ECE368 Exam 2
Spring 2016

Thursday, April 7, 2016
15:00-16:15pm
ARMS 1010

READ THIS BEFORE YOU BEGIN

This is a *closed-book, closed-notes* exam. Electronic devices are not allowed. The time allotted for this exam is exactly 75 minutes.
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 equation (or at the back of each question) for your work. Do draw an arrow to indicate that if you do so.
This exam consists 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 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 exam and will be subject to possible disciplinary action."

Printed Name:

Login:

Signature:

DO NOT BEGIN UNTIL INSTRUCTED TO DO SO …

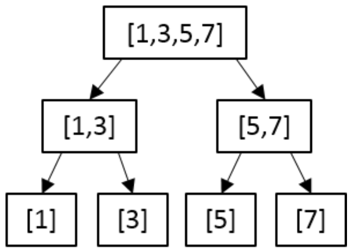
Login: ________________

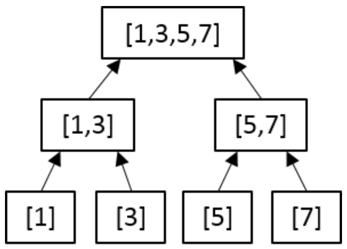
1. MergeSort (20 Pts Total)

The following code implements mergesort recursively:

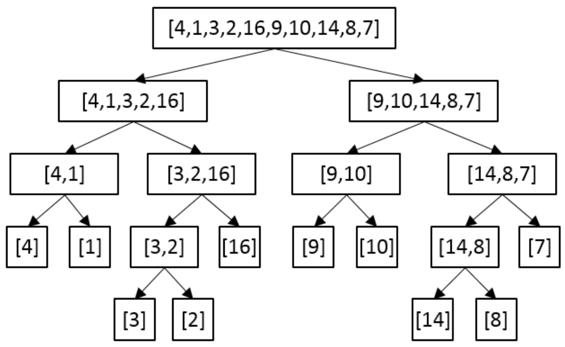
```
void Mergesort (int *arr, int first, int last) {
        if (first >= last)
                return;
        int mid = (first + last)/2;
        Mergesort (arr, first, mid);
        Mergesort (arr, mid, last);
        Merge (arr, first, mid, last); // Merge(arr, first, mid, last) merges the two sorted sub-lists
                                       // arr[first:mid-1] and arr[mid:last-1] into a single sorted list
}
```

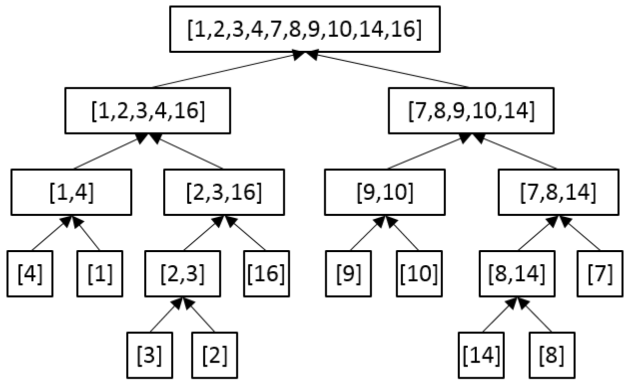
The left figure shows the partitioning process and the right shows the merging process when the input array is arr = [1, 3, 5, 7].





a) **(10 pts)** Consider an unsorted array arr = [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]. As in the preceding figures, show the **partitioning process** and **merging process** when Mergesort is applied.





correct → 10points,

half correct → 7 points,

[4,1,3][2,16][9,10,14][8,7] →

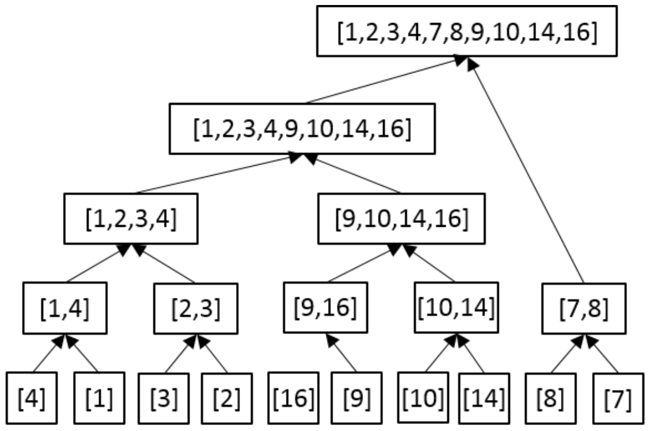
(Question 1 continued on the next page)

Login: ________________

b) **(10 pts)** Mergesort can also be implemented iteratively as follows:

```
void Iter_Mergesort (int *arr, int n) {
        int size=1;
        while (size<n) {
                int i=0;
                while (i < n-size) {
                        Merge (arr, i, i+size, min(i+2*size, n));
                        // Merge(arr, first, mid, last) merges the two sorted sub-lists
                        // arr[first:mid-1] and arr[mid:last-1] into a single sorted list
                        i = i + 2*size;
                }
                size *= 2;
        }
}
```

