Part I: Multiple Choice (2 points each)

___ 1. Which of the following sorting algorithms is *not* inversion-bound? Indicate *all* correct answers (there may be more than one)

A. MinSort C. BubbleSort E. None of the above
B. LibrarySort D. InsertionSort QiuckSort

___ 2. Which complexity class below is little-oh of each of the other classes shown here?

A. $\Theta(n^2)$ B. $\Theta(2^n)$ C. $\Theta(n \log n)$ D. $\Theta(n!)$ E. $\Theta(\sqrt{n})$ F. $\Theta(n \log_{3/2} n)$

___ 3. Suppose A is an algorithm designed to solve a particular problem. It makes sense to formulate a loop invariant for A if (select only one)

A. A is an iterative algorithm and we are attempting to compute its asymptotic running time
B. A is an iterative algorithm and we are attempting to prove correctness of A
C. A is a recursive algorithm and we are attempting to compute its asymptotic running time
D. A is a recursive algorithm and we are attempting to prove correctness of A

___ 4. The following problems were discussed in class. Which of these are known to have solutions having o(n) (little-oh) worst-case running time? (For number-theoretic problems, running time is computed in terms of *value* rather than *size* of input in bits.) Indicate *all* correct answers (there may be more than one).

A. Searching a sorted list
B. Computing the lcm of two positive integers
C. Determining whether a given positive integer is prime
D. Reversing the elements of a given list
E. Computing the nth Fibonacci number
F. All of the above

___ 5. How many *good pivots* are in the array [1,7, 3, 2, 5, 6]? Good pivots are used in the average case analysis of QuickSort and QuickSelect. (Select only one answer.)

A. 2
B. 3
C. 4 D. 5

___6. Which of the following is a solution to this recurrence relation?

$$T(1) = 1; \quad T(n) = T(\lceil \frac{n}{2} \rceil) + 2n^3.$$

A. $\Theta(n^3 \log n)$
B. $\Theta(n^3)$
C. $\Theta(n)$
D. $O(n^3 \log n)$

___ 7. Which of the following functions is (are) ***not*** $\Omega(n^2)$? Choose all that apply.

A. $f(n) = n^3$
B. $f(n) = n^2 + n$
C. $f(n) = n\log n + 100$
D. $f(n) = n!$

___ 8. Suppose A is a list of *n* integers arranged in random order, and suppose *z* is an integer that is larger than every element of A. If the Binary Search algorithm is run with input A and z, which of the following statements is correct? (Select only one answer.)

A. Binary Search will correctly output "false" (indicating z was not found in A) but its running time will be slowed down to Theta(n) or worse because of the fact that elements of A are not in sorted order

B. Binary Search will have a running time of O(log n) but output for inputs A and z could be "false" in some cases and could be "true" in other cases

C. Binary Search will correctly output "false" (indicating z was not found in A) and its running time will be O(log n)

Part II: Short Answer

1. (6 points) Give a short description (do *not* include details) of an optimal algorithm for solving each of the following problems discussed in class. If the algorithm you have in mind has a well-known name, just give that name. If your algorithm makes use of another algorithm with a well-known name, make use of the name rather than explaining how that part of your algorithm works.

   Example: The brute force solution of the SubsetSum Problem (S,k) first calls the PowerSet algorithm to get a list of all subsets of S, and then sums each of these subsets to check if any has sum = k.

Second Smallest Problem (find the second smallest element of an array of integers of size at least 2)

Algorithm SecondSmallest2(arr)
Input : An array of integers of length ⊠ 2
Output : The second smallest element in arr
for i ⊠ 0 to 1 do
nextPos ⊠ minPos(arr, i)
swap(arr, i, nextPos)
return arr[1]

Remove Duplicates (given a size-n list of integers, return the list without duplicates)

Using HashMap Contains.key

LCM Problem (given two positive integers m, n, find the least common multiple of m and n)

First we call gcd(m,n) and store in g. Then from MathFact1 we can find LCM (m,n)

Step 1. Compute g = gcd(m,n)
Step 2. Compute lcm = mn/g (using Math Fact 1)
If we can find a fast gcd algorithm, then we can improve our first lcm algorithm.

2. (3 points) Solve the following recurrence relation

$$T(1) = 1; \quad T(n) = T(\lceil \frac{n}{2} \rceil) + 2n^3.$$

T(1)= 2; T(n) = 3T([n/3]) + 4n

a=3, b=3, c= 4, d=2, k=1 , b'=3=a Ans => O(nlogn)

3. (4 points) Formulate a recurrence relation for the running time of the following algorithm (do *not* solve your recurrence relation). Write your answer in the box provided.

Algorithm recurSum1(n)
Input: a non-negative integer n
Output: the sum of $n+\frac{n}{2}+\frac{n}{4}+\ldots+1$
if (n = 0 || n = 1) **then return** n
return n + recurSum1($\frac{n}{2}$)

..... 2
....n+4+T(n/2)

T(n) = ... Cn +T(n/2)

4. (3 points) True or false: There is a comparison-based sorting algorithm which, when run on an array A of 5 distinct integers, requires only 6 comparisons to sort A, even in the worst case. *Prove your answer*. (To prove "True", name an algorithm that does this and go through the steps of the algorithm to indicate where the comparisons occur. To prove "False," use some facts established in the course that demonstrate it is impossible to sort 5 elements with only 6 comparisons in the worst case.)

