Maximum Contiguous Subsequence Problem

Array of $n$ integers

\[-1 4 -3 5 -2 -1 2 6 -2 1\]

Goal: Find contiguous piece with max sum.
3 cases

- Opt sol is on left half
- Opt sol is on right half
- Opt sol can cross boundary
divide in middle

opt sol in left half  

opt sol in right half

-1 4 -3 5 -2 -1 26 21 3 -5 2
2 3 -1 2 -3 -1 28 6 7 10 5 7

running sum moving from left of middle  

running sum moving from right of middle

Opt sol is in left half, in right half or crosses two halves
Correctness Argument for divide & conquer alg

1. (base step) Return the correct answer in the termination condition(s)

2. (inductive step) \textbf{IF} you get the right answer from recursive calls \textbf{THEN} you return the correct ans for the given problem.
n=1 or n=2 \rightarrow \text{Termination condition. Get right answer. (By 1.)}

\begin{align*}
n=4 & \quad \text{left } n=2 \checkmark \end{align*}
\begin{align*}
& \quad \text{right } n=2 \checkmark \quad \text{by 2. get right answer } n=4
\end{align*}

\begin{align*}
n=8 & \quad \text{left } n=4 \checkmark \quad \text{by 2. get right answer } n=8
\end{align*}
\begin{align*}
& \quad \text{right } n=4 \checkmark \quad \text{by 2. get right answer } n=8
\end{align*}
Which Algorithm is Best

Suppose we think of 4-5 algorithms for a problem & we've argued they always yield the correct answer.

How can we determine which will be the most efficient, especially as the input size gets large?

Why not implement all of them & run them to see how long they take?
Problems
- Takes a lot of time to implement & test
- Time complexity often depends on input itself
- What data size should you use?
What does time complexity depend on?

- Input itself
- Input size \((n = 100 \text{ vs } n = 1,000,000)\)
- Hardware (computer) \(\{\text{dependent}\} \)
- Compiler \(\{\text{machine dependent}\} \)

Focus on a machine-independent analysis.
Insertion Sort

5, 1, 7, 3, 4

1, 5, 7, 3, 4

1, 5, 7, 3, 4

1, 3, 5, 7, 4

\[C \cdot N^2\]

7, 5, 4, 3, 1

\[C \cdot N\]

1, 3, 7, 5, 7

\[N-1 \text{ comparisons}\]

\[C' \cdot N\]
Asymptotic Time Complexity

machine independent, rough measure of time complexity as a function of input size \( n \)

"Back of the envelope" calculation
What are we going to measure?

# of statements (lines of code)
executed

need to be careful that all
statements in your high-level
language take "roughly" the
same time.
Input size vs Execution time

- Insertion sort: $O(n^2)$
- Mergesort: $O(n \log n)$
Input Size vs Lines of Code

lines of code executed: merge sort vs. insertion sort

number of elements in array
How do we account for dependencies in the input?

Worst-case analysis — consider the input of the given size that is slowest

Expected-case (average-case) analysis — assume some distribution over the data
$n^2$ vs $n \log n$
How can we figure out how many statements are used by divide-and-conquer closest pair alg for n points?

For Brute Force alg, we saw it was $C \cdot n^2$ (quadratic)
Define Function

\[ T(n) = \# \text{ statements executed by divide & conquer closest pair alg for worst-case input of } n \text{ points} \]

\[ T(1) = \text{ constant} \]

\[ T(2) = \text{ constant} \]
How about the case when $n \geq 3$?

For now assume $n$ is a power of 2.

\[
\begin{array}{c|c|c}
\frac{n}{2} & \frac{n}{2} \\
\hline
\text{Splitting} + T(\frac{n}{2}) + T(\frac{n}{2}) + \overbrace{C \cdot n}^{\text{Combining}}
\end{array}
\]
For $n \geq 4$ (n is a power of 2)

$$T(n) = 2T\left(\frac{n}{2}\right) + C_1n + C_2n$$

Recurrence Relation (Equation)

$T(2) = \text{constant}$

$T(n) = 2T\left(\frac{n}{2}\right) + C \cdot n$
\[ T(n) = 2T(n/2) + Cn \]

Recurrence Tree

\( n = 16 \)

\[ \log_2 n - 1 \]

Original input

Split

Combine

\[ 16 \]

\[ 8 \]

\[ 4 \]

Combined

\[ 2 \]

Combine

\[ 2 \]

\[ 2 \]

Combine

\[ 2 \]

Combine

\[ 2 \]
\[ T(n) = (\log_2 n - 1) \cdot n + \frac{c \cdot n}{2} \]

\[ = c \cdot n \log_2 n - \frac{c \cdot n}{2} \]

low order

(not very significant in its contribution as \( n \) grows larger)