Name and Student ID: **SOLUTIONS**

Advanced Data Structures, BLG381E Midterm Exam, Nov 20, 2009

1 (20pt)	2 (10 pt)	3 (20 pt)	4 (15 pt)	5 (15 pt)	6 (20 pt)	Total (100 pt)

Name: **SOLUTIONS**
Number:
Signature:
Duration: 120 minutes.
Write your answers neatly in the space provided for them.
Write your name on each sheet.
Books and notes are closed. Good Luck!

QUESTIONS

Q1) [20 points] Quicksort
Q1a) [10 points] Show how quicksort sorts the array 5 9 7 4 0 2 8 8. Always choose the last key in an array (or subarray) to be the pivot. Draw the array once for each swap.

5|9|7 4 0 2 8|8
5 7|9|4 0 2 8|8
5 7 4|9|0 2 8|8
5 7 4 0|9|2 8|8
5 7 4 0 2|9|8|8
5 7 4 0 2 8|9|8
5 7 4 0 2 8|8|9

$\boxed{5\ 7\ 4\ 0\ 2}\boxed{8}\boxed{8\ 9}$
$\boxed{0\ 7\ 4\ 5\ 2}\boxed{8}\boxed{8\ 9}$

$\boxed{0\ 2\ 4\ 5\ 7}\boxed{8}\boxed{8\ 9}$

Q1b [10 points] Consider a list-based quicksort that computes the average (mean) value i of all the keys in the list then chooses the pivot to be $i + 1$. As usual, we partition the list into three separate lists: keys less than $i + 1$, keys equal to $i + 1$ (of which there might be none), and keys greater than $i + 1$. We sort the first and last lists recursively, and then we concatenate the three lists together. What is wrong with this algorithm?

All of the keys might go into the first list (e.g., if all the keys are equal). Then, this quicksort algorithm will call itself recursively with the same list it started with, and it will repeatedly call itself forever, never terminating.

Name and Student ID: **SOLUTIONS**

Q2) [10 points] Radix Sort

Suppose you are given the task of sorting one thousand 32-bit keys. You have decided to use radix sort for this problem and need to decide on which representation you should use: binary (1 bit per digit), octal (3 bits per digit) or hexadecimal (4 bits per digit). Which one of these representations results in the best sorting time? Assume that you are provided with a counting sort procedure with exact time complexity of 5n + 4k where n is the number of items sorted and k is the maximum element. Show your work in detail.

bits/radix digit	k	# digits	d(5n+4k)=d(5000+4k)
1	$2^1-1=1$	32/1=32	32(5000+4x1)=160,128
3	$2^3-1=7$	$\lceil 32/3 \rceil=11$	11(5000+4x7)=55,308
4	$2^4-1=15$	32/4=8	8(5000+4x15)=40,480

So, of the given choices, the hexadecimal representation (4 bits per radix digit) results in the best sorting time.

Name and Student ID: **SOLUTIONS**

Q3) [20 points] Growth of Functions and Recurrences

Q3a) [5 points] Let f(n) and g(n) be non-negative and increasing functions of n. Prove or disprove the following statement. Show details of your work.

If f(n)=O(g(n)) then f(n)+g(n) = O(g(n))

We will PROVE the statement.
If f(n)=O(g(n)) then there is a constant c such that f(n)<=cg(n) for all n>n0
f(n)+g(n)<=cg(n)+g(n)=(c+1)g(n)=d g(n)
Since f(n)+g(n)<= d g(n) where d is a constant and d=c+1, for all n>n0
f(n)+g(n) = O(g(n))

Consider the following recursive algorithm to compute the minimum of an array.
Q3b) [8 points] Write the recurrence that describes the time complexity of the algorithm MINIMUM(A,n,1) for an array of size n. You can assume that atomic operations such as assignment and comparison take unit amount of time.

```
MINIMUM(A,p,r)
1       k = length[A]
2       if k==1
3               then return A[1]
4
5       q = floor(k/2)
6       minx = MINIMUM(A,p,q)
7       miny = MINIMUM(A,q+1,r)
8       if (minx < miny)
9               then min = minx
10              else min = miny
11      return min
```

T(n)=2T(n/2)+c

where c is the amount of time it takes to perform operations on lines 1, 2, 5, 8-11. We also assume that n is a power of 2, hence we do not worry about ceilings or floors.

