CSIS385 Algorithms

Lab #6 : Divide/Decrease & Conquer Algorithms

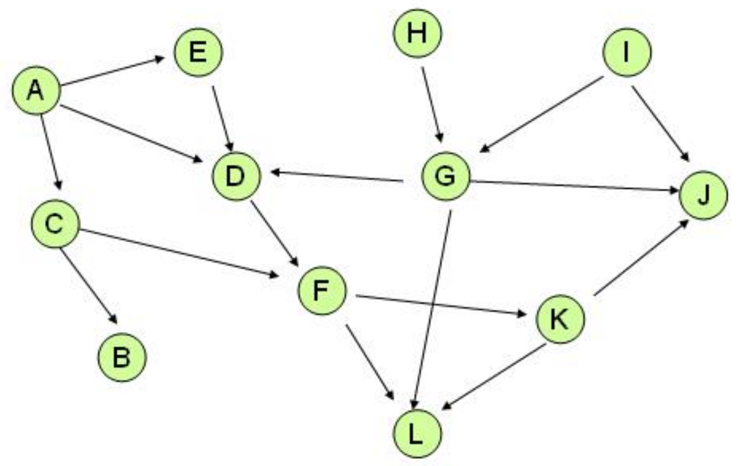
Names: ____

Learning goals:

- to understand the source-removal algorithm for topological sort
- to be able to apply the concept of binary search to new problems
- to better understand Mergesort and Quicksort and their running times

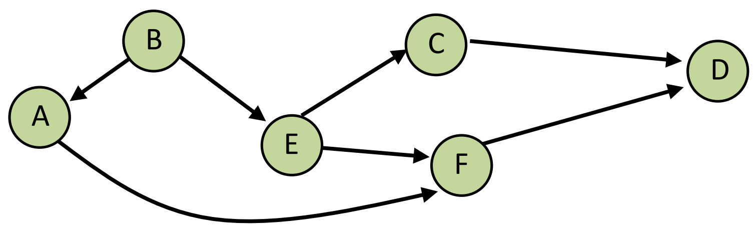
1. (5 Points) Recall that a topological ordering is a list of a directed graph's vertices in which for every edge in the graph, the vertex where the edge starts comes before the vertex where the edge ends. Below is pseudocode for the Source-Removal Algorithm which computes a topological ordering of the vertices in a directed acyclic graph (DAG). Apply the algorithm to the DAG below. List the vertices in the order in which they are outputted by the algorithm.

```
While there exists a vertex v with in-degree 0
        Output vertex v
        Delete the vertex v and all out going edges
```



Output order: ____

2. (5 Points) The Source-Removal algorithm gives one topological ordering of the vertices, but there may be others. For the graph below, how many different valid topological orderings of its vertices are there?



3. Finding the maximum.
   a. (3 Points) Consider an unordered array A[0…n-1] of n integers. How quickly can you find the maximum? Describe your algorithm in one sentence below and give its worst case asymptotic running time.

   b. (3 Points) Considered a sorted array A[0…n-1] of n integers. How quickly can you find the maximum? Describe your algorithm in one sentence below and give its worst case asymptotic running time.

   c. (9 Points) A challenge! Now consider a sorted array A[0..n-1] of unique integers that has been shifted some positions to the right with wrap around. For example, [19, 30, 35, 2, 3, 7, 9, 10, 15] is a sorted array that has been shifted 3 positions. As another example, [18, 20, 21, 24, 25, 27, 29, 35, 42, 5, 15] is a sorted array that has been shifted 9 positions. Given an array shifted by some unknown amount, the problem is to develop an efficient algorithm to find the maximum value in it. Below, design and write pseudocode for an algorithm whose worst case running time is asymptotically faster than linear.

```
// Returns the maximum value in the shifted array A[left…rite].
// The first call to this algorithm will be findMax( A[0…n-1], with left = 0 and rite = n-1.
ALGORITHM findMax( A[left…rite] )
```

