CMPS 2200 Introduction to Algorithms – Fall 17

9/27/17

5. Homework

Due **10/4/17** at the beginning of class

Remember, you are allowed to turn in homeworks in groups of two. One writeup, with two names.

1. **3-way Mergesort (8 points)**
   Consider the following variant of mergesort, where the first call is `3wayMergesort(0,n-1,A)` to sort the array $A[0..n-1]$.

```
int 3wayMergesort(int i, int j, int[] A){
  // Sort A[i..j]
  if(j-i>=2){ // At least 2 elements in A
    int l = (j-i)/3;
    3wayMergesort(i,i+l, A);
    3wayMergesort(i+l+1,i+2*l,A);
    3wayMergesort(i+2*l+1,j,A);
    merge(i,i+l+1,i+2*l+1); // Merges all three sub-arrays in linear time
  }
}
```

   (a) Set up a runtime recurrence for `3-way mergesort` above.

   (b) Use the recursion tree method to find a good guess for the runtime of `3wayMergesort`.

2. **Recursion Tree (8 points)**
   Consider the recurrence $T(n) = 2T(n/5) + n^3$; assume $T(1) = 1$.

   (a) Use the recursion tree method to find a good guess of what $T(n)$ could solve to asymptotically.

   (b) Use big-Oh induction to formally verify your guess.

3. **Divide and Conquer (6 points)**
   Let $A[0..n-1]$ be an array of n numbers. A number in A is a *majority element* if A contains this number at least $\lfloor n/2 \rfloor + 1$ times. So for example, the array $A = \{4, 3, 2, 4, 1, 3, 3, 6, 3, 3, 3, 3\}$ has 3 as its majority element.

   Write a divide-and-conquer algorithm that determines whether a given array $A[0..n-1]$ contains a majority element, and if so, returns it. Your algorithm should run in $O(n \log n)$ time. You are **not** allowed to simply sort the array.

   Set up and solve a runtime recurrence for your algorithm.

   Hint: Start by applying the generic divide-and-conquer approach. Try to divide by two. Then try to combine the results. From the given runtime you should be able to guess how much time you are allowed to spend for dividing and combining.