

Review for Exam 1

Outline of topics covered:

I. Recursion

- a. Solving problem using a solution to a sub-problem of the exact same nature
- b. Terminating condition
- c. Danger of "redoing" recursive calls

II. Recurrence Relations

- a. Iteration technique
- b. General "Master" Formula

III. Sorting

- a. Insertion Sort
- b. Merge Sort
 - i. Merge algorithm
 - ii. Recursive solution to sorting
- c. Quick Sort
 - i. Partition algorithm
 - ii. Different recursive solution to sorting
 - iii. Quick Select
- d. Heaps
 - i. Use for Priority Queue
 - ii. Heapsort
- e. Lower Bound for comparison sorting
- f. Bucket Sort
- g. Radix Sort

I. Recursion

The most difficult thing here is taking a problem and breaking it down into separate components, where at least one of them is a subproblem of the exact same nature. Here is a list of some recursive problems we looked at:

fibonacci

factorial

towers of hanoi

sorting with merge and quick sorts

minesweeper

II. Recurrence Relations

We derived our "master" formula by examining the iteration of a given recursive function definition. All you need to do is apply the formula:

$$T(n) = aT(n/b) + O(n^k)$$

if $a > b^k$, then $T(n) = O(n^{\log_b a})$

if $a = b^k$, then $T(n) = O(n^k \lg n)$

if $a < b^k$, then $T(n) = O(n^k)$

III. Sorting

Basic idea behind the insertion sort: insert each element into an already sorted list, one by one, with successive swaps for the insert.

Basic idea behind merge sort: you can efficiently merge two sorted lists into one. Using this idea, you can recursively sort the first half, then the second, then merge

Basic idea behind quick sort: you can efficiently partition an array into two sides - one with all the values less than the partition element and one with all the elements greater than the partition element. Then you can partition and recursively sort both sides of it.

Heaps can be formed by "bubbling down" the 1st half of the elements in the balanced binary tree. Once a heap is formed, we can insert and delete the minimum value from it in $O(\lg n)$ time. These methods also perform a "bubble up" or "bubble down" function while running, respectively.

Because of the vast number of possible inputs to the sorting problem, it can be shown that any comparison sort runs in $\Omega(n \lg n)$ time.

The bucket sort works for real-valued data that is distributed evenly.

The radix sort works for sorting whole numbers formed by a fixed number of digits.

Sample Questions

For the unsorted array $A[0..15]$ shown below, answer the following questions:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	8	6	10	4	14	9	15	3	13	7	2	16	5	12	1	11

a) (4 pts) Assuming that A is being Merge-Sorted, show the contents of the array right after the 7th call to the Merge method. (Note: there are a total of 15 calls to the Merge method in the entire Merge Sort.)

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value																

b) (4 pts) In an insertion sort of A , which values in the array would occupy the location $A[0]$ at some point during the sort? (Note: 1 is NOT the only answer to this question!!!)

c) (4 pts) In a Quick Sort of A , the partition elements (not indexes!!!) were 8, 4, 2, 6, 12, 10, 15, and 13, in that order. After the partition with the element 15, what are the possible values of $A[12]$?

d) (4 pts) Using logs and factorials in your answer, what is the least number of comparisons that any comparison algorithm sorting the numbers above can make

e) (4 pts) Assuming that $16! = 2.1 \times 10^{13}$, $\log_2 2.1 = 1.07$, and $\log_2 10 = 3.32$, what is the minimal integer greater than your answer for part d? This value denotes the actual minimal number of comparisons necessary to sort an array of size 16. (Note: $\log ab = \log a + \log b$.)

(10 pts) Pascal's Triangle contains combinations that can be computed recursively. Here is the mathematical definition of how to compute the value $C(n, k)$ for any positive integer n and non-negative integer $k \leq n$:

$$C(n,k) = 1, \text{ if } k=0 \text{ or } k=n \\ C(n-1,k-1)+C(n-1,k), \text{ otherwise.}$$

Write a recursive method to compute $C(n,k)$. You may assume that the n passed to this method is positive and $0 \leq k \leq n$. The prototype is given for you below:

```
public static void choose(int n, int k) {
```

```
}
```

(9 pts) Solve the following recurrence relations using the formula given in the textbook:

a) $T(n) = 5T(n/2) + O(n^2)$

$T(n) =$ _____

b) $T(n) = 8T(n/3) + O(n^2)$

$T(n) =$ _____

c) $T(n) = 32T(n/4) + O(n^{2.5})$

$T(n) =$ _____

Consider the following recursive method. What does the call somerec(4,1,2) return? For bonus points, what does this method do in general?

```
public static int somerec(int n, int x, int y) {  
    if (n < x*y)  
        return 0;  
    else if (n == x*y)  
        return 1;  
    else {  
        int sum = 0;  
        for (i=x; i<=n;i++)  
            sum += somerec(n-i,i,y-1);  
        return sum;  
    }  
}
```

In a mergesort of 16 numbers, the last Merge call merges these two lists.

2, 6, 8, 10, 13, 17, 18, 20 followed by the list

1, 4, 5, 9, 14, 15, 19, 25

Which of the following (it is possible that more than one are true) is true about this particular merge sort?

- A) The original list was sorted.**
- B) The original list was not sorted.**
- C) The first number of the original list was not 9.**
- D) There were exactly 15 calls to the Merge method in this Merge Sort.**

What is the minimum number of calls to the partition method in a quick sort of 7 values?