Problem 1. [Analysis] (25 points)

A CS 141/218 student has been trying to speed-up Karatsuba’s divide-and-conquer integer multiplication algorithm. Given two numbers $x, y$ with $n$ bits each, his algorithm (1) first divides both $x$ and $y$ into four equal-length pieces, then (2) expresses the product $x \cdot y$ using $p$ multiplications of these $n/4$-bit pieces, followed by a constant number of additions, subtractions and shifts. How small $p$ needs to be in order to give a faster algorithm than the Karatsuba’s algorithm covered in class? You can assume $n$ to be a power of 4, and $p > 4$. Justify your answer.

Solution: The divide and conquer algorithm has the following recurrence relation:

$$T(n) = p \cdot T(n/4) + O(n)$$

We have to find the biggest possible $p$ such that this algorithm is asymptotically better than $O(n^{\log_3 2})$. We have $n \in O(n^{\log_4 p - \epsilon})$ when $\epsilon = \log_4 p - 1$ (we get a tight bound in this case). We have $\epsilon > 0$ because $p > 4$. By MT case I, it follows that

$$T(n) \in O(n^{\log_4 p})$$

We need a $p$ such that

$$\log_4 p < \log_2 3$$

Since $\log_2 3 = \log_4 3 / \log_4 2 = 2 \log_4 3 = \log_4 9$, this is equivalent to $p < 9$. So to get a faster algorithm, $p$ must be not bigger than 8.

Problem 2. [Amortized Analysis] (25 points)

Answer:

We use the accounting method for the amortized analysis. We use dollars to account for the number of times we store or copy an element. We charge 3 dollars for one $A.push\_back(x)$. One dollar pays for the actual insertion of $x$, and two dollars are saved in the array; one is stored in $x$ and pays for moving $x$ when the table is expanded, and one pays for moving another item that has already been moved once when the array was expanded. The credit invariant is that at any time just before a table expansion, each element in the array has at least one dollars on it (which can be used to pay for the copy operation).

For example, suppose that the size of $A$ is $m$ immediately after an expansion. Then, the number of items in the table is $m/2$ and the table contains no credit. Each time one $A.push\_back(x)$ is performed, we charge 3 dollars. The insertion of $x$ in $A$ is paid with one dollar. Another dollar is stored in $x$. The third dollar is placed as credit on one of the $m/2$ items already in the table. Filling the table requires $m/2 - 1$ additional insertions, and thus by the time the table contains $m$ items and is full, each item in $A$ has a dollar to pay for its copy during the next expansion.

In conclusion, the total cost for $n$ $push\_back$ operations on an initialized array $A$ is $3n$.

Incidentally, this problem is discussed at length in section 17.4 of the book.

Problem 3. [Divide and Conquer - Design] (25 points)

1. Partition the set of $k$ arrays into $k/2$ pairs
2. Merge each pair of arrays using the linear-time merge procedure from MergeSort
3. Recurse on the remaining set composed of $k/2$ arrays
The recursion is $T(k) = T(k/2) + k\Theta(n)$ which has solution $O(nk\log k)$. This is essentially doing the top $k$ levels of the recursion tree from MergeSort on a partially sorted $kn$-element array.

Neal:
1. Divide the lists into two sets of $k/2$ lists, each of size $n$.
2. Recursively merge the first set of lists, giving a list of size $nk/2$.
3. Recursively merge the second set of lists, giving another list of size $nk/2$.
4. Merge the two resulting lists (of size $nk/2$ each) to get the result.

The total time $T(n,k)$ satisfies the recurrence $T(n,k) = 2T(n,k/2) + nk$, and $T(n,1) = 1$.

In the recursive calls, $n$ does not change.

At the $i$th level in the recursion tree, there are $2^i$ subproblems, each with a set of $k/2^i$ lists, each taking time $nk/2^i$. The total time at level $i$ is $2^i nk/2^i = nk$.

There are $\log_2 k$ levels. Thus, the total time is $O(nk\log k)$.

**Problem 4.** [Divide and Conquer - Analysis] (25 points)

In the algorithm Select described in class (linear time selection), the input elements are divided in groups of 5. Write the recurrence relation for the time complexity of Select if you decided to divide the input in groups of 7. You do NOT need to solve the recurrence relation.

**Answer:** For groups of 7, the number of elements greater than $x$ (and the number of element less than $x$) is at least

$$4\left(\left\lfloor \frac{n}{7} \right\rfloor - 2\right) \geq \frac{2n}{7} - 8$$

and the recurrence relation becomes

$$T_7(n) = \begin{cases} 
\Theta(1) & n < 70 \\
T_7\left(\left\lceil n/7 \right\rceil\right) + T_7(5n/7 + 8) + O(n) & n \geq 70
\end{cases}$$