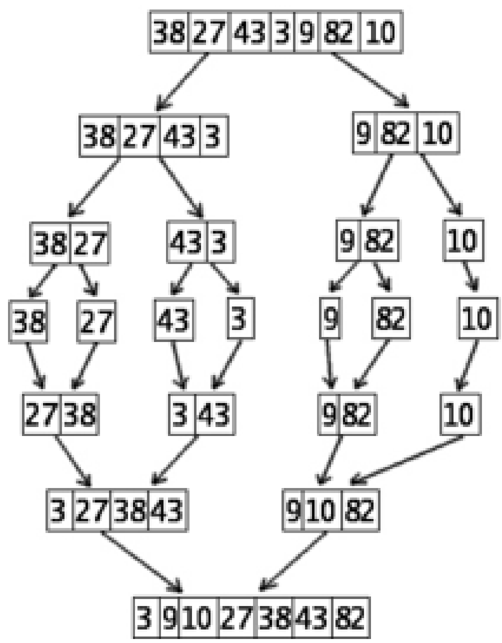
Computer Science 577 Notes
Introduction to Algorithms

Mendel C. Mayr

January 28, 2015



Contents

1 Recurrence Relations and Analysis of Algorithms

1.1 Recurrence Relations

1.1.1 Recursive Analysis of Insertion and Merge Sort

Insertion sort: let $M(n)$ be the comparisons required to sort a list of size n

Analysis: note that $M(1) = 0$ and $M(n) = M(n-1) + n$ for $n > 1$

(i) $M(n) = M(n-1) + n$

(ii) $M(n) = M(n-2) + n + (n-1)$ ...

(iii) $M(n) = M(n-k) + n + (n-1) + \ ... \ + (n-k+1)$

(iv) Let $k = n-1$, $M(n) = M(1) + n(n-n+1) + \sum_{i=1}^{n-1} i$

(v) $M(n) = 0 + n + (n-1)(n-2)/2 \approx n^2/2$

Merge sort: let $M(n)$ be the comparison required to sort a list of size n

Analysis: for simplcity, consider only n such that $n = 2^a$ for some integer a
Note that $M(1) = 0$ and $M(n) = 2M(n/2) + n$ for $n > 1$

(i) $M(n) = 2M(n/2) + n$

(ii) $M(n) = 2M(n/4) + n + (n/2)$

(iii) $M(n) = 2M(n/2^k) + n + (n/2) + \ .... \ + (n/2^{k-1})$

(iv) Let $k = a$, $2M(1) + \sum_{i=1}^{k-1} n/2^i$ (unclarified point)

(v) Mergsort is $O(n \log n)$

1.1.2 Recursive Linear Selection

Recursive linear selection algorithm: given $x_1, x_2, ..., x_n$ distinct keys, find x_k (i.e. the kth smallest element) without using sorting
Note: the rank of an element (i.e. the number of keys greater than it) can be found in linear time
Linear selection algorithm is as folows:

(i) Remove keys of known rank, to make $n = 5(mod 10)$

(ii) Divide elements into groups of 5, denoted $S[i]$ for i from 1 to $n/5$

(iii) Recursively find the median of each group, denoted $x[i]$

(iv) Let M^* be the median of the set $x[i]$ for i from 1 to $n/5$

(v) Divide keys into groups of keys less than (call this L), equal to, or greater than (call this R) M^*

(vi) Recursiveley process one of L or R

Analysis: Note that steps 1, 2, and 5 are $O(n)$, so the number of computations for this algorithm, $T(n) = T(n/5) + T(7n/10) + O(n)$
Guessing and proving: supposed that $T(n) = O(n)$, which can be proven via strong induction, i.e. $T(n) < An$ for some constant A for all n
Recall that strong induction relies on proving two claims:

(i) The statement holds for all $n \geq 1$, $\forall n \leq n_0$ (base case)

(ii) If the statmenet holds for all $i < n$, it holds for n

Proof of second part of strong inductive proof:

(i) Suppose that $T(n) = O(n)$ for $i < n$

(ii) We seek and A such that $A(n/5) + A(7n/10) + cn \leq An$

(iii) Thus, $A \geq 10c$ is sufficient for this part

Proof of first part of strong inductive proof: need A such that $n(n-1)/2 \leq An$ for $1 \leq n \leq 10$
So $A \geq 9/10$ is sufficient for this part

Conclusion: $A = max\{9/2, 10c\}$ will suffice to show that $T(n) = O(n)$

1.1.3 Recursive Quadratic Closest-Pair

Recursive quadratic algorithm: find closest pair of points

i (Supposing that $n = 2^k$) into 2 equal groups, denoted L and R

ii Recurisvely find the closest pair in L and R

iii Report closest pair form testing elements of L against elements of R

iv Report best pair out of those from steps (ii) and (iii)

Analysis: $T(n) = 2(T/n) + O(n^2)$ for $n = 2^k \geq 4$, $T(2) = 1$
The $O(n^2)$ in the recursive case comes from step (iii)

Consider the recursion tree, which is full binary tree: at the first level, the problem size is $n, n/2, n/4, ...$ at the first, second, third, etc. levels. Thus, the number of computations required is n^2 at the first level, $2(n/2)^2 = n^2/2$ at the second level, $2(n/4)^2 = n^2/4$ at the third, etc.

Thus, the maximum number of computations is $\sum_{k=1}^{\infty} n^2/2^k = 2n^2$, thus $T(n) = O(n^2)$

1.1.4 Divide and Conquer Recurrences

Master theorem: if $T(n) = aT(n/b) + O(n^d)$ for some constants $a > 0, b > 1$, and $d \geq 0$, then:

(i) $T(n) = O(n^d)$ if $d > \log_b a$

(ii) $T(n) = O(n^d \log n)$ if $d = \log_b a$

(iii) $T(n) = O(n^{\log_b a})$ if $d < \log_b a$

Proof: consider the recursion tree for such a problem
Notice that a is the branching factor of the problem. At the ith level (starting at index 0), there are a^i subproblems of size n/b^i. which means the computation that must be done at that level is $a^i O((n/b^i)^d)$
The number of levels in the recusion tree is $k = \log_b n$
As such: $T(n) = \sum_{i=0}^{k} a_i O((n/b^i)^d) = O(n^d) \sum_{i=0}^{k} (a/b^d)^i$. Now consider the cases

(i) If $d > \log_b a$, then $a/b^d < 1$
$\sum_{i=0}^{\infty} (a/b^d)^i < \infty$ (i.e. series converges)
$\sum_{i=0}^{\infty} (a/b^d)^i = O(1)$, so $T(n) = O(n^d)$

(ii) If $d = \log_b a$, then $a/b^d = 1$
$\sum_{i=0}^{k}(a/b^d)^i = \sum_{i=0}^{k} 1 = k+1$, since $k = \log_b n = \log n / \log b = O(\log n)$
Therefore, $T(n) = O(n^d \log n)$

(iii) If $d < \log_b a$, then $a/b^d > 1$
$\sum_{i=0}^{k}(a/b^d)^i = O((a/b^d)^k)$, so $T(n) = O(n^d)O(a^k)/b^{dk}$
Since $k = \log_b n, n = b^k$, $T(n) = O(n^d)O(a^k)/n^d = O(a^k)$
$a^k = a^{\log_b n} = n^{\log_b a}$, so $T(n) = O(n^{\log_b a})$

A Logarithms