2:A], [1:B], [2:Z], [0:T], [1:U],…]
n1=5, n2=5

A1 = [[8:Y], [8:E], [9:X], [10:C], [10:B],…],
A2 = [[98:T], [99:B], [99:U], [100:A], [100:Z],…]

B=[[8:Y], [8:E], [9:X], [10:C], [10:B],[98:T], [99:B], [99:U], [100:A], [100:Z]]

Note: There is really no need to call the MERGE of MERGESORT, because the array is already sorted.

Q5b) [7 pts]: Sort the following array using **RADIX SORT**. Show all the steps of your work.

22, 9, 100, 345, 329, 23, 110

Q5b) ANSWER

	Use Counting Sort (or any other stable sort) acc to least significant digit	Use Counting Sort (or any other stable sort) acc to second least significant digit	Use Counting Sort (or any other stable sort) acc to the most significant digit
022	10**0**	1**0**0	**0**09
009	11**0**	0**0**9	**0**22
100	02**2**	1**1**0	**0**23
345	02**3**	0**2**2	**1**00
329	34**5**	0**2**3	**1**10
023	00**9**	3**2**9	**3**29
110	32**9**	3**4**5	**3**45

Q5c) [8 pts]: Is **QUICKSORT** a **stable** sorting algorithm? Why or why not?

Q5c) ANSWER

Stable sorting algorithms output inputs with the same key values in the same order they appear in the input with respect to each other. QUICKSORT is NOT a stable sorting algorihm. Exhange on line 6 of PARTITION may cause the array portion larger than pivot to be unstable. Exchange on line 7 does not cause unstable sorting.

```
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
```

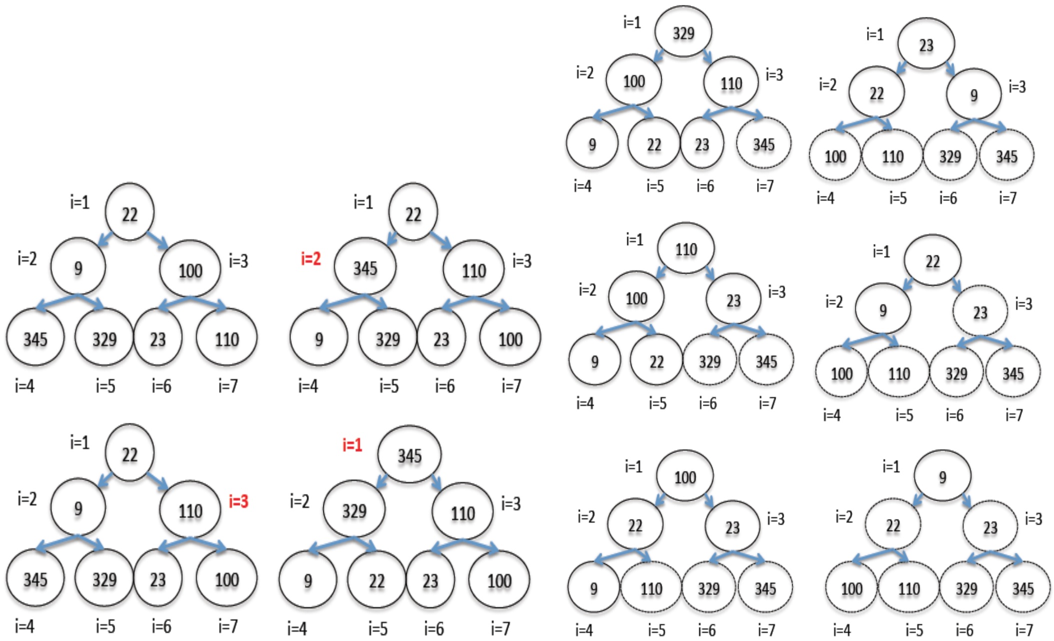
Counter Example that shows QUICKSORT is not stable:
[_,_] denotes a key value pair
Original array:
A=[[2,Q],[2,R],[1,S]]

PARTITION(A,1,3)
A=[[2,Q],[2,R],[1,S]]
x=1,
i=0,j=1
A =[[2,Q],[2,R],[1,S]] //no change
i=0, No change for j=2
A =[[2,Q],[2,R],[1,S]],
A =[[1,S],[**2,R],[2,Q**]], //exchange A[2],A[4]
[2,Q],[2,R] at the do not input appear in the same order. The call to QUICKSORT([2,Q],[2,R]) does not change their order, so they remain different ordered then the initial array. Hence QUICJSORT is unstable.

Q6) [15points] Sort the following array using **HEAPSORT** and the array representation. Show all the steps of your work.

22, 9, 100, 345, 329, 23, 110

[NOTE: Solution with MIN-HEAPIFY is also accepted.]



	1	**2**	**3**	**4**	**5**	**6**	**7**
BUILD-MAX_HEAP	22	9	100	345	329	23	110
MAX_HEAPIFY(A,3)	22	9	**110**	345	329	23	**100**
MAX_HEAPIFY(A,2)	22	**345**	110	**9**	329	23	100
MAX_HEAPIFY(A,1)	**345**	**329**	110	9	**22**	23	100
HEAPSORT for i ← length[A] downto 2 do							
exchange A[1] ↔ A[i]; heap-size[A] =6; MAX-HEAPIFY(A, 1)	329	100	110	9	22	23	**345**
exchange A[1] ↔ A[i]; heap-size[A] =5; MAX-HEAPIFY(A, 1)	110	100	23	9	22	**329**	**345**
exchange A[1] ↔ A[i]; heap-size[A] =4; MAX-HEAPIFY(A, 1)	100	22	23	9	**110**	**329**	**345**
exchange A[1] ↔ A[i]; heap-size[A] =3; MAX-HEAPIFY(A, 1)	23	22	9	**100**	**110**	**329**	**345**
exchange A[1] ↔ A[i]; heap-size[A] =2; MAX-HEAPIFY(A, 1)	22	9	**23**	**100**	**110**	**329**	**345**
exchange A[1] ↔ A[i]; heap-size[A] =1; MAX-HEAPIFY(A, 1)	9	**22**	**23**	**100**	**110**	**329**	**345**