Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

CSC373S: Algorithm Design, Analysis & Complexity

Lecture 02

Monday January 09, 2017

based on notes by Denis Pankratov

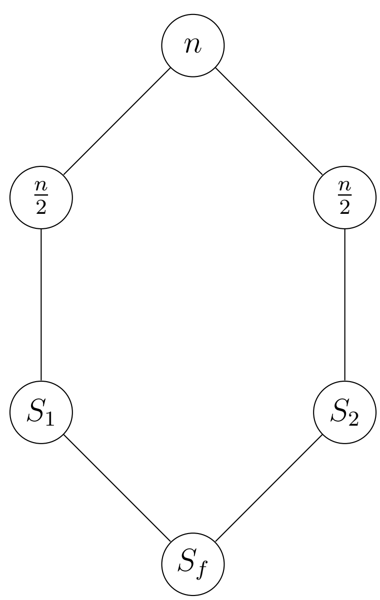
Divide & Conquer

To solve an instance of size n

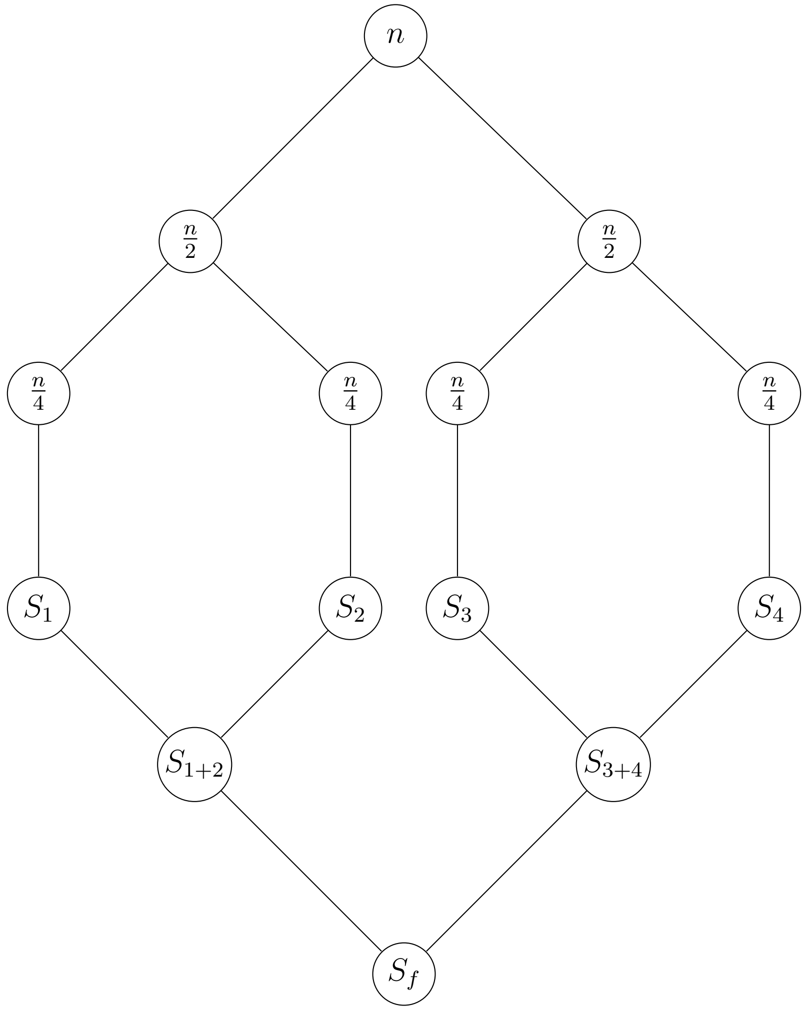
1. Divide it into two (or *more*) subinstances of smaller size (e.g. $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$)
2. Solve subinstances recursively
3. Combine solutions from (2) to get a solution for the original instance

This is a visual representation of the concept:

Dividing once:



Dividing twice:



Where we first divide into fractions of n, then solve these parts individually. The combination of partial solutions makes our final solution S_f.

However, this approach is only beneficial if the stack of operations is not too big. If this were the case, then this strategy would offer no computational advantage.

Input: Array A of n numbers (indexed from 1), $p, r \in [n] := 1, 2, \ldots, n$
Output: $A[p..r]$ contains elements from $A[p..r]$ in increasing order & other elements are left unchanged

Algorithm:

```
def MergeSort(A, p, r):
   if p < r:
      q = floor((p + r)/2)
      MergeSort(A, p, q)
      MergeSort(A, q + 1, r)
      Merge(A, p, q, r)
```

When does it stop? What is the base case?