Comparison .. (log n!) base 2....
Ceil(log 5!) = log 120 => log 128=> 7 False

III. Long Answer

1. (6 points) The diagram below shows a MergeSort recursion tree. Prove that the MergeSort recursion tree, representing the behavior of MergeSort when running on input list of size n, necessarily has height O(log n).

L =2 the power of h

log L = h log 2

h= log L

h= O(log n)

= n.O(log n)

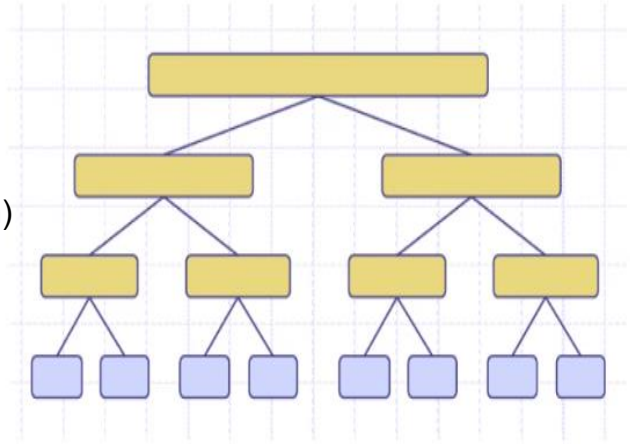
=> O(nLog n)

For :-

Partition = O(n)

Merge= O(n)

h= O(log n)



.... n

...n/2

...n/4

.
.
.
0

3) (6ponits) The following algorithm attempts to reverse a given input list of integers.

```
Algorithm reverse(List list)
Input: a non-null list of integers
OutPut: the list in reverse order
if(list.size() < 2) then return list
front <- list.remove(0)
reverse(list)
list.addLast(front)
return list.
```

Claim. Counting the initial call to reverse, the reverse method makes n-2 self calls when run on a list of size n>1.
Proof. If size of list is 2, this is obvious. Assume true for n. Then if L has n+1 elements, there is the initial call, and then reverse is called on the list after 0th element is removed (so that list has size n). By induction hypothesis the algorithm makes n-2 self-calls running on that list. So total number of self-calls is 1 + n-2 = n-1, as required.

A. Prove the algorithm is correct.
B. Determain the running time of the algorithm and Prove your Ans

4. [16 points] The *R- Numbers* are defined as follows:

$R_0 = 1, R_1 = 2, R_n = R_{n-1} * R_{n-2} + 1.$
The following algorithm `Rnum` is a recursive algorithm that computes the R- Numbers:

Algorithm `Rnum` (n)
Input: A non-negative integer n
Output: The R-number R_n
if(n = 0 || n = 1) **then**
return n + 1
return `Rnum` (n - 1) * `Rnum` (n - 2) + 1

a. [4] Show that `Rnum` is correct. (You must show that the recursion is valid and that the base case and recursive steps return correct values.)

- Valid recursion since there is a base case and self-calls lead to the base case
- Base case returns correct values
- Assuming Rnum(m) returns a correct value for m < n, we compute the return value of Rnum(n): Rnum(n) = Rnum(n-1)*Rnum(n-2) + 1 $=R_{n-1} * R_{n-2} + 1 = R_n$.

b. [4] Show that `Rnum` has an exponential running time.

- **T(n) >= S(n) (number of self-calls performed by Rnum on input n)**
- **Claim: For n > 1, S(n) >= F_n (the nth Fibonacci number) Proof: Base case: When n = 2, then S(2) =2 > F_1**

Assuming the result for m < n, we have S(n) = 2 + S(n-1) + S(n-2) > S(n-1) + S(n-2) >= F_{n-1} + F_{n-2} = F_n
- **$F_n > (4/3)_n$ for n > 4.**
- **It follows that T(n) has exponential running time**

The following is an iterative algorithm that computes the R-Numbers.
Algorithm It`Rnum` (n)
Input: A non-negative integer n
Output: The R-number R_n
storage <- new array(n+1)
if n = 0 or n = 1 then return n + 1
storage[0] <- 1
storage[1] <- 2
for i <- 2 to n do
storage[i] = storage[i-1]*storage[i-2] + 1
return storage[n]

c. [4] Prove It`Rnum` is correct (you must formulate a loop invariant, prove that it holds after each iteration of the loop, and show that the final value obtained in the loop is the correct output for the algorithm).

Loop invariant: I(i) : value in storage[i] is R_i
This is true at the end of i = 2 loop. Assuming true for i pass. Then storage[i+1] = storage [i] * storage[i-1] + 1 = $R_i * R_{i-1} + 1 = R_{i+1}$
Finally, since storage[n] stores R_n (as we just showed) the algorithm returns the correct value.

d. [4] Give an asymptotic bound for the running time of It`Rnum`.

Just one for loop – so O(n)