Name: Key

**1. (6 points)** The following recursive function sorts the elements of $A[p\,..\,r]$:

```
RECURSIVE-SORT(A, p, r)
  if p < r
      for i = r - 1 downto p
          if A[i] > A[r]
              Exchange A[i] and A[r]
      RECURSIVE-SORT(A, p, r - 1)
```

($a$) Derive a recurrence relation for the running-time $T(n)$ of RECURSIVE-SORT, where $n$ is the length of the input array (so $n = r - p + 1$).

($b$) Use a recursion tree to 'guess' an asymptotic upper bound for the recurrence relation.

($c$) Use the substitution method to prove your guess is correct.

(a) $T(n) = T(n-1) + \Theta(n)$

One recursive call of size $n-1$. Loop over $n-1$ elements of array.

(b)

$cn$

|

$c(n-1)$

|

$c(n-2)$

⋮

$c \cdot 1$

$$c\sum_{i=1}^{n} i = c\frac{n(n+1)}{2}$$

so $O(n^2)$

(c) Suppose $T(k) \le ck^2$ for $k < n$. Then

$$T(n) = T(n-1) + \Theta(n)$$
$$\le c(n-1)^2 + dn \quad \left(\text{for some } d>0\right)$$
$$= cn^2 - 2cn + c + dn$$
$$= cn^2 - n(2c-d) + c$$
$$\le cn^2 \text{ for } n \text{ sufficiently large and } c \ge \tfrac{d}{2}.$$

[Base Case: For any finite $n_0$, $T(1), T(2), \ldots, T(n_0)$ are bounded by a constant $c'$, and thus by $c'n^2$. If necessary, we can choose $c \ge c'$.]

(continued on other side)

**2. (4 points)** The following recursive function computes the sum of the elements in $A[p\,..\,r]$ where $A$ is a numeric array:

```
RECURSIVE-SUM(A, p, r)
if p == r
    return A[p]
else
    q = ⌊(p + r)/2⌋
    x = RECURSIVE-SUM(A, p, q)
    y = RECURSIVE-SUM(A, q + 1, r)
    return x + y
```

($a$) Derive a recurrence relation for the running-time $T(n)$ of RECURSIVE-SUM on an $n$-long input array.

($b$) Solve the recurrence relation to determine the running time $T(n)$.

(a) $T(n) = 2T(n/2) + \Theta(1)$
Two recursive calls of size $n/2$; non-recursive work is $\Theta(1)$.

(b) $a = b = 2$, $\log_b a = \lg 2 = 1$ so $n^{\log_b a} = n^1$. Since $f(n) = \Theta(1)$, $f(n) = O(n^{1-\varepsilon})$ with small $\varepsilon > 0$, say $\varepsilon = 0.5$.
By MT (case 1), $T(n) = \Theta(n)$.

**Theorem** (Master Theorem). *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence*

$$T(n) = aT(n/b) + f(n)$$

*where we interpret $n/b$ to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:*

1. *If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.*
2. *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.*
3. *If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.*

Name: Key

**1. (6 points)** The following recursive function computes the minimum value in an array $x$ of length $x.length$:

```
RECURSIVE-MIN(x)
n = x.length
if n == 1
    return x[1]
else
    p = RECURSIVE-MIN(x[2..n])
    return MIN(x[1], p)        // two-argument MIN() function
```

(*a*) Derive a recursion for the running-time of RECURSIVE-MIN.

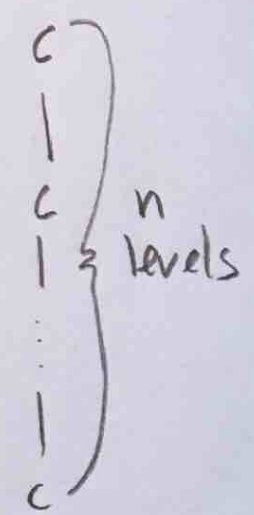
(*b*) Use a recursion tree to 'guess' an asymptotic bound for the recursion.

(*c*) Use the substitution method to prove your guess is correct.

(a) $T(n) = T(n-1) + \Theta(1)$

There is a single recursive call of size $n-1$. The non-recursive work is $\Theta(1)$.

(b)




$T(n) = O(n)$

(c) Suppose that $T(k) \le ck$ for some positive constant $c$ and $k < n$. Then

$$
\begin{aligned}
T(n) &= T(n-1) + \Theta(1) \\
&\le T(n-1) + d \quad (\text{for some } d > 0) \\
&\le c(n-1) + d \\
&= cn - c + d \\
&\le cn \quad \text{so long as } c \ge d.
\end{aligned}
$$

[Base Case: for any $n_0$, $T(1), T(2), \ldots, T(n_0)$ are bounded by a constant $c'$, and thus by $c'n$. We may require that $c \ge c'$.]

(continued on other side)

**2. (4 points)** Consider the pseudocode for MERGE-SORT:

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, P, Q, R)
```

(*a*) Derive a recurrence relation for the running-time $T(n)$ of MERGE-SORT on an $n$-long input array. You may assume that the MERGE function has $\Theta(n)$ running time.

(*b*) Solve the recurrence relation to determine the running time $T(n)$.

(a) $T(n) = 2T(n/2) + \Theta(n)$

(b) $a = b = 2 \quad \log_b a = \lg 2 = 1$

$n^{\log_b a} = n$

Since $f(n) = \Theta(n)$, $f(n) = \Theta(n^{\log_b a}) = \Theta(n)$.

By MT (case 2),

$T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n \lg n)$.

**Theorem** (Master Theorem). *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence*

$$T(n) = aT(n/b) + f(n)$$

*where we interpret $n/b$ to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:*

1. *If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.*
2. *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.*
3. *If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.*