Q3c) [7 points] Solve your recurrence.

$T(n)=2T(n/2)+c$
$2T(n/2)=2^2T(n/4)+2c$
$2^2T(n/4)=2^3T(n/8)+ 2^2c$
….
$2^{lgn}\, T(2)= 2^{1+lgn}\, T(2)+ 2^{lgn}\, c$

$T(n)=d+c\ \sum_{i=0\ldots lgn} 2^i$
$=d+c(2^{lgn} -1)=d-c+cn= \Theta(n)$

Alternatively,
$T(n)=\Theta(n)$

using Master Method, 1st case,

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

$f(n)=O\left(n^{\log_b a-\varepsilon}\right) \Rightarrow T(n)=\Theta\left(n^{\log_b a}\right)$

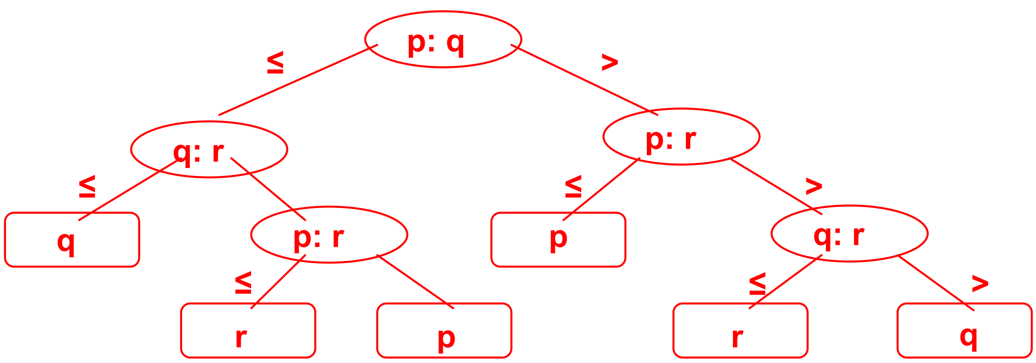
Name and Student ID: **SOLUTIONS**

Q4a) [10 points] Algorithm Correctness
Consider the following algorithm to compute the minimum of an array A.
Prove that min contains the minimum of the array in line 6. **Hint:** Use a loop invariant and induction.

```
MINIMUM(A)
1 k = length[A]
2 min ← A[1]
3 for i ← 2 to k
4     do if min > A[i]
5            then min ← A[i]
6 return min
```

Loop invariant: min contains the minimum of A[1..i]
INITIALIZATION:
At the beginning of the loop, loop invariant is true, because i=1 and min = A[1]
MAINTENANCE:
At any iteration i>1, let us assume that min already contains the minimum of A[1..i-1]. At iteration i, there are two possibilities:
If min needs to be updated, i.e. A[i] is smallest of A[1..i], min is assigned to A[i] on line 5,
If min does not need to be updated, i.e. A[i] is not the smallest of A[1..i], min does not change.
Therefore, at the end iteration i, min contains the minimum of A[1..i]
TERMINATION: When i=n, min contains the minimum of A[1..n], hence the minimum of the whole array.

Q4b) [5 points] Decision Tree
Draw a decision tree that finds the median of three numbers *p*, *q*, and *r*.



Q5) [15 points] Hash Table

Insert numbers B = {1, 2, 8, 15} into an empty hash table of size 7.

Q5a) [5 points] Use open addressing and linear probing. What is the number of collisions?

Q5b)[10 points] Use open addressing and double hashing. What is the number of collisions?

m=7, We will let C show the number of collisions during an insertion.

Two ordinary hash functions:

h’(k)=k mod 23 h”(k)=k mod 11

(You could have chosen some other prime number instead of 23 and 29)

Q5a) Linear probing will use the hash function:

h(k,i) = (h’(k) +i) mod m

h(k,i)=((k mod 23)+i) mod 7

As shown below, the total number of collisions is 2+3=5.