4. Below is the Mergesort algorithm from class.

```
ALGORITHM mergesort(A[0..n-1])
  if n>1
        copy first half of array A into a temp array B
        copy second half of array A into a temp array C
        mergesort(B)
        mergesort(C)
        // merge B and C into A
        merge(B, C, A)
```

```
ALGORITHM merge(B[0..p-1], C[0..q.1], A[0..(p+q-1)])
        while B and C have more elements
                choose smaller of items at the start of B or C
                remove the item from B or C & add it to the end of A
        copy remaining items of B or C into A
```

a. (5 Points) Show the recursive call tree when merge sorting the array A = [90, 70, 40, 20, 30, 50, 80, 10]. Show all calls to mergesort and merge, similar to how the first call is done below, where ms is short for mergesort, m is short for merge, and the values in arrays B and C when used as input to merge still need to be filled in.

ms(A = [90, 70, 40, 20, 30, 50, 80, 10])
m(B = [   ], C = [   ] , A[…])

b. (2.5 Points) How many calls to mergesort were made (counting the first call): ___________

c. (2.5 Points) How many calls to merge were made: ___________

5. (12 Points) In Quicksort, the Partition step picks a pivot and then rearranges the array ele
those smaller than the pivot come first, followed by the pivot, followed by those larger than the pivot.
Below is pseudocode for partitioning the values A[left…rite] about the pivot element A[left].

```
ALGORITHM Partition( A[left…rite] )
  // A[left…rite] is an array of unique non-negative integers
  p ← A[left]
  i ← left
  j ← rite + 1
  repeat
          repeat i ← i + 1 until ( A[i] >= p )  or  ( i > rite )
          repeat j ← j - 1 until A[j] <= p
          if ( i < j ) swap A[i] and A[j]
          // show contents of array A and values of i and j at this point in code
  until i >= j
  swap A[left] and A[j]
  return j
```

Suppose the array A[0…9] below is given as input to the Partition algorithm. Show the contents of the array after each iteration of the **outer repeat loop**.

12	**3**	**1**	**19**	**25**	**8**	**34**	**7**	**45**	**23**

After first iteration of outer repeat loop: i = __________ j = ___________

After second iteration of outer repeat loop: i = __________ j = ___________

After third iteration of outer repeat loop: i = __________ j = ___________

After outer repeat loop is done and last swap (just before the "return j" statement) is done:

6. (6 Points) What value does Partition return in this specific example? ____________

7. (5 Points) Describe in words what the last swap in Partition does.

8. (4 Points) In general, describe in words what “return j” returns when called for an arbitrary array A[left…rite].

9. (10 Points) What is the worst case Big O running time of Partition(A[left…rite])? Express your answer in terms of n, the number of elements between left and rite. Explain why your analysis is correct.

The complete Quicksort algorithm is given below. Use it to answer the questions that follow.