Consider an unsorted array arr = [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]. As in the preceding figures, show the **merging process** when Iter_Mergesort is applied.



correct → 10points,

more than half → 5-6points

2. Heaps and HeapSort (20 Pts Total)

It is required to implement the code for the HeapSort Algorithm that we discussed in class. You may assume the functions "parent, left and right" have been implemented already.

```
//Returns index of parent node given index of child
int parent(int child_index);

//Return index to left or right child of a parent node
int left(int index_parent);
int right(int index_parent);

// Convert the array representation of a complete tree into a heap of size n
void max_heapify( int *Array, int N);

// Pop and return the top of a heap defined in an array of size n
int max_heap_pop( int *Array, int N);

//Heap Sort
void heapsort(int *Array, int N)
{
   max_heapify(Array,N);

   for (int i=n-1; i > 1; --i)
      array[i] = max_heap_pop( Array, i+1);
}
```

a) Implement the functions ***max_heapify (10 Pts)*** and ***max_heap_pop (10 Pts)***. *You may write C code or list each step of your function in bullet points.*

(extra space is given on the next page)

Login: ________________

Many possible solutions but the pseudo codes are:

```
Max Heapify ( Array, N)
  $\textbf{for}\ i\ from\ 1\ to\ N$
      $index = i$
      $\textbf{while}(Array[parent(index)] < Array[index]\ \&\&\ index > 0)$
          $swap(Array, index, parent)$
          $index = parent(index)$
      $\textbf{end while}$
  $\textbf{end for}$
$\textbf{end Max Heapify}$
```

-Start from top node traversing in a BFS.
-**Upward Heapify** for each node:
    -If child is smaller than parent, swap.
    -Repeat swapping until parent is larger.

-*May also start with a subheap at* $i = N/2$ *and increase* i.
- Swap parent with **largest** child if child > parent. (**Downward Heapify)**
- If swap was made, heapify on the child that was swapped (either with recursion or with a while loop) as long as child < N.

Grading Criteria.
+3 Points if swapped when parent is larger.
+3 Pts if looped through all the nodes.
+2 Pts if used while/recursion to check next parent/children.
+2 Pts if checked for boundary conditions.

```
Max_Heap_Pop ( Array, N)
  $maximum = Array[0]$
  $Array[0] = Array[N-1]$
  $index = 0$
  while$(index < (N-1)/2)$
    if $(Array[left(index)] > Array[right(index)])$
      $MaxChild = Array[left(index)]$
      $MaxIndex = left(index)$
    else
      $MaxChild = Array[right(index)]$
      $MaxIndex = right(index)$
    end if
    if$(Array[index] \geq Array[MaxChild]$
      break
    else
      swap $(Array, index, MaxIndex)$
      $index = MaxIndex;$
    end if
  end while
Return maximum;
end Max_Heap_Pop
```

-Store Array[0]
-Replace Array[0] with last node in heap.
-Heapify down replacing parent with its largest child recursively/while loop.
-Return stored value

Grading Criteria.
+3 Points if returned Array[0].
+3 Points if heapified on reduced-size heap
(swapped with largest child).
+2 Pts if used while/recursion to check next parent/children.
+2 Pts if checked for boundary conditions.

-Many students used their Max_Heapify function. They received full credit if they specified a size reduced heap. Otherwise they got -2 Pts.

-Many students did not replace root node with last node in the heap. They still got full credit but should note that by omitting that step, they do not keep a complete B-tree.

Login: ________________

3. QuickSort (25 Pts Total)

a) **(10 pts)** Sort the following list using quicksort (Make pivot be the 1st element of the sub-array and use insertion sort for any sub-list of size 4 or less). Show the new array in every step (including when n < 5).

______

34	15	65	59	68	42	40	80	50	65	23

Solution 1

34 (pivot)	15	65	59	68	42	40	80	50	65	23

(swap: 65 - 23)

34 (pivot)	15	**23**	59	68	42	40	80	50	65	**65**

(swap: 34 - 23)

23	15	**34**	59	68	42	40	80	50	65	65

(insertion sort: {23, 15})

15	**23**	34	**59 (pivot)**	68	42	40	80	50	65	65

(swap: 68 - 50)

15	23	34	59 (pivot)	**50**	42	40	80	**68**	65	65

(swap: 59 - 40)

15	23	34	**40**	50	42	**59**	80	68	65	65

(insertion sort: {40, 50, 42} {80, 68, 65, 65})

15	23	34	**40**	**42**	**50**	59	**65**	**65**	**68**	**80**

Solution 2

34 (pivot)	15	65	59	68	42	40	80	50	65	23

______(swap: 34 - 23)

23	15	65	59	68	42	40	80	50	65	**34 (pivot)**

(swap: 65 - 34)

