CSCE 5150 – Analysis of Computer Algorithms

Homework No. 1 - Introduction & Growth of Functions

Neha Goud Baddam
11519516

Q1. Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in 8n 2 steps, while merge sort runs in 64nlg n steps. For which values of n does insertion sort beat merge sort? (Exercise 1.2-2)

To compare implementations of insertion sort and merge sort on the same machine., we need to compare the time complexities of insertion and merge sort

Solution 1: Insertion sort beats merge sort (for input size n), when $8n^2$ (insertion sort steps) is less than $64n\log n$ (merge sort steps)

Then $8n^2 < 64n \log n$

$\cancel{8}^{1} n^{\cancel{2}} < \cancel{64}^{8} \cancel{n} \log n$

$n < 8 \log n$

$\frac{n}{8} < \log_2 n$

$2^{n/8} < n$

Lets substitute 'n' values to balance the equation

→ Firstly lets assume that $n=8$, then $2^{8/8} < n$

When we keep substituting n values, we notice that for $n=43$, Insertion sort starts to beat merge sort

For $n=43$, $2^{43/8} = 42.4 < n$

$n=44$, $2^{44/8} = 44.8 > n$

Therefore for $n \leq 43$, insertion sort beats merge sort. & for $n \geq 44$ merge sort beats insertion sort

tion 2 :

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 1 - Introduction & Growth of Functions

Neha Goud Baddam
11519516

Q2. We can express insertion sort as a recursive procedure as follows. In order to sort A [1 .. n], we recursively sort A [1 .. n-1] and then insert A [n] into the sorted array A [1 .. n-1]. Write a recurrence T(n) for the running time of this recursive version of insertion sort. (Exercise 2.3-4)

Solution 2 :

Recursive procedure for insertion sort of A[1..n] :

Firstly, we sort A[1..n-1] recursively

Then, we insert A[n] into the above sorted array.

Initially for n=1, as array A[1..n-1] is initially empty, the algorith takes $\Theta(1)$ time to insert first element.

Then for n=2, it takes $(2-1) + \Theta(n)$ time for insertion

⋮

Similarly for $n \geq 1$, it takes n-1 recursions to sort array + $\Theta(n)$ for insertion into sorted array.

$$\text{Recurrence} = T(n) = \begin{cases} \Theta(n) & \text{if } n = 1 \\ T(n-1) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$$= \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Q3. Describe a Θ (n lg n) - time algorithm that, given a set S of n integers and another integer x, determines whether there exist two elements in S whose sum is exactly x. (Exercise 2.3-7)

we can sort the array with merge sort Θ (n log n) and then for each element (say y) in the array, we can do a binary search for (x – y) on the sorted array. So, the algorithm will run in Θ(nlog n).

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 1 - Introduction & Growth of Functions

Neha Goud Baddam
11519516

Solution 3 :

Given set S of n integers.
x, be an integer
Algorithm checks if there exists two elements in S whose sume is exactly x.
1. Firstly, we sort set S
2. For all values(y) in S, if S has $x - y$ then the numbes is foun

For above algorithm, sorting takes $O(n \log n)$ time, which loops n times performing binary search on each iteration in time $O(\log n)$
Therefore the algorithm takes $O(n \log n)$ to run
If we use merge sort, then the algorithm takes $O(n \log n)$

Q4. Sort all the functions below in increasing order of asymptotic (big-O) growth. If some have the same asymptotic growth, then be sure to indicate that. As usual, lg means base 2.

1. $5n$
2. n^4
3. $4 \lg n$
4. $n^{n/4}$
5. $n^{1/2}\log^4_n$

Answer: $4\lg n < 5n < n^4 < n^{1/2}\log^4_n < n^{n/4}$

Solution 4 : Time complexities for Big-O are in the following order
$O(\log n) < O(n) < O(n \log n) < O(n^c) < O(c^n) < O(n!)$
In increasing order → $4\lg n < 5n < n^4 < n^{1/2}\log^4_n < n^{n/4}$

CSCE 5150 – Analysis of Computer Algorithms

Homework No. 1 - Introduction & Growth of Functions

Neha Goud Baddam
11519516

Q5. Prove that 2 n+1 = O(2n). (Exercise 3.1-4) (Hint: Try to satisfy the definition of O-notation with some constants c, n0 > 0)

Solution 5 : Prove that $2^{n+1} = O(2^n)$

O-notation : $f(n) = O(g(n))$, if there exists constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le c g(n)$ for all $n \ge n_0$

$2^{n+1} = O(2^n)$

$2^{n+1} \le c \cdot 2^n$

If $c = 2$, $2^{n+1} \le 2 \cdot 2^n$

$c = 3$, $2^{n+1} \le 3 \cdot 2^n$

$c = 4$, $2^{n+1} \le 2^2 \cdot 2^n$

$\vdots$

$c = n$, $2^{n+1} \le n \cdot 2^n$

Hence $2^{n+1} = O(2^n)$ for all values of $C \ge 2$