Q5b) Double hashing will use the hash function:

h(k,i) = (h’(k) +i*h”(k)) mod m

h(k,i)=((k mod 23)+i*(k mod 11)) mod 7

As shown below, the total number of collisions is 2+1=3.

	Insert(1) h(1,0)=1 C=0	Insert(2) h(2,0)=2 C=0	Insert(8) h(8,0)=1* h(8,1)=2* h(8,2)=3 C=2	Insert(15) h(15,0)=1* h(15,1)=2* h(15,2)=3* h(15,3)=4 C=3
0				
1	1	1	1	1
2		2	2	2
3			8	8
4				15
5				
6				

LINEAR PROBING

	Insert(1) h(1,0)=1 C=0	Insert(2) h(2,0)=2 C=0	Insert(8) h(8,0)=1* h(8,1)=2* h(8,2)=3 C=2	Insert(15) h(15,0)=1* h(15,1)=5 C=1
0				
1	1	1	1	1
2		2	2	2
3			8	8
4				
5				15
6				

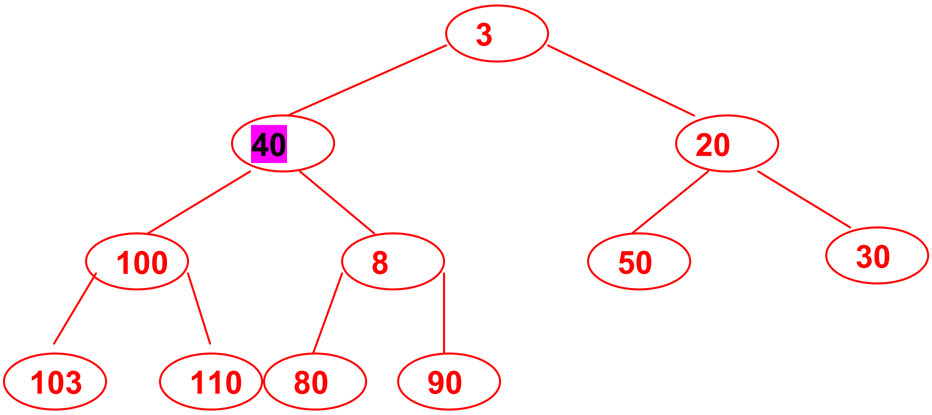
LINEAR PROBING

Name and Student ID: **SOLUTIONS**

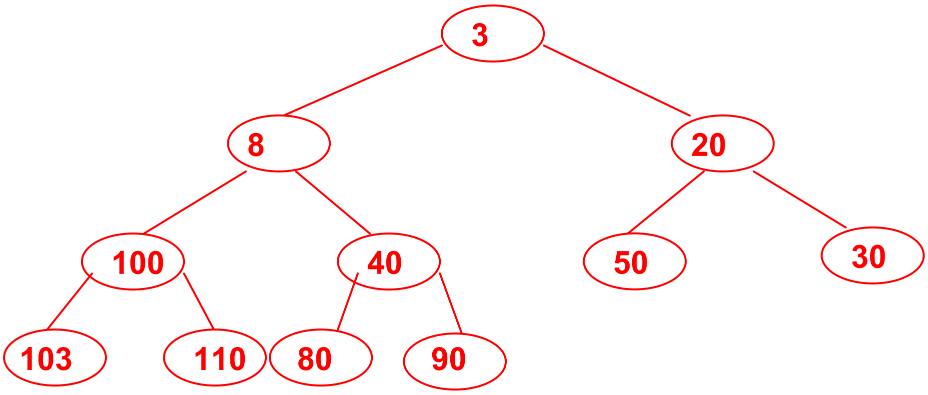
Q6) [20 points] Heapsort
Consider the following array representation of a min-heap.