Execution:

```
Merge(A, p, q, r)
  init array L of size q − p + 2
  init array R of size r − q + 1

  for i = 1 to q − p + 1:
    L[i] = A[p + i − 1]

  for j = 1 to r − q:
    R[j] = A[q + j]

  L[q − p + 2], R[r − q + 1] = R[r − q + 1], float('inf')
  i = j = 1

  # of interest for loop invariant (LI)
  for k = p to r:
    if L[i] < R[j]:
      A[k] = L[i]
      i++
    else:
      A[k] = R[j]
      j++
```

Loop Invariant:

1. $A[p..k-1]$ contains $k-p$ smallest elements from L, R in increasing order, and $L[i], R[j]$ are the smallest elements in L, R, respectively, not in $A[p..k-1]$.
2. MergeSort is correct by trivial induction and part (1)

Measure of Interest: # of comparisons of array elements (*i.e.* $L[i] < R[j]$ is of true interest to us since its computation will dictate runtime.

$T(n)$ = worst case # of comparisons performed by MergeSort on inputs of length r.

$$T(n) = \begin{cases} 0, & \text{for } n \le 1 \ (\textit{base case}) \\ 2T(n/2) + \mathcal{O}(n), & \text{for } n > 1 \end{cases}$$

By the Master Theorem (*see below*), this solves to $\mathcal{O}(n \log n)$, where $n = r - p + 1$.

Integer Multiplication

Input: X, Y n-digit (decimal) integers
Output: $X \cdot Y$

Most computers can handle $n = 19$ digits, so what do we do if we want more digits?

Measure of Interest: single digit operations

Observation:

1. Base (10, 2, etc) does not matter as long as it is *constant*
2. Elementary school algorithm costs $\mathcal{O}(n^2)$, but *Karatsuba's Algorithm* (1960) does better
3. Write $X = 10^{n/2}X_1 + X_2, Y = 10^{n/2}Y_1 + Y_2$, so that:

$$\begin{aligned} X \cdot Y &= (10^{n/2}X_1 + X_2)(10^{n/2}Y_1 + Y_2) \\ &= 10^n X_1 Y_1 + 10^{n/2}(X_1Y_2 + X_2Y_1) + X_2Y_2 \end{aligned}$$

4. Karatsuba's trick: compute $X_1Y_1, X_2Y_2, (X_1+X_2)(Y_1+Y_2)$, so that $X_1Y_1+(X_1Y_2+X_2Y_1)+X_2Y_2$. From this, the middle part was computed to be Z, which can be evaluated the following way:

$$\begin{aligned} Z &= (X_1Y_2 + X_2Y_1) \\ &= (X_1 + X_2) \cdot (Y_1 + Y_2) - X_1Y_1 - X_2Y_2 \end{aligned}$$

Z can be computed using only 3 *recursive calls*!

Algorithm:

```
def Mult(X, Y, n):
  if n = 1:
    return X*Y
  else:
    # Following two lines are pseudo Python
    X = (10 ** (n/2)) * X_1 + X_2
    Y = (10 ** (n/2)) * Y_1 + Y_2
    A = Mult(X_1, Y_1)
    B = Mult(X_2, Y_2)

    # T_1, T_2 are only n/2 digits long
    T_1 = X_1 + X_2
    T_2 = Y_1 + Y_2

    # Takes only 3 recursive calls!
    C = Mult(T_1, T_2, n/2)

  return (10**n) * A + (10 ** (n/2)) (C - A - B) + B
```

Let $T(n) = \#$ of single digit operations performed by multiplication on inputs of length n.

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ 3T(n/2) + \mathcal{O}(n), & \text{if } n > 1 \end{cases}$$

By Master Theorem, this solves to $\mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.56})$.

Review: Master Theorem

Let $T : \mathbb{N} \to \mathbb{R}^+$ be a recursively defined function with recurrence relation

$$T(n) = cT\left(\frac{n}{d}\right) + f(n)$$

for some constants $c, d \in \mathbb{Z}^+, d > 1$, and $f : \mathbb{N} \to \mathbb{R}^+$.

Furthermore, suppose $f(n) \in \Theta(n^k)$ for some $k \in \mathbb{R}, k \geq 0$. Then:

1. if $k = \log_d c$, then $T(n) \in \mathcal{O}(n^k \log n)$;
2. if $k < \log_d c$, then $T(n) \in \mathcal{O}(n^{\log_d c})$;
3. if $k > \log_d c$, then $T(n) \in \mathcal{O}(n^k)$;