Question 3. [Master Theorem; 18 pts.]

Given an equation of the form

$$T(n) = aT(n/b) + f(n) \tag{1}$$

the Master Theorem states that, if $f(n) \in \Theta(n^d), d \geq 0$, then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

a. Explain why the parameter d does not contribute to the rate of growth of $T(n)$ in the third case ($T(n) \in \Theta(n^{\log_b a})$).

3 points

~~[illegible]~~ $f(n) \in \Theta(n^d)$ represents the cost of splitting and recombining the parts. If $a > b^d$, the cost of computing the algorithm on the leaves dominates the total cost, so, in figuring Θ, we can ignore the cost of splitting + recombining

b. You are analyzing the cost C of a new recursive algorithm. At each stage, the data is broken into 3 roughly equal parts, and the algorithm calls itself recursively on at most 2 of the 3 parts. The results of the recursive calls are then combined; the cost of this recombination is $2n$, where n is the size of each of the parts. The cost of running the algorithm on input of size 1 is 1.

note!

Write a recurrence relation for $C(n)$:

3 points

$C(n) = 2C\left(\frac{n}{3}\right) + 2\left(\frac{n}{3}\right)$, $C(1) = 1$

c. Use the Master Theorem to give Θ bounds on the rate of growth of function $C(n)$ from part (*b*). State which case of the Master Theorem applies.

$a = 2$ $\quad b = 3$ $\quad d = 1$ $\quad$ *Case :* $a < b^d$

3 points

$C(n) \in \Theta(n)$

d. Solve your recurrence relation from part (*b*) exactly, for $n = 3^k$, where $k \geq 0$

6 points

e. Use the Master Theorem to give Θ bounds on the solution to the recurrence $T(n) = 8T(\frac{n}{2}) + n$.

$a =$ 8 $\quad b =$ 2 $\quad d =$ 1

$C(n) \in \Theta(n^{\lg 8}) = \Theta(n^3)$ $\quad$ Case $a > b^d$

3 points

Question 4. **[Divide and Conquer; 10 pts]**

a. Write an algorithm in pseudocode for a *recursive* version of binary search on an array A. Use a 3-way comparison function COMPARE(a, b) that answers LESS when $a < b$, EQUAL when $a = b$, and GREATER when $a > b$. (Do not use $<$ or $>$.) Here is the heading for the search procedure:

```
procedure BinarySearch(A[l..r], K)
// Implements binary search recursively.
// Input: A sorted (sub)array A[l..r], where l ≥ 0 is the lower bound and
// r ≥ (l − 1) is the upper bound of the sub-array to be searched, and a key K.
// Output: An index i, l ≤ i ≤ r such that A[i] = K, or −1 if there is no such index.
```

```
~~mid~~ if l ≥ r then
    if COMPARE(A[l], K) = EQUAL then return l
    else return −1
else
    mid ← ⌊l+r/2⌋
    case COMPARE(K, A[mid])
        when LESS return BINARY SEARCH(A[l, mid−1], K)
        when GREATER return BINARY SEARCH(A[mid+1, r], K)
        when EQUAL return mid
```

7 points

b. Write a recurrence for the number of calls to COMPARE made by your algorithm.

$C(1) = 1$

$C(n) = C(\frac{n}{2}) + 1$

3 points