23	15	**34**	59	68	42	40	80	50	65	**65**

(insertion sort: {23, 15})

15	**23**	34	**59 (pivot)**	68	42	40	80	50	65	65

(swap: 59 - 65)

15	23	34	**65**	68	42	40	80	50	65	**59 (pivot)**

(swap: 65 - 50)

15	23	34	**50**	68	42	40	80	**65**	65	59 (pivot)

(swap: 68 - 40)

15	23	34	50	**40**	42	**68**	80	65	65	59 (pivot)

(swap: 68 - 59)

15	23	34	50	40	42	**59**	80	65	65	**68**

(insertion sort: {50, 40, 42} {80, 65, 65, 68})

15	23	34	**40**	**42**	**50**	59	**65**	**65**	**68**	**80**

correct ⟶ 10points,

minor mistake ⟶ 6-8points

major mistake ⟶ 3-5points

Question 3 continued on the next page)

b) **(5 Pts)** What is the average memory requirement?

O(ln n) for stack memory.

correct → 5points,

otherwise → 1points

c) **(3 Pt)** Suppose you are asked to implement an efficient QuickSort and your team is debating about four different approaches to choose a pivot, which one is the best option?

i) Pick a random element in array.
ii) Pick 3 random elements in array and find the median.
iii) Do part ii) 3 times and pick median of the 3 medians.
iv) Pick first, medium, and last element of the array and find median.

correct → 3 points,

otherwise → 1points

d) **(7 Pts)** Justify your answer to part c).

-If you assume to have a large n, the goal is to find the pivot which will split the array in two perfect halves $(\frac{1}{2}n, \frac{1}{2}n)$. This would be picking the exact median.

-Choosing random pivots will divide(on average) items into sets of size $\frac{1}{4}n \ and \frac{3}{4}n$.

-Choosing random elements and finding the median of three will yield(on average) a partition of: $\frac{5}{16}n, and \frac{11}{16}n$.

-Similarly, choosing median of first, medium and last element will divide (on average) items into sets of $\frac{5}{16}n, \frac{11}{16}n$ since these numbers are random too.

-By choosing median of 3 for 3 times, you increase your sample size which will give you a pivot closer to the true median $(\frac{n}{2}, \frac{n}{2})$ while not increasing the time complexity significantly.

-Mention it has a closer value to the true median: 7 Pts

-Choosing option iv) and arguing a better distribution or more efficient time complexity than option iii): 5 Pts.

-All other answers will vary depending on robustness.

Login: ________________

4. Choosing Sorting Algorithms (15 Pts Total)

You are given an array of N items to sort. You know that the value of each integer falls between 1 and 1000N.

a) **(5 Pts)** What is the algorithm that gives the **best** asymptotic time complexity?

Bucket Sort (because the maximum value m=O(n)), the asymptotic time complexity is then O(m+n)=O(n)

Correct → 5 points
Radix Sort → 3 or 4 points
Other Sorting algorithm → 0 to 2 points (depending on justification or further explanation)

b) **(10 Pts)** Explain how this algorithm works and give its time complexity in Big-O notation?

Description of Bucket Sort available in lecture slides
Correct description of bucket sort → 10 points
Incomplete description of bucket sort or description of radix sort → 7 to 9 points
Description of other sorting algorithms → 4 points or less

5. Graphs (20 Points Total)

a) **(5 Pts)** What are the conditions on the number of edges |E| in terms of the vertices |V| of any connected graph?

|E| greater than or equal to |V|-1 (there is also the trivial upper bound on the maximum number of edges on any graph, |E| is at most "V choose 2")

Correct lower bound → 5 points
Only the trivial upper bound → 1 point

b) **(6 Pts)** Justify the conditions for part a).

Proof by induction:
Base case: A graph with two vertices is connected only if it has at least one edge
Induction hypothesis: Any connected graph with n vertices has to have at least n-1 edges
Induction Step: We want to prove that any graph with n+1 vertices has to have at least n edges. Consider the vertex induced subgraph of the vertices 1 up to n, if the original graph is connected, then the new subgraph is connected. By the induction hypothesis, the subgraph has at least n-1 edges. Further, if the original graph is connected, then the vertex n+1 has to have at least one edge to the considered subgraph. It follows that the original graph has to have at least n edges.

Correct proof → 6 points
Just correct intuition → 2 to 4 points
Proof of trivial upper bound → 0 or 1 point

c) **(3 Pts)** Is the following statement true or false?

C1: Any connected graph is a forest
False

Correct → 3 points
Not Correct → 0 points

d) **(6 Pts)** Justify your answer to part c).

A Forest is an acyclic graph, a connected graph can have cycles
Correct → 6 points
Example without justification → 2 to 4 points
Proof that statement is true → 0 points

Login: ________________

Question	Score
Q1	/20
Q2	/20
Q3	/25
Q4	/15
Q5	/20
Total	/100