The Maximum Subarray Problem

COMS10017 - Algorithms 1

Dr Christian Konrad

Generalizing the Analysis

Divide and Conquer Algorithm:

Generalizing the Analysis

Divide and Conquer Algorithm:

Let **A** be a divide and conquer algorithm with the following properties:

Divide and Conquer Algorithm:

Let **A** be a divide and conquer algorithm with the following properties:

1. **A** performs two recursive calls on input sizes at most $n/2$

Generalizing the Analysis

Divide and Conquer Algorithm:

Let **A** be a divide and conquer algorithm with the following properties:

1. **A** performs two recursive calls on input sizes at most $n/2$
2. The combine operation in **A** takes $O(n)$ time

Generalizing the Analysis

Divide and Conquer Algorithm:

Let **A** be a divide and conquer algorithm with the following properties:

1. **A** performs two recursive calls on input sizes at most $n/2$
2. The combine operation in **A** takes $O(n)$ time

Then:

Generalizing the Analysis

Divide and Conquer Algorithm:

Let **A** be a divide and conquer algorithm with the following properties:

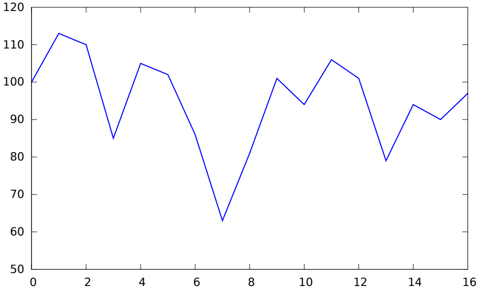
1. **A** performs two recursive calls on input sizes at most $n/2$
2. The combine operation in **A** takes $O(n)$ time

Then:

$$\boxed{\textbf{A} \text{ has a runtime of } O(n \log n)\ .}$$

Buy Low, Sell High Problem

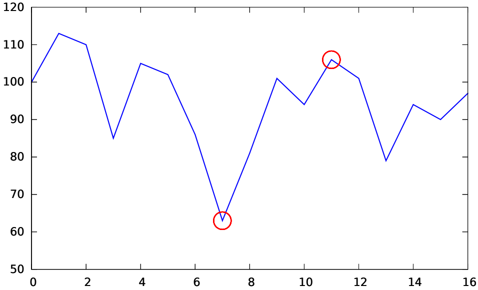
- **Input:** An array of n integers
- **Output:** Indices $0 \leq i < j \leq n-1$ such that $A[j] - A[i]$ is maximized



Maximum Subarray Problem

Buy Low, Sell High Problem

- **Input:** An array of n integers
- **Output:** Indices $0 \leq i < j \leq n-1$ such that $A[j] - A[i]$ is maximized



Focus on Array of Changes:

Day	0	1	2	3	4	5	6	7	8	9	10	11
$	100	113	110	85	105	102	86	63	81	101	94	106
Δ		13	-3	-25	20	-3	-16	-23	18	20	-7	12

Maximum Subarray Problem

Focus on Array of Changes:

Day	0	1	2	3	4	5	6	7	8	9	10	11
$	100	113	110	85	105	102	86	63	81	101	94	106
Δ		13	-3	-25	20	-3	-16	-23	18	20	-7	12

Focus on Array of Changes:

Day	0	1	2	3	4	5	6	7	8	9	10	11
$	100	113	110	85	105	102	86	63	81	101	94	106
Δ		13	-3	-25	20	-3	-16	-23	18	20	-7	12

Maximum Subarray Problem

- **Input:** Array A of n numbers
- **Output:** Indices $0 \leq i \leq j \leq n-1$ such that $\sum_{l=i}^{j} A[l]$ is maximum.

Maximum Subarray Problem

Focus on Array of Changes:

Day	0	1	2	3	4	5	6	7	8	9	10	11
$	100	113	110	85	105	102	86	63	81	101	94	106
Δ		13	-3	-25	20	-3	-16	-23	18	20	-7	12

Maximum Subarray Problem

- **Input:** Array A of n numbers
- **Output:** Indices $0 \leq i \leq j \leq n-1$ such that $\sum_{l=i}^{j} A[l]$ is maximum.

Trivial Solution: $O(n^3)$ runtime

Maximum Subarray Problem

Focus on Array of Changes:

Day	0	1	2	3	4	5	6	7	8	9	10	11
\$	100	113	110	85	105	102	86	63	81	101	94	106
Δ		13	-3	-25	20	-3	-16	-23	18	20	-7	12

Maximum Subarray Problem

- **Input:** Array A of n numbers
- **Output:** Indices $0 \leq i \leq j \leq n-1$ such that $\sum_{l=i}^{j} A[l]$ is maximum.

