Introduction To Algorithms
HomeWork 4 Solutions

Junzhe Zheng

February 24, 2014

1.Give an example where quicksort requires $O(n^2)$ steps.

Consider a list:

$$10, 9, 8, 7, 6, 5, 4, 3, 2, 1$$

We choose the last digit in the last as the pivot.Thus time complexity is given as:

$$T(n) \ = \ T(n-1) + T(0) + \Theta(n)$$

By using substitution method, we could get:

$$T(n) \ = \ \Theta(n^2)$$

If it is $\Theta(n^2)$, it is also a $O(n^2)$.

2.Problem 4-6 (Page 110) CLRS(3rd Edition).

a. Need to prove "if and only if", thus the proof will have to separate parts

Proof of 'Only if':

If A is a Monge array, by definition, we have:

$$A[i,j] \ + A[k,l] \ < \ A[i,l] \ + A[k,j] \ \ \forall i \ , \ j \ , \ k \ , \ l$$

$$where \ 1 < i < k < n \ , \ 1 < j < l < m$$

Let $k = i + 1, \ l = j + 1$, we will have:

$$A[i,j] \ + A[i+1,j+1] \ < \ A[i,j+1] \ + A[i+1,j] \ \ \forall i \ , \ j$$

$$where \ 1 < i < i+1 < n \ , \ 1 < j < j+1 < m$$

$$where \ 1 < i < n-1 \ , \ 1 < j < m-1$$

‘Only if’ has been proved.

Proof of ‘if’:
Induction method will be used separately on rows and columns.
For columns:
Let us consider a 2×2 submatrix of A. This is a base case and is True because $k = i + 1, l = j + 1$.
For a $2 \times n - 1$ matrix A', assuming it is true.
If $j, l \in A'$, then by induction method, $\forall i, j, k, l,\ A[i, j] + A[i + 1, j + 1] \leq A[i, j + 1] + A[i + 1, j]$ is True.
If $j \in A'$ and $l \in A$, let j=n-1,l=n,k=i+1,therefore we have:

$$A[i, n-1] + A[i+1, n] \leq A[i, n] + A[i+1, n-1] \quad (1)$$

This is True for $A[i, j] + A[i + 1, j + 1] \leq A[i, j + 1] + A[i + 1, j]$.
Let j=i,l=n-1,k=i+1,we have:

$$A[i, i] + A[i+1, n-1] \leq A[i, n-1] + A[i+1, i] \quad (2)$$

Sum (1) and (2),we have:

$$A[i, i] + A[i+1, n] \leq A[i, n] + A[i+1, i]$$

which is equal to:

$$A[i, j] + A[k, n] \leq A[i, n] + A[k, i]$$

Thus we proved $2 \times n$ for columns.

For rows:
Let us consider a base case $2 \times n$ which is True from above.
Assume a submatrix A', $m - 1 \times n$, in matrix A is True.
Assume k=n-1,l=i,r=m-1,we have:

$$A[m-1, n-1] + A[m, i] \leq A[m, n-1] + A[m-1, i]$$

$$A[i, n-1] + A[m-1, i] \leq A[i, i] + A[m-1, n-1]$$

sum these two up:

$$A[i, n-1] + A[m, i] \leq A[i, i] + A[m, n-1]$$

which is equal to:

$$A[i, k] + A[m, l] \leq A[i, l] + A[m, k]$$

which proves $m \times n$.Thus A is a Monge array.

(b). If a Matrix is a Monge array, then it must follows:

$$\forall i,j \; A[i,j] \; + A[i+1,j+1] \; < \; A[i,j+1] \; + A[i+1,j]$$

$$A[1,2] + A[2,3] = 30 > A[1,3] + A[2,2] = 28$$

Just change A[1,3] to 25 (any number no less then 24).

(c). Assume $f(i) > f(i+1)$
Let $m = f(i), n = f(i+1)$
Thus A[i,m]+A[i+1,n] is the smallest sum in row i and i+1.
But for a monge array,$A[i,n] + A[i+1,m] \leq A[i,m] + A[i+1,n]$.
Cotradiction.Therefore,$f(i) < f(i+1)$.

(d). From part c, we know that the leftmost minimum of row i is not greater than the leftmost minimum of row i+1.Let i is odd,then $f(i-1) < f(i) < f(i+1)$.Assuming no two f_i, f_j are equal,for $i \neq j$, to find i, the cost is n.If there are some f(i),f(j) are equal, at most we need to compare m times between m rows, the cost is m.Thus the time complexity is $O(m+n)$.

(e).

$$\begin{aligned} T &= T(\frac{m}{2}) + O(m+n) && (1)\\ &= T(\frac{m}{4}) + O(\frac{m}{2}+n) + O(m+n) && (2)\\ &= ... && (3)\\ &= n + nlog_2m + 2m && (4)\\ &= O(m + nlogm) && (5) \end{aligned}$$

3.Using the version of heap sort as defined in CLRS(chapter 6-4),show an example where heapsort requires Ω(nlogn) steps.
For an array that each elements are already sorted in an increasing order, the performance of heapsort is $\Omega(nlg(n))$. Because that each of the n-1 calls of Max_HEAPIFY (for i= A.length downto 2)takes $\Omega(lg(n))$.

4.Consider radix sort with numbers (using base 10) that are variable length. Show that you can output any number as soon as you have considered all its digits. Design a method to sort in $O(n+k)$ time where k is the total number of digits in all the numbers.

For radix sort we start at checking the least digits. At the i^{th} digits sorting process, the number are ordered by the i^{th} least digits.

For a number with length j, once we finished checking j^{th} digits, we could output it in the right position in an sorted array.

To output a number, we need to check every number remain in to-be-sort list after l^{th} iteration. For a number with length i, if i=l, we output this number to sorted list. If i>l, then put this number to l+1 bucket for $l+1^{th}$ iteration.

There are n number and k is the total length over all n digits. Thus the time complexity is $O(n+k)$.

5.Suppose that you are given a sequence of n elements to sort. The input sequence consists of $\frac{n}{log\ n}$ subsequences, each containing $(logn)$ elements. The elements in a given subsequence are all smaller than the elements in the succeeding subsequence and larger than the elements in the preceding subsequence. Thus, all that is needed to sort the whole sequence of length n is to sort the $(logn)$ elements in each of the $\frac{n}{log\ n}$ subsequences. Show an $\Omega(nloglogn)$ lower bound on the number of comparisons $logn$ needed to solve this variant of the sorting problem. (Hint: It is not rigorous to simply combine the lower bounds for the individual subsequences.)

Consider each subsequence has $logn$ elements, and those elements can be in any order, so there are $(logn)!$ permutations. And there are $\frac{n}{logn}$ subsequence, thus total permutations is given by $((logn)!)^{\frac{n}{logn}}$.

This means a decision tree at least have $(logn!)^{\frac{n}{logn}}$ leaves. We also know any binary tree with height h at most have 2^h leaves. So $2^h \geq (logn!)^{\frac{n}{logn}}$,this lets to:

$$\begin{aligned} h &\geq \frac{n}{logn} lg((logn)!) && (6)\\ &\geq \frac{n}{logn} lg(\frac{logn}{2}^{\frac{logn}{2}}) && (7)\\ &= \frac{n}{2} lg(\frac{logn}{2}) && (8)\\ &= \frac{n}{2} lglogn - \frac{n}{2} lg2 && (9)\\ &\geq \frac{n}{2} loglogn - \frac{n}{2} && (10)\\ &= \Omega(nloglogn) && (11) \end{aligned}$$