ECE368 Exam 2
Spring 2009

Tuesday, April 14, 2009
6:30-7:30pm
MSEE B012

READ THIS BEFORE YOU BEGIN

This is a *closed book* exam. Calculators are not needed but they are allowed. Since time allotted for this exam is *exactly* 60 minutes, you should work as quickly and efficiently as possible. *Note the allocation of points and do not spend too much time on any problem*.

Always show as much of your work as practical–partial credit is largely a function of the *clarity and quality* of the work shown. Be *concise*. It is fine to use the *blank page opposite each question* for your work.

This exam consists of 11 pages (including this cover sheet and a grading summary sheet at the end); please check to make sure that *all* of these pages are present *before* you begin. Credit will not be awarded for pages that are missing – it is *your responsibility* to make sure that you have a complete copy of the exam.

IMPORTANT: Write your login at the TOP of EACH page. Also, be sure to *read and sign* the *Academic Honesty Statement* that follows:

"In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exercise and will be subject to possible disciplinary action."

Printed Name:

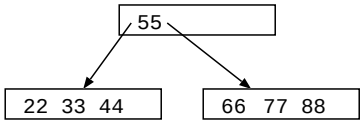
login:

Signature:

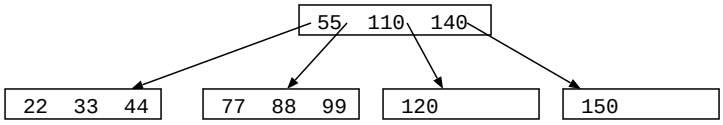
DO NOT BEGIN UNTIL INSTRUCTED TO DO SO ...

1. (20 points) For this problem, assume B-trees of order 4. When you split a node due to overflow, promote the median *before* the new key insertion to the parent node. (Consider a sorted array $A[lb..ub]$, the median is $A[\lfloor lb+ub/2 \rfloor]$.) When you delete a key from a non-leaf node, replace it with its immediate in-order predecessor.

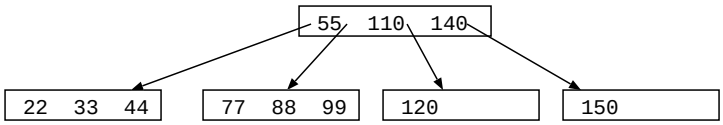
(a) (5 points) Given the following B-tree, insert key 99.



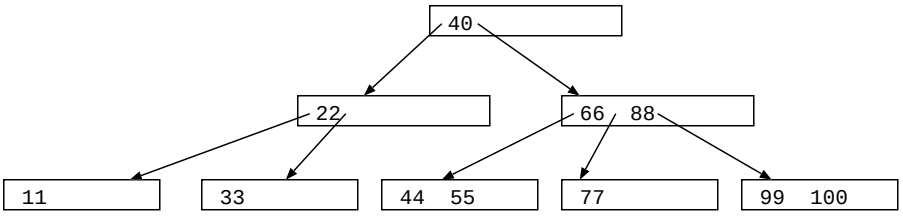
(b) (5 points) Given the following B-tree, insert key 66.



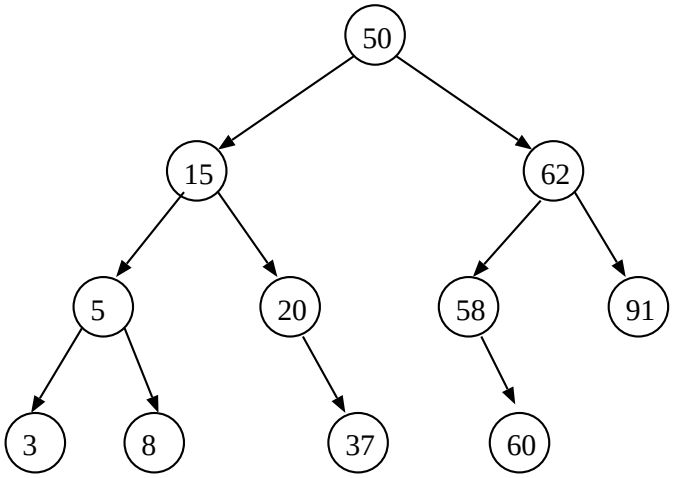
(c) (5 points) Given the following B-tree, delete key 150.



(d) (5 points) Given the following B-tree, delete key 40.



