O COMP 215 Algorithms Θ

Assignment #3

Hand in your answers written (neatly please) on paper on Wednesday, November 19 at the *beginning* of class.

1. In the textbook, problems 13 from Section 4.5
2. The following is inspired from problem 2 from Section 5.1, but significantly expanded. It will show how small design details can have an important influence on the running time of an algorithm
   We want to write a function that will find both the maximum value and the minimum value in an array. For simplicity, we assume that it is possible to write the return statement of this function as something like: "`return (min, max);`".
   (a) Write function MinMax1(A,n) to simultaneously find the min value and max value of an array. Your function should use a single loop that goes through all the elements of the array. Analyse the number of comparisons made by this function.
   (b) Write a function MinMax2(A,left,right) that uses a divide and conquer strategy. The initial call to the function should be `MinMax2(A,0,n-1)`, and the base case of the function should be that when `left` = `right`, the function returns (`A[left]`, `A[right]`). (when dividing the problem into 2 subcases, use a strategy similar to what we did for binary search, except that you will need to do a recursive call on both halves). Find the recurrence describing the number of comparisons made by this function and solve it assuming that $n = 2^k$. Is this better or worse than the brute force algorithm?
   (c) Write a function MinMax3(A,left,right) that uses a divide and conquer strategy. The initial call to the function should be `MinMax3(A,0,n-1)`, and the base case of the function should be that when `left` $\geq$ `right` $- 1$, the function returns (`A[left]`, `A[right]`) when `A[left]` $<$ `A[right]`, and the function returns (`A[right]`, `A[left]`) otherwise. (when dividing the problem into 2 subcases, use a strategy similar to what we did for binary search, except that you will need to do a recursive call on both halves). Find the recurrence describing the number of comparisons made by this function and solve it assuming that $n = 2^k$. Is this better or worse than the brute force algorithm?
3. In the textbook, problem 5 from section 5.1 (use the Master Theorem)
4. In the textbook, problem 2 from section 5.3 (if you are not certain, implement it and see what it does)
5. In the textbook, problem 2 from section 6.1