ECE368 Mid-Term Exam 2
Spring 2005

Monday, April 11
7:00-8:00pm
GRIS 180

READ THIS BEFORE YOU BEGIN

This is a *closed book* exam. Calculators are not needed nor are they 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. *An answer without justification would not receive full credit.* Be *concise*. It is fine to use the blank page opposite each question for your work.

This exam consists of 8 pages (plus this cover sheet and a grading summary sheet at the end – total of 10 pages); 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 CLASS NUMBER 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:`

`Class Number:`

`login:`

`Signature:`

DO NOT BEGIN UNTIL INSTRUCTED TO DO SO ...

1. (8 points) In homework 4, we perform partitioning and quicksort as follows:

```
Partition_hw(A[], lb, ub):
  pivot ← A[lb]
  down ← lb - 1
  up ← ub + 1
  while TRUE {
    do {
      up ← up - 1
    } while A[up] > pivot
    do {
      down ← down + 1
    } while A[down] < pivot
    if down < up {
      A[down] ↔ A[up]
    } else {
      return up
    }
  }


Quicksort_hw(A[], lb, ub):
  if (lb < ub) {
    pivot_idx ← Partition_hw(A[], lb, ub)
    Quicksort_hw(A[], lb, pivot_idx)
    Quicksort_hw(A[], pivot_idx +1, ub)
  }
```

What is the complexity of the function call `Quicksort_hw(A[], 0, n-1)` when all elements in the array `A[0..n-1]` have the same value?

2. (16 points) In the class, we perform partitioning and quicksort as follows:

```
Partition_lec(r[], lb, ub):
  pivot ← r[lb]
  down ← lb
  up ← ub
  while down < up {
    while r[down] ≤ pivot and down < ub {
      down++
    }
    while r[up] > pivot {
      up--
    }
    if down < up {
      r[down] ↔ r[up]
    }
  }
  r[lb] ← r[up]
  r[up] ← pivot
  return up


Quicksort_lec(r[], lb, ub)
  if lb ≥ ub {
    return
  }
  pivot_idx ← Partition_lec(r[], lb, ub)
  Quicksort_lec(r[], lb, pivot_idx-1)
  Quicksort_lec(r[], pivot_idx+1, ub)
```

Write down the pseudo-code of a new quicksort algorithm, using the mean (or average) of each subarray as the pivot. Note that the mean may not be an element of the subarray. Use the empty page opposite this page if necessary.

3. (10 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:

```
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)
   }
```

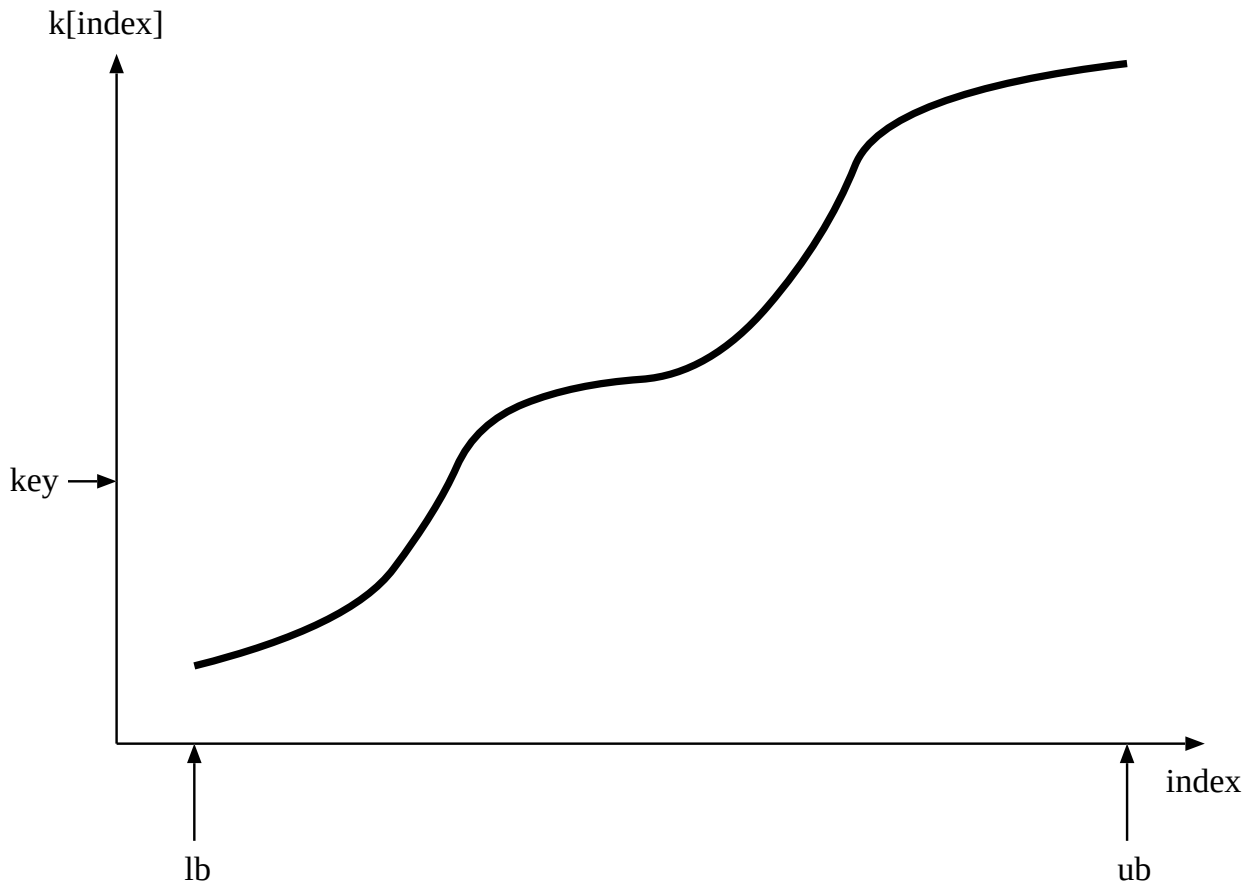
You are now given a *min-heap*. You are also given a new routine `Downward_heapify` to maintain the min-heap property properly. Suggest two different ways to sort an array in *ascending order* and *in-place* using the basic idea of heapsort. For each method you propose, write down its corresponding pseudo-code.

