**True/False.** (14points) Below are 7 true/false questions. Mark either T or F in

the space provided.

\_\_\_T\_\_ 1. When QuickSelect runs on input array S = [1, 3, 5, 2, 1, 7, 2, 1, 4, 1], with k = 4, the return value is 1.

\_\_T\_\_ 2. All known comparison-based sorting algorithms, when run on an array of 6 distinct integers, require, in the worst case, 10 or more comparisons to sort the integers.

.

\_\_\_F\_\_ 3. Suppose the running time of an algorithm is given by the recurrence

*T*(1) = *d*  *(d* > 0)

*T*(*n*) = 2*T*(2*n*/3) + 2*n*.

Then *T*(*n*) belongs to Ω(*n*2).

\_\_\_\_ 4. Although InsertionSort is an inversion-bound sorting algorithm, the generalized version of InsertionSort which we have named LibrarySort in the class is *not* inversion-bound..

\_\_\_\_ 5. Suppose f(n) > g(n) for all n. Then f(n) is *not* O(g(n)).   
Consider f(n) = 2n+1, g(n) = n.

\_\_\_\_\_ 6. n! is o((n+1)!).

\_\_\_\_ 7. Suppose log(*f*(*n*)) is O(log(*g*(*n*))). Then *f*(*n*) is O(*g*(*n*)).

**II.**  **Problems.** Solve the problems below. The more you explain your work, the more

partial credit you will receive (in case your answer is incorrect).

1. [12] Use RadixSort, with two bucket arrays and radix = 11, to sort the following array:

[63, 1, 48, 53, 24, 10, 12, 30, 100, 115, 17].

Show all steps of the sorting procedure. Then explain why the running time is O(*n*).

Return: [1, 10, 12, 17, 24, 30, 48, 53, 63, 100, 115]

*Running Time:* O(n):

* Initializing each of the bucket arrays costs O(n)
* Populating r[] and q[] costs O(n)
* Reading output from q[] is O(n)

1. [10] There are two parts of this problem – you must do ONLY ONE of the two parts. CIRCLE the letter that labels the problem you will do. Part (A) must be done using the non-limit definition of big-oh. In Part (B) you may use the limit definition of little-oh.

1. Show that n2 + 2 is *not* O(n)

B. Show that n1/2 + log n is o(n)

3. [15] A *twisty table*  is a kind of data structure that stores Strings and that uses an array in the background (assume it is already initialized and has as many available slots as necessary without resizing) and that has three primary operations:

* AddOne(x) – adds String x to the background array in the next available slot
* AddThree(x,y,z) – adds Strings x, y, and z to the background array by placing them in the next three available slots.
* ClearAll() – removes all Strings currently in the array by setting them to null.

Show that the amortized running time of ClearAll is O(1). Do the steps required by filling in the following table. Hint: For your amortized cost function, try charging 2 cyberdollars for AddOne, 6 cyberdollars for AddThree and 0 cyberdollars for ClearAll.

|  |
| --- |
| **The cost function c:**  c(AddOne) = 1  c(AddThree) = 3  c(ClearAll) = k, where k is number of non-null Strings in the  array |
| **Your amortized cost function ĉ:** |
| **Show that your amortized cost function never results in negative amortized profit:**    If AddOne is executed k times and AddThree is executed m times, the profit will be k + 3m, which is enough to pay for a ClearAll operation, which would cost k + 3m. |
| **Provide a bound for the amortized cost of running n of these operations in succession.** |
| **Explain why ClearAll has amortized running time O(1):**        Amortized running time is O(n)/n = O(1) |

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

T0 = 0, T1 = 2, Tn = Tn - 1 + 3Tn - 2 + 1.

The following algorithm Tnum is a recursive algorithm that computes the R- Numbers:  
   
Algorithm Tnum (n)

**Input**: A non-negative integer n

**Output**: The T-number Tn

**if**(n = 0) **then**

**return** 0

**else if** (n = 1) **then**

**return** 2

**return** Tnum (n - 1) + 3 \* Tnum (n - 2) + 1

1. [4] Show that Tnum is correct. (You must show that the recursion is valid and that the base case and recursive steps return correct values.)
2. [4] Show that Tnum has an exponential running time.

------------------------------------------------------------------

Question 1: 10 points (3 + 5 + 2)  
 a. Complete the following table. Explain your answer as appropriate.

|  |  |
| --- | --- |
| Algorithm | Worst Case / Stable / In-Place |
| Insertion Sort |  |
| Selection Sort |  |
| Merge Sort |  |
| Quick Sort |  |
| Radix Sort |  |
| Counting Sort |  |

b. Determine whether f is O, o, Big omega or small omega of g where

f=n^kandg=c^n; c>1andk>=1.

c. Showthat 6n2isΘ(n2).

Question 2: 11 points (3 + 3 + 5)

For each of the following recurrences, derive an expression for the running time using iterative, substitution or Master Theorem.

a.

b.

c. Consider the following recurrence algorithm procedure T(n,x)

if(n==1) return true

else {

x=x+n Call T(n-1,x)

}  
 end procedure

i. (2 point) Write a recurrence equation for T(n) Solution:

ii. (3 points) Solve recurrence equation using iterative method i.e. give an expression for the runtime T(n).

Question 3: 12 points (2 + 5 + 5)

a. Illustrate (by drawing) the operations of the mergesort algorithm on the array A={5,3,17,10,84,19,6,22,9}.

b. Use the QuickSelect algorithm to manually compute the 5th smallest element of the array [1, 5, 23, 0, 8, 4, 33]. Assume that the rightmost element is used as the pivot in each case. Show what happens in each self-call, indicating the new input array and the current value of k.

c. Use RadixSort, with two bucket arrays and radix = 6, to sort the following array: [1, 8, 3, 2, 34, 21].

Show all steps of the sorting procedure. Then explain why the running time is O(n). What would be the running time if you used ONE bucket?

* Question 4: 13 points (3 + 6 + 4)
* a. Is mathematics decidable? Explain the Halting problem in your own words (no need to prove).
* b. Given an array A of n numbers, suggest an O(n) expected time algorithm to determine whether there is a number in A that appears more than n / 2 times.

c. Use Decision tree and binary tree basic ideas to prove the following theorem:

"Every comparison based sorting algorithm has, for each n, running on input of size n, a worst case in which its running time is Ω(nlog n)".

h ≥ lg n n  e 

= n lg n − n lg e = Ω(nlgn)

n! >  n n  e 