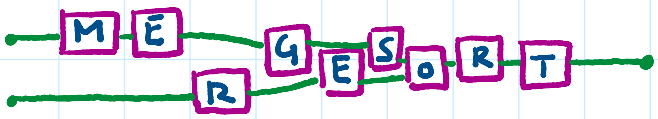
CMSC351-2023-10-04

Wednesday, October 4, 2023 8:29 AM



① Intro: We've seen divide and conquer solve the maximum contiguous sum, can we use D.A.C. to sort?

② Idea: Some observations!

(A) A list of length 1 is sorted!

(B) Sps we have two sorted lists. We can merge them together to create one big sorted list!

ex 1 2 4 10
3 5 6 9 11
} → 1 2 3 4 5 6 9 10 11

These two ideas form the basis of merge sort!

Add'l: If our two sorted lists have a combined total length of n then we can merge them in $\Theta(n)$, by simply picking the smallest from each list to build our new list, emptying out those lists as we go. If one list empties completely, just go through the rest of the other list.

③ All of this ↑ becomes this pseudocode:

```
\\ PRE: A is a list of integers.
function mergesort(A)
    if len(A) > 1
        m = len(A) // 2
        L = A[0,...,m-1]
        R = A[m,...,len(A)-1]
        L = mergesort(L)
        R = mergesort(R)
        \\ Merge L and R back on top of A.
        Lind = 0
        Rind = 0
        Aind = 0
        while Lind < len(L) and Rind < len(R)
            if L[Lind] <= R[Rind]:
                A[Aind] = L[Lind]
                Lind ++
                Aind ++
            else:
                A[Aind] = R[Rind]
                Rind ++
                Aind ++
        end
        while Lind < len(L)
            A[Aind] = L[Lind]
            Lind ++
            Aind ++
        end
        while Rind < len(R)
            A[Aind] = R[Rind]
```

if len(A)=1, A is sorted, return it!

cut in half to make NEW LISTS L and R ← $\Theta(n)$

sort those halves recursively! ← via recursion!

merging two lists together as discussed! $\Theta(n)$ where n = combined lengths

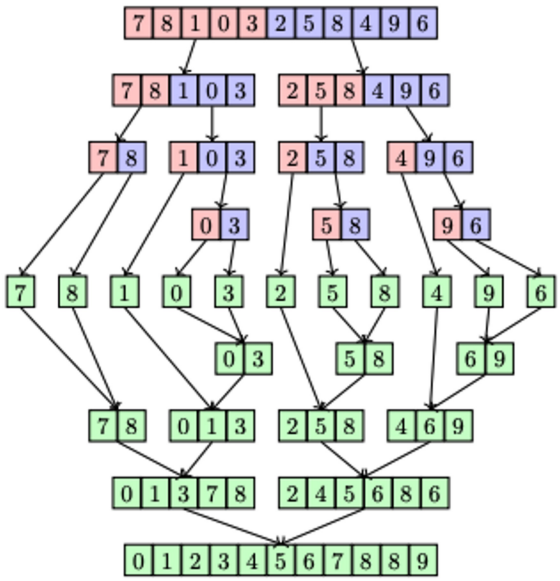
also note the base case, when len(A)=1, is $\Theta(1)$

← note, L, R are merged back on top of A.

```
            Aind ++
        end
        while Rind < len(R)
            A[Aind] = R[Rind]
            Rind ++
            Aind ++
        end
    end
    return(A)
end
\\ POST: A is sorted.
```

n = combined lengths of L, R

④ Fun Picture!



Recursive Splitting!

merging!

⑤ Time Complexity: Let's use the Master Theorem!

Sps $T(n)$ = time required to sort a list of length n.

Looking at the pseudocode:

$$T(n) = \underbrace{2T(\tfrac{n}{2})}_{\text{Rec. calls of lists of half the length}} + \underbrace{\Theta(n)}_{\text{some fctn which is } \Theta(n)}$$

creation of L and R and merging process.

and $T(1)$ = some constant!

Master Theorem! $T(n) = aT(\frac{n}{b}) + f(n)$

Case 1? $f(n) = O(n^1)$ so $c=1$ and $\log_b a = \log_2 2 = 1 > 1 = c$ NO

Case 2? $f(n) = \Theta(n^1)$ so $c=1$ and $\log_b a = \log_2 2 = 1 = 1 = c$ YES

So $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n \lg n)$

So $T(n) = \Theta(n^{\cdot\cdot} \lg n) = \Theta(n \lg n)$

Note: this is best, worst, and avg case!

⑥ Aux Space: B/c we create L and R each time, the aux space is not $\Theta(1)$. what is it?!

Sps $S(n)$ = aux space required to sort a list of length n.

Observe that:

(A) we create L, R and that's $\Theta(n)$ more space. } together $\Theta(n)$

(B) we have some indices, thats $\Theta(1)$

(C) we have two rec. calls to lists of length $\frac{n}{2}$.

So maybe:

$$S(n) = 2S(\tfrac{n}{2}) + \Theta(n) \quad ?$$

wrong b/c first rec. call ends, and its aux mem freed, before the second rec. call. ← f(n)

So $S(n) = S(\frac{n}{2}) + \Theta(n)$

M.T.! Case 1: $O(n)$ so $c=1$ and $\log_2 1 = 0 \neq 1 = c$

Case 2: $\Theta(n)$ so $c=1$ and $\log_2 1 = 0 \neq 1 = c$

Case 3: $\Omega(n)$ so $c=1$ and $\log_2 1 = 0 \langle 1 = c$ Case 3!

So $S(n) = \Theta(f(n)) = \Theta(n)$.

⑦ To finish

(A) Aux Space: $\Theta(n)$

(B) In-Place: No.

(C) Stable: yes b/c of the:

```
if L[Lind] <= R[Rind]:
```

!