Trivial Solution: $O(n^3)$ runtime

- Compute subarrays for every pair i, j

Maximum Subarray Problem

Focus on Array of Changes:

Day	0	1	2	3	4	5	6	7	8	9	10	11
\$	100	113	110	85	105	102	86	63	81	101	94	106
Δ		13	-3	-25	20	-3	-16	-23	18	20	-7	12

Maximum Subarray Problem

- **Input:** Array A of n numbers
- **Output:** Indices $0 \leq i \leq j \leq n-1$ such that $\sum_{l=i}^{j} A[l]$ is maximum.

Trivial Solution: $O(n^3)$ runtime

- Compute subarrays for every pair i, j
- There are $O(n^2)$ pairs, computing the sum takes time $O(n)$.

Divide and Conquer Algorithm for Maximum Subarray

Divide and Conquer Algorithm for Maximum Subarray

Divide and Conquer:

Divide and Conquer Algorithm for Maximum Subarray

Divide and Conquer:
Compute maximum subarrays in left and right halves of initial array

$$A = L \circ R$$

Divide and Conquer Algorithm for Maximum Subarray

Divide and Conquer:
Compute maximum subarrays in left and right halves of initial array

$$A = L \circ R$$

Combine:

Divide and Conquer Algorithm for Maximum Subarray

Divide and Conquer:
Compute maximum subarrays in left and right halves of initial array

$$A = L \circ R$$

Combine:
Given maximum subarrays in L and R, we need to compute maximum subarray in A

Divide and Conquer Algorithm for Maximum Subarray

Divide and Conquer:
Compute maximum subarrays in left and right halves of initial array

$$A = L \circ R$$

Combine:
Given maximum subarrays in L and R, we need to compute maximum subarray in A

Three cases:

Divide and Conquer Algorithm for Maximum Subarray

Divide and Conquer:
Compute maximum subarrays in left and right halves of initial array

$$A = L \circ R$$

Combine:
Given maximum subarrays in L and R, we need to compute maximum subarray in A

Three cases:

1. Maximum subarray is entirely included in L ✓

Divide and Conquer Algorithm for Maximum Subarray

Divide and Conquer:
Compute maximum subarrays in left and right halves of initial array

$$A = L \circ R$$

Combine:
Given maximum subarrays in L and R, we need to compute maximum subarray in A

Three cases:

1. Maximum subarray is entirely included in L ✓
2. Maximum subarray is entirely included in R ✓

Divide and Conquer Algorithm for Maximum Subarray

Divide and Conquer:
Compute maximum subarrays in left and right halves of initial array

$$A = L \circ R$$

Combine:
Given maximum subarrays in L and R, we need to compute maximum subarray in A

Three cases:

1. Maximum subarray is entirely included in L ✓
2. Maximum subarray is entirely included in R ✓
3. Maximum subarray crosses midpoint, i.e., i is included in L and j is included in R

Divide and Conquer Algorithm for Maximum Subarray

Maximum Subarray Crosses Midpoint:

Divide and Conquer Algorithm for Maximum Subarray

Maximum Subarray Crosses Midpoint:

- Find maximum subarray $A[i, j]$ such that $i \leq \frac{n}{2}$ and $j > \frac{n}{2}$ (assume that n is even)

Divide and Conquer Algorithm for Maximum Subarray

Maximum Subarray Crosses Midpoint:

- Find maximum subarray $A[i, j]$ such that $i \leq \frac{n}{2}$ and $j > \frac{n}{2}$ (assume that n is even)
- Observe that: $\sum_{l=i}^{j} A[l] = \sum_{l=i}^{\frac{n}{2}} A[i] + \sum_{l=\frac{n}{2}+1}^{j} A[l]$.

Divide and Conquer Algorithm for Maximum Subarray

Maximum Subarray Crosses Midpoint:

- Find maximum subarray $A[i, j]$ such that $i \leq \frac{n}{2}$ and $j > \frac{n}{2}$ (assume that n is even)
- Observe that: $\sum_{l=i}^{j} A[l] = \sum_{l=i}^{\frac{n}{2}} A[i] + \sum_{l=\frac{n}{2}+1}^{j} A[l]$.

Two Independent Subproblems:

Divide and Conquer Algorithm for Maximum Subarray

Maximum Subarray Crosses Midpoint:

- Find maximum subarray $A[i,j]$ such that $i \leq \frac{n}{2}$ and $j > \frac{n}{2}$ (assume that n is even)
- Observe that: $\sum_{l=i}^{j} A[l] = \sum_{l=i}^{\frac{n}{2}} A[i] + \sum_{l=\frac{n}{2}+1}^{j} A[l]$.

