CSCI270 Week 5

Jacob Ma

October 12, 2022

1 Divide & Conquer

High-level idea:

- Take your input of size n
- Divide into one or more input of smaller size
- Solve smaller inputs recursively (For small enough, solve directly)
- Then combine the solutions of the smaller problems into a solution of the whole problem

Example 1.1: Merge Sort Algorithm.

```
MergeSort(a[],int L, int R)
  if R==L then do nothing
  else
    Let $m = \frac{R+L}{2}$, rounded down
    MergeSort(a,L,m)
    MergeSort(a,m+1,R)
    Merge (a,L,m,R)
  end if
Merge(a[],int L, int m, int R)
  array $b$ to copy into [0-indexed]
  int i=L, j=m+1, k=0;
  while i<=m | j <=R do
    if (j>R | (i<=m && a[i]<=a[j])) then
      b[k] = a[i]; i++; k++;
    else
      b[k]=a[j];j++;k++;
    end if
  end while
  for i=L; i<=R; i++ do
    a[i] = b[i-k]
```

```
end for
```

Lemma 1.1.1: Merge

Merge takes two sorted array and puts them into one array. If the input array a[L...m] and a[m+1,...R] are sorted, the final output is sorted.

Proof. Skipped. Formulate the right loop invariant, and prove by induction on j or on $i+j$. □

Theorem 1.2

Merge Sort is correct. If you call **MergSort(a[],L,R)**, it terminates, and afterwards, a[L...R]is sorted, and still contains the same elements.

Proof. Idea: The algorithm terminates, and the output is sorted, so a[i]<=a[i+1] for all i. We prove byy induction on $n = R-L+1$, which is the size of the array we're working on. Note: Correctness proofs for basically all recursion need induction, because the calls for bigger inputs require correctly solving the smaller inputs.

Base case: $n = 1 \implies R = L$. The algorithm does nothing, correctly sorts one element.

Induction Hypothesis: Assume correct for size $1, 2, \ldots, n-1$. (We use strong induction)

Induction Step: (array size n). For recursive call:

$$m+1-L = \frac{L+R}{2}+1-L < R+1-L = n$$
$$R+1-m = R-\frac{L+R}{2} < R+1-L (\text{True if } R > L)$$

Because both subarrays are smaller, we can use the induction hypothesis on them, so both calls terminate, and the results are sorted and contain the same elements as before. Then apply Merge lemma, which proves that the final output a[L..R] is sorted and contains the same elements. □

2 Master Theorem

Theorem 2.1: Master Theorem

Let $a \geqslant 1, b > 1$, and $T(n)$ is defined by $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, $T(1) = \Theta(1)$. (Ignore rounding of $\frac{n}{b}$). Then:

(1) If $f(n) = O\left(n^{\log_b a-\epsilon}\right)$ for some $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$

(2) If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

(3) If $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ for some $\epsilon > 0$, and $a \cdot f\left(\frac{n}{b}\right) \leqslant c \cdot f(n)$ for some $c < 1$ and sufficiently large n, then $T(n) = \Theta(f(n))$.

Example 2.2: Integer Multiplication. The multiplication of two binary numbers of size m and n, would take $O(mn)$ runtime. We can do it faster with divide and conquer.

For the first number x, it could be divided into

$$x = x^+ \cdot 2^{\frac{n}{2}} + x^- \qquad \text{where + and - equally partitions } x$$

Same for y. Thus,

$$\begin{aligned}(x \cdot y) &= \left(x^+ 2^{\frac{n}{2}} + x^-\right) \cdot \left(y^+ 2^{\frac{n}{2}} + y^-\right) \\ &= x^+ y^+ \cdot 2^n + x^- y^+ 2^{\frac{n}{2}} + x^+ y^- 2^{\frac{n}{2}} + x^- y^-\end{aligned}$$

If our recursive calls complete these four, then the formula gave us $x \cdot y$.

Proof. Let's see the running time: 4 recursive calls, with each size $\frac{n}{2}$. Additional time for shifting and adding the results: $O(n)$.
Recurrence: $T(n) = 4T\left(\frac{n}{2}\right) + O(n)$
Apply Master Theorem: $f(n) = O(n), a = 4, b = 2 \implies \log_b a = 2$. $f(n)$ vs. $n^{\log_b a} = n^2$, we get $f(n) = O\left(n^2\right)$, thus case 1.
Master Theorem Case (1) : $T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^2\right)$. So no improvement so far.

$$x \cdot y = x^+ y^+ \cdot 2^n + \underbrace{\left(x^- y^+ + x^+ y^-\right)}_{(x^+ + x^-)(y^+ y^-) - x^+ y^+ - x^- y^-} \cdot 2^{\frac{n}{2}} + x^- y^-$$

Rule: Compute $a = x^+ y^+, b = x^- y^-, c = (x^- + x^+)(y^+ + y^-)$
Then compute $a \cdot 2^n + (c - a - b) \cdot 2^{\frac{n}{2}} + b$
We just showed that this gives correct answer. Time for computations: $\Theta(n)$.
Recurrence: $T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$.

$$\log_b(a) = \log_2(3) \approx 1.59$$

Master Theorem: Case (1) : $T(n) = \Theta\left(n^{1.59}\right)$ □

Example 2.3: Some more Master Theorem Practice.

(1)

$$T(n) = aT\left(\frac{n}{3}\right) + n \log n, \qquad a = 9, b = 3$$

$f(n)$ gives faster than $n^{\log_a b}$ (eg. $\epsilon = 0.5$).

Case 1: $T(n) = \Theta n^2$

(2)

$$T(n) = 2T\left(\frac{n}{2}\right) + c \cdot n, \qquad a = 2, b = 2$$

$\log_b(a) = 1$. $f(n)$ grows equally fast as $n^{\log_b a}$.

Case 2: $T(n) = \Theta \log n \log n$

(3)

$$T\left(n\right) = 3T\left(\frac{n}{4}\right) + k \cdot n, \qquad a = 3, n = 4$$

$\log_b\left(a\right) < 1$

Case 3: Verify $a \cdot f\left(\frac{n}{b}\right) \leqslant c \cdot f\left(n\right)$ for some $c < 1$. $\forall$