Topic-04-maximum-contiguous-sum-Part-2

Tuesday, June 11, 2024 11:51 AM

MAXIMUM CONTIGUOUS SUM - DIVIDE AND CONQUER!

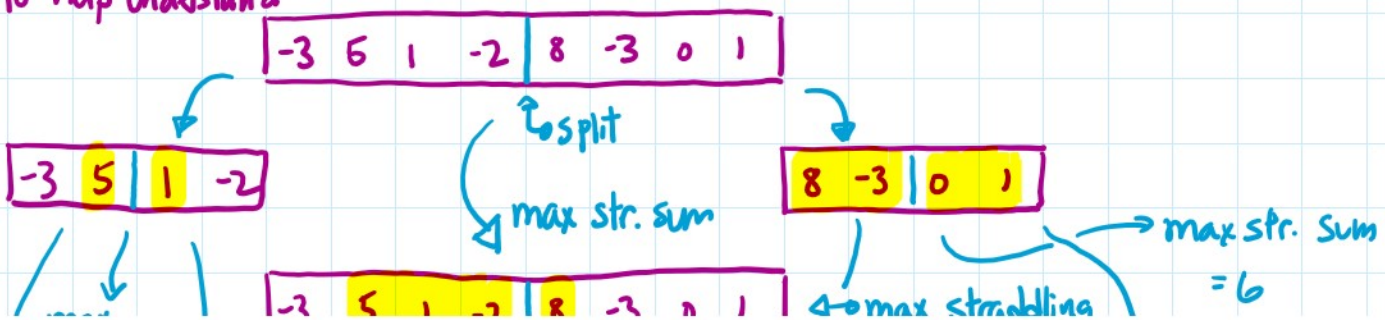
① Divide and Conquer

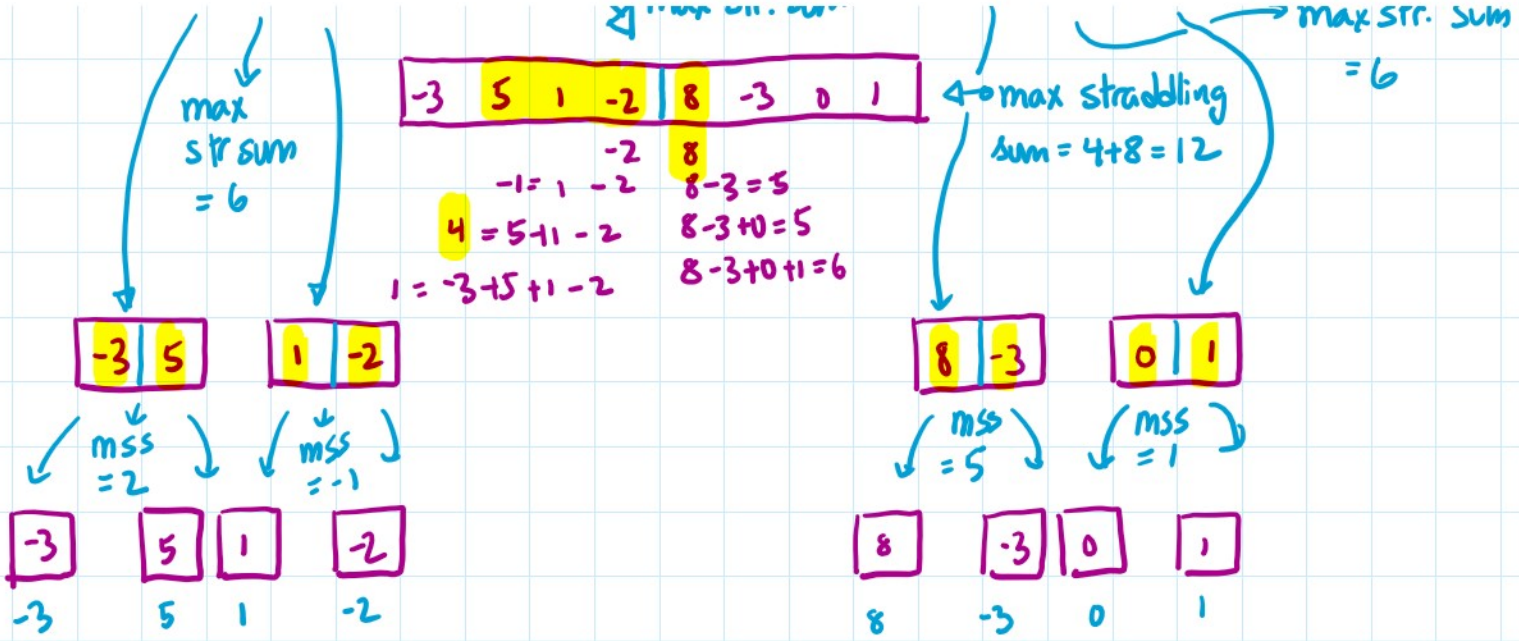
In DAC - we generally split our list into two (or more) sublists (generally equal size, not always).
Do something w/ those sublists - generally recursive.
Combine the results.
Often other work!

