Maximum Subarray (Bentley's Problem)

Problem:

- **Find the contiguous subarray within an array which has the largest sum.**
- For example, given the array $[31, -41, 59, 26, -53, 58, 97, -93, -23, 84]$, the subarray $[59, 26, -53, 58, 97]$ has the maximum sum of 187.
- Given $A[1 \ldots n]$, find

$$\max_{1 \le i \le j \le n} \sum_{k=i}^{j} A[k]$$

Solution1: Brute force

- Evaluation all possible subarrays and select the one with largest sum.
- Psudocode:

```
max := 0;
for i := 1 to n do
for j := i to n do
  sum := 0;
  for k := i to j do
    sum := sum + A[k];
  if sum > max then max := sum;
```

- Time complexity: $\Theta(n^3)$

$$\begin{aligned} T(n) &= \sum_{i=0}^{n}\left(\sum_{j=i}^{n}\left(\sum_{k=i}^{j} 1\right)\right) \\ &= \frac{1}{6}n(n^2 + 3n + 2) \in \Theta(n^3) \end{aligned}$$

Solution2

- Solution 1 computes the sum for every subarray and initilizes `sum=0` for every `j` loop. We can avoid doing this by only use one sum for each differen `i`
- Psudocode:

```
max := 0;
for i := 1 to n do
sum := 0;
for j := i to n do
  sum := sum + A[j];
  if sum > max then max := sum;
```

- Time complexity: $\Theta(n^2)$

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{n}\left(\sum_{j=i}^{n} c\right) \\
&= \frac{1}{2}cn(n+1) \in \Theta(n^2)
\end{aligned}
$$

Solution2b: some pre-computation

- Based on solution2, we can improve a bit by adding some precomputation
  - If $B[i] = A[1] + \cdots + A[i]$, then we know
  - For any subarray $A[i..j]$ of A: $A[i..j] = B[j] - B[i]$
- Psudocode

```
B[0] := 0;
for i := 1 to n do
  B[i] := B[i-1] + A[i];
max := 0
for i := i to n do
  if B[j] - B[i-1] > max then
    max := B[j] - B[i-1]
```

- Time complexity: $\Theta(n^2)$

Solution3: Divide and Conquer

- We can divide the array into two "equally-sized" parts.
  - The maximum solution will be either entirely one of the two parts, or it must corss the partition line.
- Therefore, we can recursively find the max values on left `maxL` and right `maxR` part, and compare them with the `maxM`, the maximum subarray that crosses the partition border.
- The solution will then be $\max\{\text{maxL,maxR, maxM}\}$
- Pseudocode:

```
recursive maxsum(lo, hi)
  if lo > hi return 0;
```

```
  if lo = hi return max(0, A[low]);

  mid := (lo + hi)/2;

  leftmax := sum := 0;
  for i := mid downto mid
    sum = sum + A[i];
    leftmax := max(leftmax, sum);

  rightmax := sum := 0;
  for i := mid+1 to hi
    sum := sum + A[i];
    rightmax := max(rightmax, sum);

  return max(leftmax+rightmax, maxsum(lo, mid), maxsum(mid+1, hi))
```

- Time complexity: $\Theta(n \log(n))$, *To be proved*

Solution4: $\Theta(n)$ alogrithm

- Consider $A[1..i+1], 1 \leq i+1 \leq n$,
- If given a the value of the maximum subarray for $A[1..i]$, we know for sure that one of the following happens:
  - The maximum subarray for $A[1..i+1]$ has the same value as the solution for $A[1..i]$, OR
  - The maximum subarray for $A[1..i+1]$ is a tail of $A[1..i+1]$. This happens when the last element $A[i+1]$ plus the maximum tail of $A[1..i]$ is greater than the solution for $A[1..i]$
- Pseudocode

```
maxsol := 0; tail := 0;
for i := 1 to n do
  tail := max(tail + A[i], 0);
  maxsol := max(maxsol, tail)
```

- Time complexity: $\Theta(n)$