1	2	3	4	5	6	7	8	9	10	11
3	40	20	100	8	50	30	103	110	80	90

Q6a)[5 points] Which element violates the min-heap property?
Q6b)[5 points] How could you modify your heap so that it satisfies the min-heap property? Provide details of your steps.
Q6c)[10 points] Use heapsort to sort the array in increasing order.



Q6a) Since 40 is larger than 8, it violates the min-heap property.
Q6b) We need to use MIN-HEAPIFY(A,2)
Min[A[2],A[4],A[5]) = 8, index of the min=5, therefore swap(A[2],A[5]),
40 is replaced by 8.
Hence, the correct min-heap becomes:



Name and Student ID: **SOLUTIONS**

Extra sheet

Q6b) CONTINUED MIN-HEAPIFY is a modified version of MAX-HEAPIFY

```
MIN-HEAPIFY(A, i)
    l ← LEFT(i)
    r ← RIGHT(i)
    if l ≤ heap-size[A] and A[l] < A[i]
       then smallest ← l
       else smallest ← i
    if r ≤ heap-size[A] and A[r] < A[smallest]
       then smallest ← r
    if smallest ≠ i
       then exchange A[i] ↔ A[smallest]
  MIN-HEAPIFY(A, smallest)
```

Q6c) CONTINUED (SORTED PART IS SHOWN IN BOLD)

	1	2	3	4	5	6	7	8	9	10	11
	3	40	20	100	8	50	30	103	110	80	90
EXTRACT-MIN	90	40	20	100	8	50	30	103	110	80	**3**
MIN-HEAPIFY(A,1)	8	40	20	100	80	50	30	103	110	90	**3**
EXTRACT-MIN	90	40	20	100	80	50	30	103	110	**8**	**3**
MIN-HEAPIFY(A,1)	20	40	30	100	80	50	90	103	110	**8**	**3**
EXTRACT-MIN	**110**	40	30	100	80	50	90	103	**20**	**8**	**3**
MIN-HEAPIFY(A,1)	30	40	50	100	80	110	90	103	**20**	**8**	**3**
EXTRACT-MIN	**103**	40	50	100	80	110	90	**30**	**20**	**8**	**3**
MIN-HEAPIFY(A,1)	40	80	50	100	103	110	90	**30**	**20**	**8**	**3**
EXTRACT-MIN	90	80	50	100	103	110	**40**	**30**	**20**	**8**	**3**
MIN-HEAPIFY(A,1)	50	80	90	100	103	110	**40**	**30**	**20**	**8**	**3**
EXTRACT-MIN	110	80	90	100	103	**50**	**40**	**30**	**20**	**8**	**3**
MIN-HEAPIFY(A,1)	80	100	90	110	103	**50**	**40**	**30**	**20**	**8**	**3**
EXTRACT-MIN	103	100	90	110	**80**	**50**	**40**	**30**	**20**	**8**	**3**
MIN-HEAPIFY(A,1)	90	100	103	110	**80**	**50**	**40**	**30**	**20**	**8**	**3**
EXTRACT-MIN	110	100	103	**90**	**80**	**50**	**40**	**30**	**20**	**8**	**3**
MIN-HEAPIFY(A,1)	100	110	103	**90**	**80**	**50**	**40**	**30**	**20**	**8**	**3**
EXTRACT-MIN	103	110	**100**	**90**	**80**	**50**	**40**	**30**	**20**	**8**	**3**
MIN-HEAPIFY(A,1)	103	110	**100**	**90**	**80**	**50**	**40**	**30**	**20**	**8**	**3**
EXTRACT-MIN	110	**103**	**100**	**90**	**80**	**50**	**40**	**30**	**20**	**8**	**3**
MIN-HEAPIFY(A,1)	110	**103**	**100**	**90**	**80**	**50**	**40**	**30**	**20**	**8**	**3**

```
HEAPSORT(A)
  BUILD-MIN-HEAP(A)
  for i ← length[A] downto 2 do
        exchange A[1] ↔ A[i]
        heap-size[A] ← heap-size[A] – 1
        MIN-HEAPIFY(A, 1)
```

Name and Student ID: **SOLUTIONS**

Extra sheet