Two Independent Subproblems:

- Find index i such that $\sum_{l=i}^{\frac{n}{2}} A[i]$ is maximized

Divide and Conquer Algorithm for Maximum Subarray

Maximum Subarray Crosses Midpoint:

- Find maximum subarray $A[i,j]$ such that $i \leq \frac{n}{2}$ and $j > \frac{n}{2}$ (assume that n is even)
- Observe that: $\sum_{l=i}^{j} A[l] = \sum_{l=i}^{\frac{n}{2}} A[i] + \sum_{l=\frac{n}{2}+1}^{j} A[l]$.

Two Independent Subproblems:

- Find index i such that $\sum_{l=i}^{\frac{n}{2}} A[i]$ is maximized
- Find index j such that $\sum_{l=\frac{n}{2}+1}^{j} A[l]$ is maximized

Divide and Conquer Algorithm for Maximum Subarray

Maximum Subarray Crosses Midpoint:

- Find maximum subarray $A[i,j]$ such that $i \leq \frac{n}{2}$ and $j > \frac{n}{2}$ (assume that n is even)
- Observe that: $\sum_{l=i}^{j} A[l] = \sum_{l=i}^{\frac{n}{2}} A[i] + \sum_{l=\frac{n}{2}+1}^{j} A[l]$.

Two Independent Subproblems:

- Find index i such that $\sum_{l=i}^{\frac{n}{2}} A[i]$ is maximized
- Find index j such that $\sum_{l=\frac{n}{2}+1}^{j} A[l]$ is maximized

We can solve these subproblems in time $O(n)$. (how?)

Maximum Subarray Problem - Summary

Require: Array A of n numbers
  if $n = 1$ **then**
    return A
  Recursively compute max. subarray S_1 in $A[0, \lfloor \frac{n}{2} \rfloor]$
  Recursively compute max. subarray S_2 in $A[\lfloor \frac{n}{2} \rfloor + 1, n - 1]$
  Compute maximum subarray S_3 that crosses midpoint
  return Heaviest of the three subarrays S_1, S_2, S_3

Recursive Algorithm for the Maximum Subarray Problem

Maximum Subarray Problem - Summary

Require: Array A of n numbers
  if $n = 1$ **then**
    return A
  Recursively compute max. subarray S_1 in $A[0, \lfloor \frac{n}{2} \rfloor]$
  Recursively compute max. subarray S_2 in $A[\lfloor \frac{n}{2} \rfloor + 1, n - 1]$
  Compute maximum subarray S_3 that crosses midpoint
  return Heaviest of the three subarrays S_1, S_2, S_3

Recursive Algorithm for the Maximum Subarray Problem

Analysis:

Maximum Subarray Problem - Summary

Require: Array A of n numbers
  if $n = 1$ **then**
    return A
  Recursively compute max. subarray S_1 in $A[0, \lfloor \frac{n}{2} \rfloor]$
  Recursively compute max. subarray S_2 in $A[\lfloor \frac{n}{2} \rfloor + 1, n - 1]$
  Compute maximum subarray S_3 that crosses midpoint
  return Heaviest of the three subarrays S_1, S_2, S_3

Recursive Algorithm for the Maximum Subarray Problem

Analysis:

- Two recursive calls with inputs that are only half the size

Maximum Subarray Problem - Summary

Require: Array A of n numbers
  if $n = 1$ **then**
    return A
  Recursively compute max. subarray S_1 in $A[0, \lfloor \frac{n}{2} \rfloor]$
  Recursively compute max. subarray S_2 in $A[\lfloor \frac{n}{2} \rfloor + 1, n-1]$
  Compute maximum subarray S_3 that crosses midpoint
  return Heaviest of the three subarrays S_1, S_2, S_3

Recursive Algorithm for the Maximum Subarray Problem

Analysis:

- Two recursive calls with inputs that are only half the size
- Conquer step requires $O(n)$ time

Maximum Subarray Problem - Summary

Require: Array A of n numbers
  if $n = 1$ **then**
    return A
  Recursively compute max. subarray S_1 in $A[0, \lfloor \frac{n}{2} \rfloor]$
  Recursively compute max. subarray S_2 in $A[\lfloor \frac{n}{2} \rfloor + 1, n - 1]$
  Compute maximum subarray S_3 that crosses midpoint
  return Heaviest of the three subarrays S_1, S_2, S_3

Recursive Algorithm for the Maximum Subarray Problem

Analysis:

- Two recursive calls with inputs that are only half the size
- Conquer step requires $O(n)$ time
- Identical to Merge Sort, runtime $O(n \log n)$!