# ECE368 Exam 2
# Spring 2010

*Wednesday, April 21, 2010*
*9:30-10:20am*
*EE 236*

*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* 50 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 7 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."*

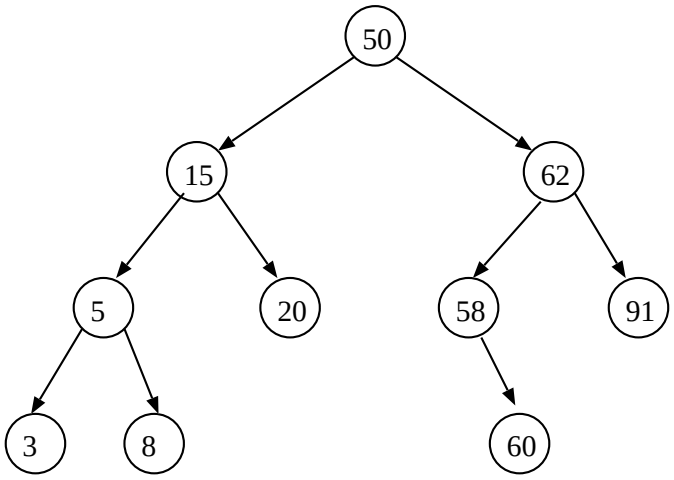
`Printed Name:`

`login:`

`Signature:`

## DO NOT BEGIN UNTIL INSTRUCTED TO DO SO ...

**1. (30 points)** The following diagram shows a *height-balanced* binary search tree (AVL-tree). Performs the following successive operations: Insert 10, insert 30, and delete 62. When you delete a non-leaf node with two child nodes, replace the deleted node with its immediate in-order successor. 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. Use the blank page opposite this page if necessary.



**2. (30 points) (a) (15 points)** In the heapsort algorithm introduced in class, we use a max-heap to sort an array `A[0..n-1]` in ascending order. One version of the heapsort algorithm is as follows:

```
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


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(A[], n):
  for i ← 1 to n-1 {
    Upward_heapify(A[], i+1)
  }
  for i ← n-1 downto 1 {
    A[i] ↔ A[0]
    Downward_heapify(A[], 1, i)
  }
```

Consider an array `A = [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]`. Show the array after the excecution of the for-loop to construct max-heap using iterations of `Upward_heapify`.

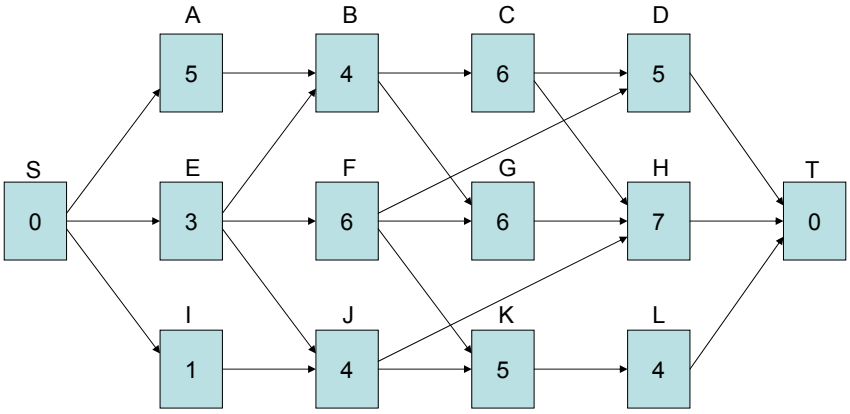
**(b) (15 points)** In the class, we perform partitioning and quicksort as follows:

```
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)
```

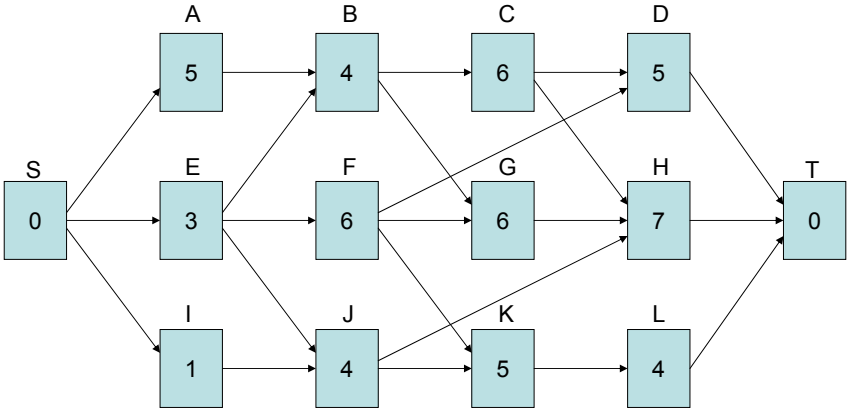
Modify the partitioning algorithm to use the mean (or average) of `r[lb..ub]` as the pivot. Write down the pseudo-code of a new quicksort algorithm based on the modified partitioning algorithm. Note that the mean may not be an element of the subarray.

**3. (40 points) (a) (10 points)** The following figure shows a project with various tasks depicted by rectangles. The time it takes to complete each task is shown within the corresponding rectangle. A task cannot start unless all its predecessors have been completed. The project starts with task $S$ and ends with task $T$.

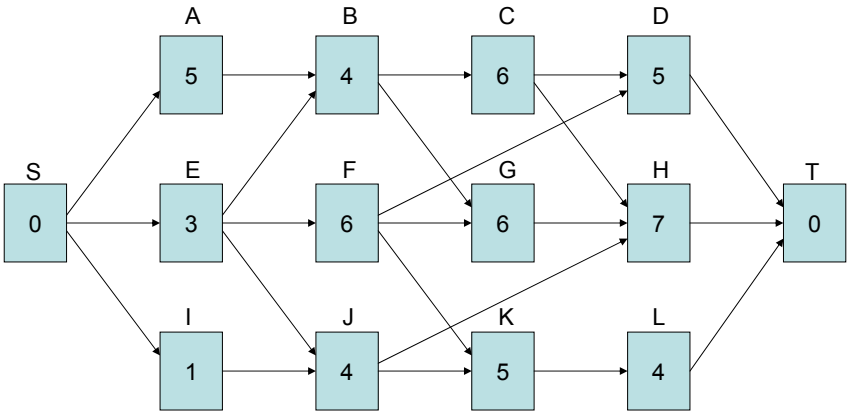



Assume a graph representation of the project, where a node in the graph is a task, and a directed edge $< u, v >$ in the graph means that task $u$ has to be completed before task $v$ can start. Assume that the nodes are sorted according to the following order {`A, B, C, D, E, F, G, H, I, J, K, L, S, T`} in the node list `V[G]` as well as in the adjacency list `adj[u]` of every node. Show the ordering of nodes produced when you run the depth-first-search-based topological sort algorithm on the graph representation of the project.

**(b) (10 points)** Assume that the project starts at time 0. Write down in the following figure the earliest start time (EST) and the earliest finish time (EFT) of each task.



**(c) (10 points)** Based on the earliest finish time of the project you have computed in **(b)**. Write down in the following figure the latest start time (LST) and the latest finish time (LFT) of each task such that the finish time of the project is not affected.




**(d) (10 points)** How many critical paths are there altogether? Write down all the critical paths in the project.

| Question | Max | Score |
|---|---|---|
| 1 | 30 | |
| 2 | 30 | |
| 3 | 40 | |
| Total | 100 | |