Problem 1: Finding consecutive elements in an array that maximize sum.

Input: An array $A[1, \ldots, n]$ where each $A[i] \in \mathbb{Z}$.

Output: i, j, s.t. $1 \le i \le j \le n$, and $\sum_{k=i}^{j} A[k]$ is maximum over all i, j.

Note: If $\forall k$, $A[k] \ge 0$, then output $i = 1$, $j = n$.

Note: $A = [-3, 7, 2, -8, 9, -1, 3, -15, 4, -7]$

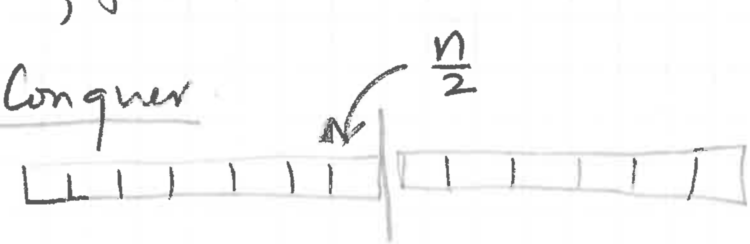
$\underbrace{7, 2, -8, 9, -1, 3}_{12}$

Naive: Try all $1 \le i \le j \le n$ $\quad O(n^3)$

Brute Force

Try all
$i = 1, \; j = 1, 2, 3, \ldots, n$
$i = 2, \; j = 2, 3, 4, \ldots, n$
$\vdots$
$i = n, \; j = n$
$\Big\} \; O(n^2)$

Divide & Conquer



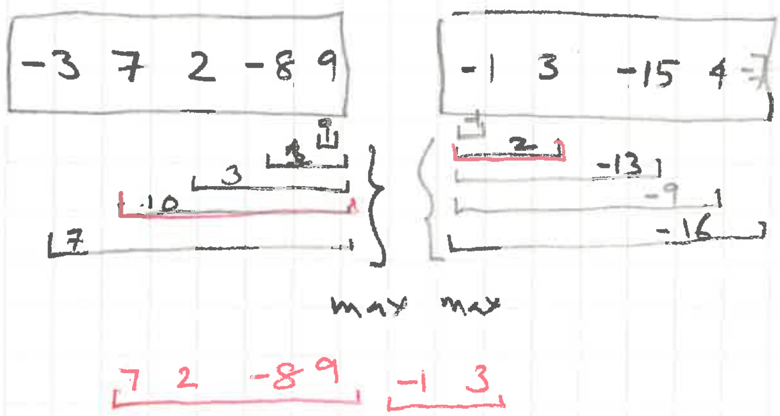
Case 1: $i, j \leq \frac{n}{2}$

Recursive calls

Case 2: $i, j \geq \frac{n}{2}$

Case 3: $i \leq \frac{n}{2}$, $j \geq \frac{n}{2}$ Solution is of Special Type

$O(n)$



See pages 2.1-2.4 for more details

$$\text{Running Time} \equiv T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$
$$= O(n \log n).$$

[KADANE's ALGORITHM

Last One: $\forall k$, $1 \leq k \leq n$, find the best solution ending at k, given the best solution ending at $k-1$.

Let v_k = value of best solution ending at k.

Let v_k^* = value of best solution seen so far.

Given v_{k-1}, v_{k-1}^*; how to find v_k & v_k^*.

KADANE'S ALGORITHM

INPUT: $A[1..n]$ of elements in $\mathbb{Z}$.

OUTPUT: (i,j) such that

$$\sum_{k=i}^{j} A[k] \text{ is maximized for all } 1 \le i \le j \le n$$

ALGORITHM:

Definition: For $k := 0$ to n define

a) v_k = value of best solution ending at k

b) v_k^* = value of best solution for subarray $A[1..k]$.

INITIALIZE $\quad v_0 := -\infty; \quad v_0^* := -\infty$

INCREMENT STEP

For $i := 1$ to n do

$v_i := \text{MAX}[A[i], v_{i-1} + A[i]]$;

$v_i^* := \text{MAX}[v_{i-1}^*, v_i]$

RETURN

Indices corresponding to v_n^*.

$v_0 := -\infty$
$v_0^* := -\infty$

Assume $\exists$ an element in A that is ≥ 0.

for $i := 1$ **to** n **do**

$\quad v_i := \text{MAX}[A[i], A[i] + v_{i-1}]$;

$\quad v_i^* := \text{MAX}[v_{i-1}^*, v_i]$

Example:

		-3	7	2	-8	+9	-1	3	-15	4	-7
v_i	0	-3	7	9	1	10	9	12	-3	4	-3
v_i^*	0	0	7	9	9	10	10	12	12	12	12

— x —

Running Time: $O(n)$.