2. (20 points) The following diagram shows a *height-balanced* binary search tree (AVL-tree). Performs the following operations: Insert 4, insert 30, delete 91, insert 92, and delete 5. Draw the resulting height-balanced binary search tree after each insertion or deletion (*not after each rotation*). Clearly state the rotation operations that you have used.



3. (20 points) Consider the following binary search routine, where a[0..$n-1$] is a sorted array:

```
Binary_Search(a[0..n-1], search_key)
  lb ← 0
  ub ← n-1
  while (lb ≤ ub) {
    mid ← (lb+ub)/2
    if (search_key == a[mid])
      return mid
    if (search_key < a[mid])
      ub ← mid - 1
    else
      lb ← mid + 1
  }
  return -1
```

Note that (lb+ub)/2 is equivalent to $\lfloor$(lb+ub)/2$\rfloor$. The maximum number of data comparisons is approximately $2\log n$. In an attempt to reduce the maximum number of data comparisons to $\log n$, Professor Koh developed the following routine:

```
Fast_Binary_Search(a[0..n-1], search_key)
  lb ← 0
  ub ← n-1
  while (lb < ub) {
    mid ← (lb+ub)/2
    if (search_key < a[mid])
      ub ← mid − 1
    else
      lb ← mid
  }
  if (a[lb] == search_key)
    return lb
  else
    return -1
```

(a) (10 points) However, the Fast_Binary_Search routine does not work properly; it sometimes terminates with the right answer, but it sometimes iterates forever. Identify the main reason for the routine to iterate forever.

(b) (10 points) Modify the `Fast_Binary_Search` routine such that it indeed reduces the maximum number of data comparisons to $\log n$, and it terminates properly.

4. (20 points) In the heapsort algorithm introduced in class, we use a max-heap to sort an array `r[0..n-1]` in ascending order. The pseudo-code is given below:

```
Downward_heapify(A[], i, n):
  temp_r ← A[i-1]
  while i ≤ n/2 {
    j ← 2 × i
    if j < n and A[j-1] < A[j] {
      j ← j+1
    }
    if temp_r ≥ A[j-1] {
      break
    } else {
      A[i-1] ← A[j-1]
      i ← j
    }
  }
  A[i-1] ← temp_r


Heapsort(r[], n):
  for i ← n/2 downto 1 {
    Downward_heapify(r[], i, n)
  }
  for i ← n-1 downto 1 {
    r[i] ↔ r[0]
    Downward_heapify(r[], 1, i)
  }
```

(a) (10 points) Consider the following integers in an unsorted array $A = [4,1,3,2,16,9,10,14,8,7]$. Show the array after the execution of the first for-loop of the `Heapsort` algorithm. In other words, construct a max-heap using iterations of `Downward_heapify`.

(b) (10 points) You are also given the following routine that allows you to maintain a max-heap.

```
Upward_heapify(A[], n):
  new ← A[n-1]
  child ← n-1
  parent ← (child-1)/2
  while A[parent] < new and child > 0 {
    A[child] ← A[parent]
    child ← parent
    parent ← (child-1)/2
  }
  A[child] ← new
```

Give an $O(\log n)$ implementation of the operation `Heap_Delete(A[], i, n)` that deletes `A[i-1]` from a max-heap of n elements. After the deletion, `A[0..n-2]` should contain the remaining items and remain a max-heap.

5. (20 points) Consider the following partitioning algorithm:

```
Partition(A[], lb, ub):
   pivot ← A[ub]
   down ← lb - 1
   for up ← lb to ub {
      if A[up] ≤ pivot {
         down ← down + 1
         A[down] ↔ A[up]
      }
   }
   return down
```

(a) (4 points) Consider the following integers in an unsorted array $A = [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]$. Show the array after the application of the partitioning algorithm. What is the value of `down` returned by the algorithm, assuming that the array starts at position 0.

(b) (4 points) What is the maximum number of times that an element can be moved in a single call of `Partition(A[], lb, ub)`? Give an example to justify your answer.

(c) (4 points) Write down a quicksort algorithm `Quicksort(A[], lb, ub)` using the sub-routine `Partition(A[], lb, ub)`.

(d) (8 points) Modify the algorithms `Partition(A[], lb, ub)` and `Quicksort(A[], lb, ub)` to use the mean (or average) of each subarray as the pivot. Note that the mean may not be an element of the subarray.

Question	Max	Score
1	20	
2	20	
3	20	
4	20	
5	20	
Total	100	