② Applying to MCS

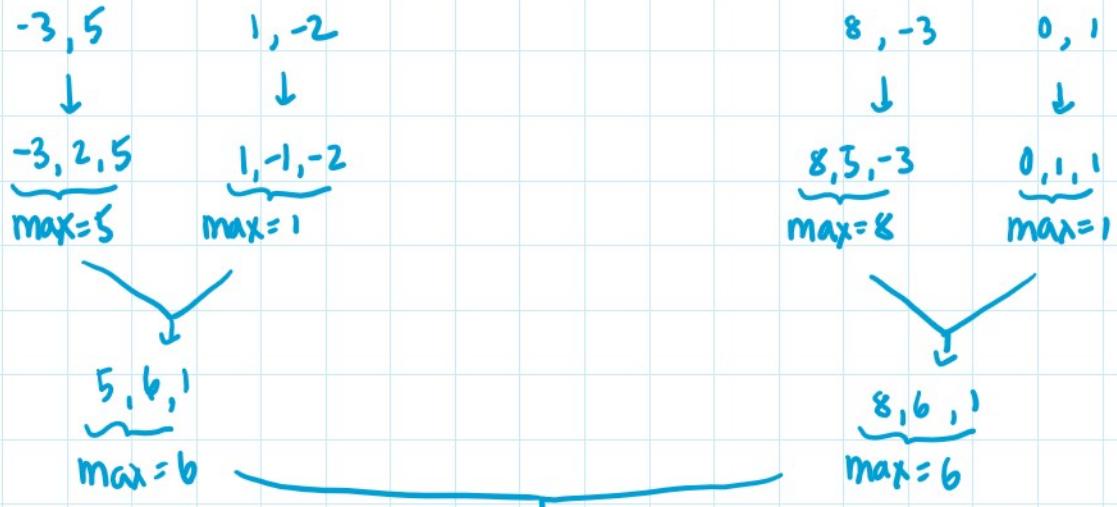
observe: For a list of length = 1, MCS = that single element ⊛
Thus we proceed as follows:
Given a list of length n we:
- divide in half (or as close as possible)
- find MCS via recursion for each of those, noting ⊛ which is where recursion ends
- w/ the list we've split in half we find the maximum straddling sum - max sum which "straddles" the splitting point.
- at each step we take the max from among our recursive calls and our max. straddling sum.

ex to help understand!

