Quiz Submission

Raghad Abujabal

Student ID: 385605

Question 1

1. SuperSort(A)
sorts array A recursively

2. // base cases

3. if (length(A) == 1) return;

4. if (length(A) == 2)

5. swap the two elements if they're out of order;

6. return;

7. else // recursive calls

8. SuperSort the first two thirds of A;

9. SuperSort the second two thirds of A;

10. SuperSort the first two thirds of A again;

11.

12.

Answer:
so if length a = 1, we will return, if it's 2, we will swap elements(base), and then we will do 3 recursive calls on the first 2/3 of a, then 2nd, then 3rd.
therefore, Our recursive solution would be T(n)=3T(2n/3)+O(n).

Question 2

Part B: Give a recurrence relation upper bound, tight up to constant factors, for the performance of your algorithm given in Part A. Don't forget the base case(s).

Answer:
T(n)=T(k/2)+O(1)

Question 3

Part C: Solve your recurrence relation given in Part B by providing an asymptotic solution tight up to constant factors, but do NOT use the master theorem or the master-master theorem/nuclear bomb. Show your work.

Answer:
the solution would just be adding the constant factors,

Question 4

Suppose you are given positive integers where for nonnegative integer . You would like to determine using only the elementary operations of addition, subtraction, multiplication, and division.

A brute force algorithm is described below.

The brute force algorithm requires time to find the correct value of . To help with this task, your friend designed a function called which takes input of the form and computes in time. In Part A, you will use to design a divide and conquer algorithm to find the correct value of with smaller asymptotic runtime than the above brute force algorithm. Hint: , so such that if is even or if is odd.

Part A: Design a divide and conquer algorithm that has an asymptotically faster performance than the given brute force algorithm.

Answer:
base : if a =1, then k= 0. our design is, If let's say that we have a variable n and n = sqrta so the means that n = b^k/ 2. we want to decide if k is even or odd, if it's even then n would be (b^k/2) so that makes n^2 then k= 2(k/2) then if it's odd then a = b multiplied by b*(b^k/2)^2 and that would make k= 2(k/2)+ 1. then the runtime would be O(log k)