

## Homework 1

$$\textcircled{1} \quad T(n) = \begin{cases} 2 & \text{if } n=2 \\ 2T(n/2) + n & \text{if } n=2^k \text{ for } k \geq 1 \end{cases}$$

is  $T(n) = n \lg n$ .

Base case is when  $n=2$ , we have  $n \lg n = 2 \lg 2 = 2 \cdot 1 = 2$

For the inductive step, our inductive hypothesis is that  $T(n/2) = (n/2) \lg(n/2)$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(n/2) \lg(n/2) + n \\ &= n(\lg n - 1) + n \\ &= n \lg n - n + n \\ &= n \lg n \end{aligned}$$

which completes the inductive proof for exact powers of 2.

$\textcircled{2}$  Since it takes  $\Theta(n)$  time in the worst case to insert  $A[n]$  into the sorted array  $A[1..n-1]$ , we get the recurrence

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

Solution is  $T(n) = \underline{\underline{\Theta(n^2)}}$



③

a) Insertion sort takes  $\Theta(k^2)$  time per  $k$ -element list in the worst case. Hence sorting  $n/k$  lists of  $k$  elements each takes  $\Theta(k^2 n/k) = \Theta(nk)$  worst-case time.

b) Extending the 2-list merge all the lists at once would take  $\Theta(n \cdot (n/k)) = \Theta(n^2/k)$  time. [ $n$  from copying each element once into the result list,  $n/k$  from examining  $n/k$  lists at each step to select next item for result list].

We merge the lists pairwise to achieve  $\Theta(n \lg(n/k))$  time merging, merge the resulting lists pairwise, until there's just one list. The pairwise merging  $\Theta(n)$  work at each level, since we are still working on  $n$  elements, even if they are partitioned among sublists. The number of levels, starting with  $n/k$  lists (with  $k$  elements each) and finishing with 1 list (with  $n$  elements), is  $\lceil \lg(n/k) \rceil$ . Hence, the total running time for the merging is  $\Theta(n \lg(n/k))$ .

c) The modified algorithm has the same asymptotic running time as standard merge sort when  $\Theta(nk + n \lg(n/k)) = \Theta(n \lg n)$ .  $k = \Theta(\lg n)$  is the largest asymptotic value of  $k$  as a function of  $n$  that satisfies the condition.

$k$  cannot be more than  $\Theta(\lg n)$  which means it can't have a higher-order term than  $\lg n$ .



otherwise the left-hand expression wouldn't be  $\Theta(n \lg n)$  (because it would have a higher-order term than  $n \lg n$ ). So all we need to do is verify that  $k = \Theta(\lg n)$  works, which we can do by plugging  $k = \lg n$  into  $\Theta(nk + n \lg(n/k)) = \Theta(nk + n \lg n - n \lg k)$  to get

$$\Theta(n \lg n + n \lg n - n \lg \lg n) = \Theta(2n \lg n - n \lg \lg n)$$

which, by taking just the high-order term and ignoring the constant coefficient, equals  $\Theta(n \lg n)$

- d)  $k$  should be the largest list length on which insertion sort is faster than merge sort.

### Insertion Sort Output:-

initial data:

21, 35, 0, 9, 8, 45, 33, 19, 4

sorted data:

0, 4, 8, 9, 19, 21, 33, 35, 45

### List of References

1. <http://bayanbox.ir/view/4177858657730907268/introduction-to-algorithms-3rd-edition.pdf>
2. [http://www.ime.usp.br/~geiser/courses/MAC5711%20-%20An%C3%A1lise%20de%20Algoritmos/Introduction%20to%20Algorithms%20\(Instructor's%20Manual\).pdf](http://www.ime.usp.br/~geiser/courses/MAC5711%20-%20An%C3%A1lise%20de%20Algoritmos/Introduction%20to%20Algorithms%20(Instructor's%20Manual).pdf)
3. <http://www.c4learn.com/c-programs/program-to-implement-insertion-sort-in-c-programming.html>