4. (16 points) Consider an unsorted array of n elements. You have been asked to design an algorithm to obtain the largest i numbers in sorted order. (Hint: Use algorithms in Questions 1, 2, and 3.)

a. (8 points) Inspired by all the sorting algorithms that you know, design an algorithm (as pseudo code or in words) to obtain a sorted list of elements ranked i-th and higher. The goal here is achieve the lowest possible complexity for your algorithm. What is the complexity of your algorithm in terms of n and i?

b. (8 points) Suppose there exists an algorithm that can return the i-th largest element in an unsorted array in $O(n)$, design a new algorithm (as pseudo code or in words) such that it has a lower time complexity than that in (a). What is the complexity of your algorithm in terms of n and i?

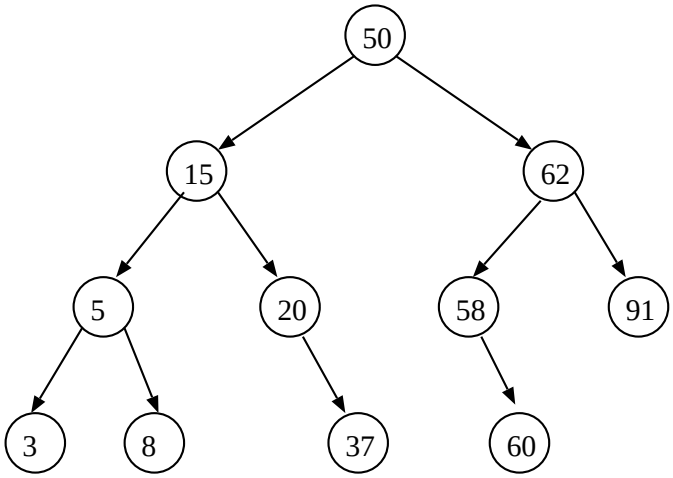
5. (8 points) The following figure plots the keys against their respective indices in a sorted array `k[lb..ub]`. The search key is shown on the vertical axis.



(a)

Show in the figure the indices that are probed in three iterations of interpolation search.

6. (16 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 operation. Clearly specify the rotation operations that you have used. Use the blank page opposite this page if necessary.



7. (16 points) Starting from an empty 2-3-4 tree (B-tree of order 4), perform the following operation: Insert 55, insert 66, insert 77, insert 88, insert 99, insert 11, insert 22, insert 33, insert 44, insert 40, insert 100, delete 44, and delete 11. Draw the resulting B-tree after each operation. Clearly specify the operations (splitting, borrowing, or concatenating/merging) that you have used. Also, state whether you are using the bottom-up approach or the top-down approach to perform insertion/deletion. Use the blank page opposite this page if necessary.

8. (10 points) Calculate the minimum and maximum number of *nodes* in a height-balanced tree of height h, for $h = 1,2,3,4,5$. Also calculate the minimum and maximum number of *keys* in an order-4 B-tree of height h, for $h = 1,2,3,4,5$. Use your results to complete the following table:

height	height-balanced tree		order-4 B-tree	
h	minimum	maximum	minimum	maximum
1				
2				
3				
4				
5				

Question	Max	Score
1	8	
2	16	
3	10	
4	16	
5	8	
6	16	
7	16	
8	10	
Total	100	