Why is it correct?

More details on Divide-and-Conquer Algorithm

Partition A in two equal halves. Assume $n = 2^k$, $k > 0$.

$$A = \boxed{1 \quad \cdots \quad \tfrac{n}{2}} \quad \boxed{\tfrac{n}{2}+1 \quad \cdots \quad n}$$

Note: Our objective is to find $1 \le i \le j \le n$

s.t $\sum_{k=i}^{j} A[i]$ is maximized.

i & j must satisfy exactly one of the following

Case 1. $1 \le i \le j \le \frac{n}{2}$ [Both i & j in left half]

Case 2. $\frac{n}{2}+1 \le i \le j \le n$ [Both i & j in right half]

Case 3: $1 \le i \le \frac{n}{2}$ and $\frac{n}{2}+1 \le j \le n$

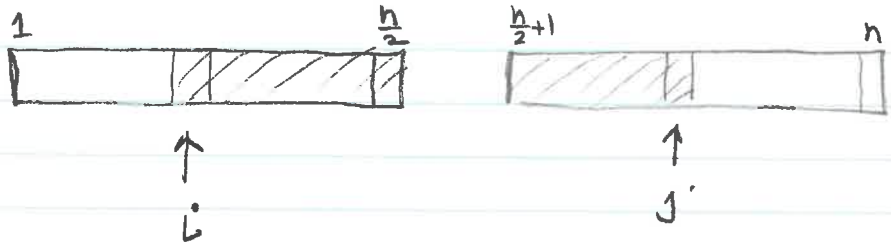
[i in left half, j in right half]

We do not know where i & j are, so we try all three cases.

For Case 1 & 2, we appeal to recursion.

Case 3 has a special structure as explained on the next page.

Case 3:



In Case 3, we know $1 \le i \le \frac{n}{2}$ and $\frac{n}{2}+1 \le j \le n$.

We are interested to find what values of i & j will maximize the Sum

$$\sum_{\substack{k=i,\\ i \le \frac{n}{2}\\ j \ge \frac{n}{2}+1}}^{j} A[k].$$

Note:

$$\sum_{\substack{k=i,\ i\le \frac{n}{2},\\ j \ge \frac{n}{2}+1}}^{j} A[k] = \sum_{k=i}^{\frac{n}{2}} A[k] + \sum_{k=\frac{n}{2}+1}^{j} A[k].$$

Thus, we want to find index i such that $A[i] + A[i+1] + A[i+2] + \cdots + A[\frac{n}{2}]$ is maximum

Find index j such that $A[\frac{n}{2}+1] + \cdots + A[j]$ is maximum.

Both of these can be found by performing a simple scan in their respective Subarrays.

Analysis of Divide & Conquer algorithm

Our solution consists of the best solution among the best solution obtained from Cases 1, 2, & 3. If $T(n)$ represents the time to solve the problem on an array of size n then the analysis of the above algorithm can be done as follows.

$$T(1) = 1.$$

$$T(n) = \underbrace{T\left(\frac{n}{2}\right)}_{\text{Time for Case 1}} + \underbrace{T\left(\frac{n}{2}\right)}_{\text{Time for Case 2}} + \underbrace{O(n)}_{\text{Time for Case 3.}}$$

$$= 2T\left(\frac{n}{2}\right) + O(n)$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2O\left(\frac{n}{2}\right) + O(n)$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^2 O\left(\frac{n}{2^2}\right) + 2O\left(\frac{n}{2}\right) + O(n).$$

$$\cdots\cdots$$

$$= 2^l T\left(\frac{n}{2^l}\right) + 2^{l-1} O\left(\frac{n}{2^{l-1}}\right) + 2^{l-2} O\left(\frac{n}{2^{l-2}}\right) + \cdots + O(n) \quad \text{— (I)}$$

Since $n = 2^{\ell}$ (by assumption)

$\iff \ell = \log_2 n$.

Thus Equation I can be expressed as

$$T(n) = 2^{\ell} T(1) + \underbrace{O(n) + O(n) + \cdots + O(n)}_{\log_2 n \text{ times}}$$

Since $T(1) = 1$, and $n = 2^{\ell}$, we obtain

$$T(n) = O(n \log n).$$