Chapter 2

November 14, 2022

1 Exercise 2.3-1

Using Figure 2.4 as a model, illustrate the operation of merge sort on the array A= $\{3, 41, 52, 26, 38, 57, 9, 49\}$

3	41	52	26	38	57	9	49
3	41	26	52	38	57	9	49
3	26	41	52	9	38	49	57
3	9	26	38	41	49	52	57

2 Exercise 2.3-2

Rewrite the MERGE procedure so that it does not use sentinels, instead stopping once either array L or R has had all its elements copied back to A and then copying the remainder of the other array back into A.

The following is a rewrite of MERGE which avoids the use of sentinels. Much like MERGE, it begins by copying the subarrays of A to be merged into arrays L and R. At each iteration of the while loop starting on line 13 it selects the next smallest element from either L or R to place into A. It stops if either L or R runs out of elements, at which point it copies the remainder of the other subarray into the remaining spots of A.

3 Exercise 2.3-3

Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2, & \text{if } n = 2. \\ 2T(n/2) + 2, & \text{if } n = 2^k, fork > 1. \end{cases} \text{ is T(n) = n lg(n)} \tag{1}$$

- As given n is power of 2, i.e. $n = 2^k$.
- if k=1, then n=2, T(2)=2 lg(2)
- suppose it is true for k , then we will show it is true for k+1.
- $$T(2^{k+1}) = 2T(\frac{2^{k+1}}{2}) + 2^{k+1}$$
- $$T(2^{k+1}) = 2T(2^k) + 2^{k+1}$$
- $$= 2 * 2^k * lg(2^k) + 2^{k+1}$$
- $$= 2^{k+1}(k) + 2^{k+1}$$
- $$= 2^{k+1}(k+1)$$
- $$= 2^{k+1}lg(2^{k+1})$$
- $$= nlg(n)$$

4 Exercise 2.3-4

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 for the running time of this recursive version of insertion sort.

5 Exercise 2.3-5

Referring back to the searching problem (see Exercise 2.1-3), observe that if the sequence A is sorted, we can check the midpoint of the sequence against and eliminate half of the sequence from further consideration. The binary search algorithm repeats this procedure, halving the size of the remaining portion of the sequence each time. Write pseudocode, either iterative or recursive, for binary search. Argue that the worst-case running time of binary search is ,$\Theta(lgn)$.

```
Bineary_search(a,b,v)
a=1
b=n

if(a>b) do
    return Nil
end if

mid = [a+b/2]
if(m= v)
    then return m
end if

if(m <v)
    return Bineary_search(a,m,v)
end if

return Bineary_search(m+1,b,v)
```

6 Exercise 2.3-6

Observe that the while loop of lines 5–7 of the INSERTION-SORT procedure in Section 2.1 uses a linear search to scan (backward) through the sorted subarray A[1....j-1]. Can we use a binary search (see Exercise 2.3-5) instead to improve the overall worst-case running time of insertion sort to ,$\Theta(lgn)$?

A binary search wouldn't improve the worst-case running time. Insertion sort has to copy each element greater than key into its neighboring spot in the array. Doing a binary search would tell us how many how many elements need to be copied over, but wouldn't rid us of the copying needed to be done.

7 Exercise 2.3-7

Describe a $\Theta(nlgn)$ -time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x?

```
Search_Sum(a,b,x)

i=1
j=n
while (i <j) do
    if(A[i]+A[j] == x)
        return True
    end if
    if(A[i]+A[j]< x) then
        i=i+1
    end if
    if(A[i]+A[j]> x)
        j=j-1
    end if
end while
return False
```