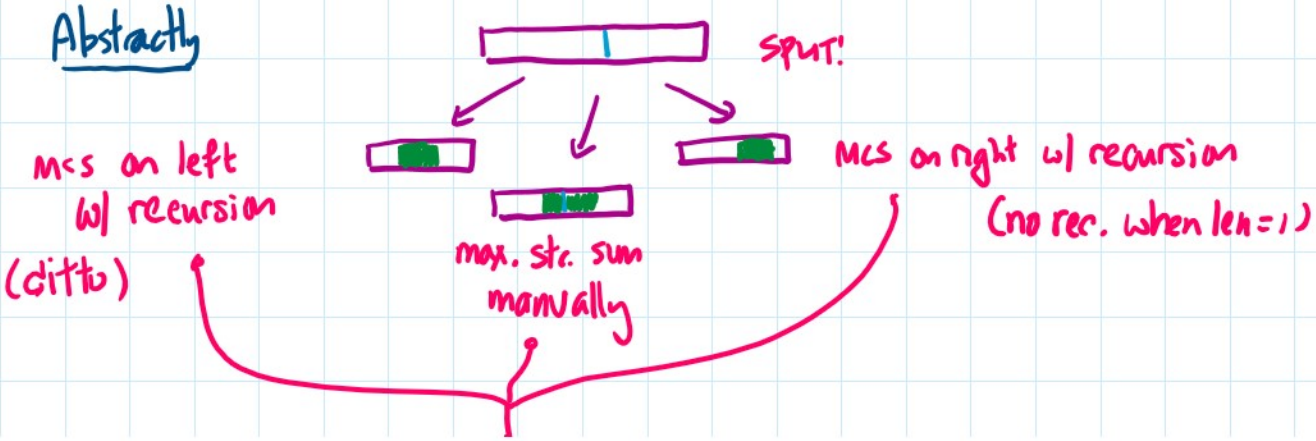


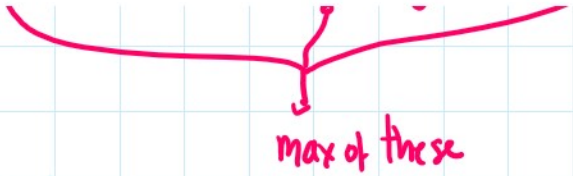


Going back up!



Abstractly





③ Pseudocode!

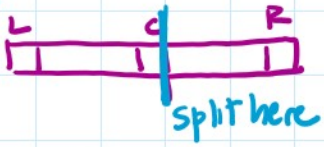
```
function mcs(A,L,R)
    if L == R
        return(A[L])
    else
        C = (L+R) // 2
        Lmax = mcs(A,L,C)
        Rmax = mcs(A,C+1,R)
        \\ Calculate the straddle max
        Lhmax = -infinity
        Lsum = 0
        for i = C downto L                  C − L + 1 times
            Lsum = Lsum + A[i]              } c
            if Lsum > Lhmax
                Lhmax = Lsum
            end
        end
        Rhmax = -infinity
        Rsum = 0
        for i = C+1 to R                    R − (C + 1) + 1 = R − C times
            Rsum = Rsum + A[i]              } c
            if Rsum > Rhmax:
                Rhmax = Rsum
            end
        end
        Smax = Lhmax + Rhmax
        \\ Return the overall max
        return(max([Lmax,Rmax,Smax]))
    end
end
result = mcs(A,0,len(A)-1)
```

List = A. L,R = indices of the part of the list we're looking at

if length = 1, return the element

c splits at C

Recursive calls



Finds the max. straddling sum!

Finds max

Find max

add those

choose maximum!

call on list!

④ Time Complexity

How can we do this w/ a recursive alg?

- At 0th Level of recursion (top level)
  - we do some $\Theta(1)$ things like $C = (L+R)//2$ ↔ ignorable of the cn below
  - we make 2 recursive calls (don't count yet!)
  - we find max straddling sum → do n steps (half on each side)
    - and each takes $\Theta(1)$ time (in fact time c)
  - total time at 0th level essentially cn
- At 1st Level we have 2 lists of length $\frac{n}{2}$
  - w/ each, same as above.
  - total time at 1st level essentially $c(\frac{n}{2}) + c(\frac{n}{2}) = cn$
- At 2nd Level we have 4 lists of length $\frac{n}{4}$

total time at 1st level essentially $c(2)+c(2)=cn$

- At 2nd level we have 4 lists of length $\frac{n}{4}$
  w/ each, same as above
  total time at 2nd level essentially $c(\frac{n}{4})+c(\frac{n}{4})+c(\frac{n}{4})+c(\frac{n}{4})=cn$
- Each level of rec: time = cn

Q: When does this stop?! ✓ stop at some k

A: Levels of rec. go $0,1,2,\ldots,k$
At level k, list length is $\frac{n}{2^k}$
rec. ends when length = 1, so

$$\frac{n}{2^k}=1 \quad \text{so} \quad 2^k=n \quad \text{so} \quad k=\lg n$$

we have levels $0,1,2,\ldots,\lg n$ ← $1+\lg n$ levels total!
Each level takes cn time, for a total of

$$cn(1+\lg n)$$

Thus $T(n)=\Theta(n\lg n)$

Note this is "better" = faster than $\Theta(n^2)$ we had for brute force!

⑤ Can we do still better? Yes!