CS11001/CS11002
Programming and Data Structures (PDS)
(Theory: 3-1-0)

Analysis of Algorithms

Sorting problem

- Input: A sequence of n numbers, a1,a2,…,an
- Output: A permutation (reordering) (a1’,a2’,…,an’) of the input sequence such that a1’≤a2’ ≤… ≤an’
  - Comment: The number that we wish to sort are also known as keys

Insertion Sort

- Efficient for sorting small numbers
- In place sort: Takes an array A[0..n-1] (sequence of n elements) and arranges them in place, so that it is sorted.

It is always good to start with numbers

5	2	4	6	1	3

1	2	3	4	5	6

j=5

Invariant property in the loop:

At the start of each iteration of the algorithm, the subarray a[0...j-1] contains the elements originally in a[0..j-1] but in sorted order

Pseudo Code

- **Insertion-sort(A)**

```
1.  for j=1 to (length(A)-1)
2.      do key = A[j]
3.  #Insert A[j] into the sorted sequnce A[0...j-1]
4.      i=j-1
5.       while i>0 and A[i]>key
6.          do A[i+1]=A[i]
7.                i=i-1
    8.        A[i+1]=key  //as A[i]<=key, so we place
                         //key on the right side of A[i]
```

Lets analyze the Insertion sort

- The time taken to sort depends on the fact that we are sorting how many numbers
- Also, the time to sort may change depending upon whether the array is almost sorted (can you see if the array was sorted we had very little job).
- So, we need to define the meaning of the **input size** and **running time**.

Input Size

- Depends on the notion of the problem we are studying.
- Consider sorting of n numbers. The input size is the cardinal number of the set of the integers we are sorting.
- Consider multiplying two integers. The input size is the total number of bits required to represent the numbers.
- Sometimes, instead of one numbers we represent the input by two numbers. E.g. graph algorithms, where the input size is represented by both the number of edges (E) and the number of vertices (V)

Running Time

- Proportional to the Number of primitive operations or steps performed.
- Assume, in the pseudo-code a constant amount of time is required for each line.
- Assume that the ith line requires ci, where ci is a constant.
- There is no concurrency

Run Time of Insertion Sort

Steps	Cost	Times
for j=1 to n-1	c_1	n
key=A[j]	c_2	n-1
i=j-1	c_3	n-1
while i>0 and A[i]>key	c_4	$\sum_{j=1}^{n-1} t_j$
do A[i+1]=A[i]	c_5	$\sum_{j=1}^{n-1} (t_j - 1)$
i=i-1	c_6	$\sum_{j=1}^{n-1} (t_j - 1)$
A[i+1]=key	c_7	(n-1)

The total time required is the sum of that for each statement:

$$T(n)=c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{j=1}^{n-1} t_j + c_5\sum_{j=1}^{n-1}(t_j - 1) + c_6\sum_{j=1}^{n-1}(t_j - 1) + c_7(n-1)$$

Best Case

- **If the array is already sorted:**
  - *While* loop sees in 1 check that A[i]<key and so while loop terminates. Thus t_j=1 and we have:

$$T(n)=c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{j=1}^{n-1} 1 + c_5\sum_{j=1}^{n-1}(1-1) + c_6\sum_{j=1}^{n-1}(1-1) + c_7(n-1)$$

$$=(c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

<u>The run time is thus a linear function of n</u>

Worst Case: The algorithm cannot run slower!

- If the array is arranged in reverse sorted array:
  - *While* loop requires to perform the comparisons with A[j-1] to A[0], that is t_j=j

$$T(n)=c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{j=1}^{n-1} j + c_5\sum_{j=1}^{n-1}(j-1) + c_6\sum_{j=1}^{n-1}(j-1) + c_7(n-1)$$

$$=(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2})n^2 + (c_1 + c_2 + c_3 - \frac{c_4}{2} - \frac{3c_5}{2} - \frac{3c_6}{2})n + (c_5 + c_6 - c_2 - c_3 - c_7)$$

<u>The run time is thus a quadratic function of n</u>

Divide & Conquer Algorithms

- Many types of problems are solvable by reducing a problem of size n into some number a of independent subproblems, each of size $\leq \lceil n/b \rceil$, where $a \geq 1$ and $b > 1$.
- The time complexity to solve such problems is given by a recurrence relation:
  - $T(n) = a\,T(\lceil n/b \rceil) + g(n)$

Time for each subproblem

Time to combine the solutions of the subproblems into a solution of the original problem.

Why the name?

- Divide: This step divides the problem into one or more substances of the same problem of smaller size
- Conquer: Provides solutions to the bigger problem by using the solutions of the smaller problem by some additional work.

Divide and Conquer Examples

- **Binary search:** Break list into 1 sub-problem (smaller list) (so *a*=1) of size $\leq\lceil n/2 \rceil$ (so *b*=2).
  - So $T(n) = T(\lceil n/2 \rceil) + 2$ ($g(n)=c$ constant)
  - g(n)=2, because two comparisons are needed to conquer. One to decide which half of the list to use. Second to decide whether any term in the list remain.

Solving the recurrence

Assume, $n=2^t \Rightarrow t = \log(n)$

$$\begin{aligned}
T(n) &= T(n/2) + 2 \\
&= T(n/4) + 2 + 2 \\
&= T(n/2^2) + 2.2 \\
&= \ldots \\
&= T(n/2^t) + 2.t \\
&= T(1) + 2\log(n) = 1 + 2\log(n) \\
&= O(\log n)
\end{aligned}$$

Merge Sort

$$Assume, n = 2^t$$

$$T(n) = 2T(n/2) + cn$$

$$=2[2T(n/4)+c(n/2)]+cn$$

$$=2^2 T(n/2^2) + 2cn$$

$$=2^t T(n/2^t) + tcn$$

$$=2^t T(1) + tcn$$

$$=n+cn\log(n)=O(n\log n)$$

Best of Luck for End Sems!