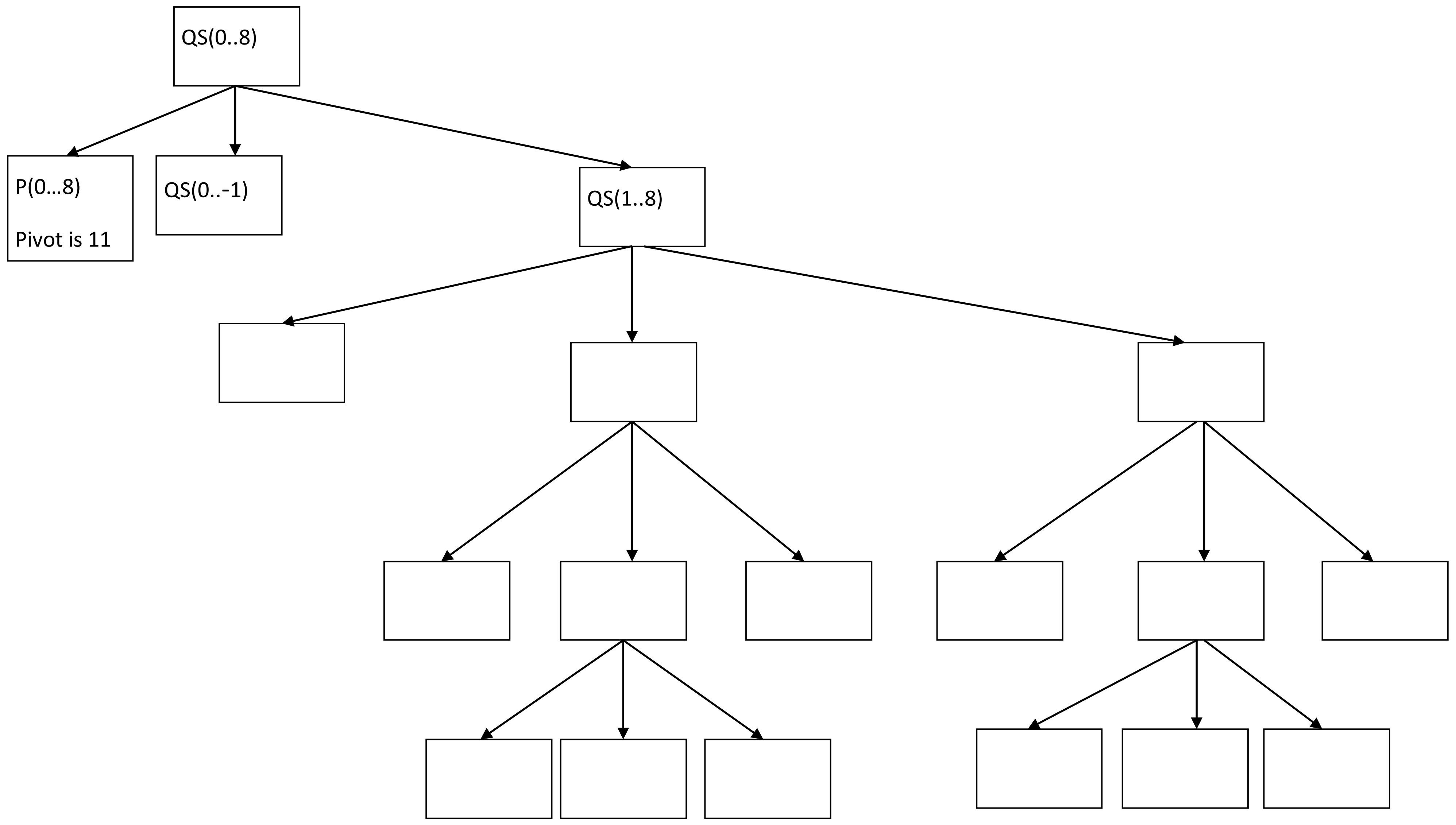
```
Quicksort( A[left…rite] )
    if ( left < right )
      s ← Partition( A[left…right] )
      Quicksort( A[left…s -1]  )
      Quicksort( A[s+1…rite]  )
```

```
ALGORITHM Partition( A[left…rite] )
    p ← A[left]
    i ← left
    j ← rite + 1
    repeat
        repeat i ← i + 1 until ( A[i] >= p )  or  ( i > rite )
        repeat j ← j - 1 until A[j] <= p
        if ( i < j ) swap A[i] and A[j]
    until i >= j
    swap A[left] and A[j]
    return j
```

10. (20 Points) Consider the tree of subroutine calls made when Quicksort(A[0…8]) above is called with array A[] containing the values {11, 55, 22, 77, 88, 44, 99, 33, 66}. Fill in the tree of subroutine calls found on the last page of this lab using the Quicksort and Partition algorithm in this lab. Show calls to both Partition and Quicksort. I have filled in four boxes for you. Follow the style given and abbreviate Partition and Quicksort with P and QS.

11. (4 Points) Write a recurrence formula for Quicksort when the pivot in every call to Partition is the smallest value among all values A[left…rite]. This is the worst case for Quicksort. Find a closed form for your recurrence using any method you wish (do not show your work here). Then write below the Big O worst case running time for Quicksort.

12. (4 Points) Write a recurrence formula for Quicksort when the pivot is the median value in every call to Partition. To keep things simple, you may assume that there are exactly n/2 items to the left of the median value and n/2 to the right. This is the best case for Quicksort. Find a closed form for your recurrence using any method you wish (do not show your work here). Then write below the Big O best case running time for Quicksort.


QS(0..8)
P(0…8)
Pivot is 11
QS(0..-1)
QS(1..8)